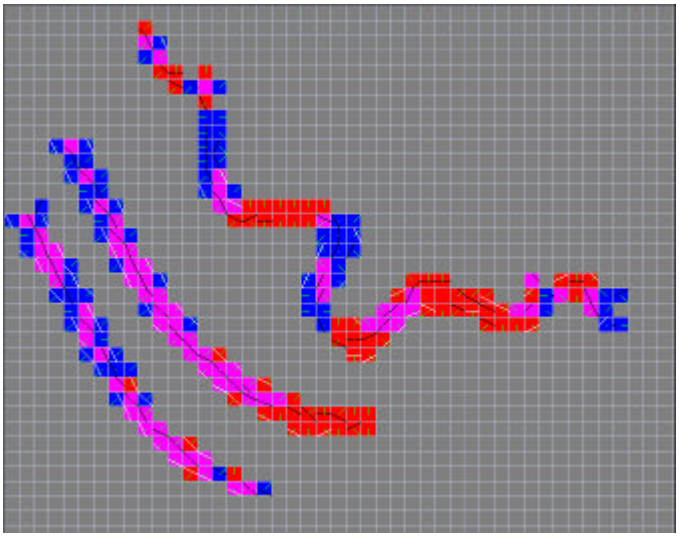
FOI-R--0499--SE
May 2002
ISSN 1650-1942
Scientific report

Mats Sjövall

Object and Feature Recognition in a Digital Terrain Model



Command and Control Systems
SE-581 11 Linköping

SWEDISH DEFENCE RESEARCH AGENCY

Command and Control Systems
P.O. Box 1165
SE-581 11 Linköping

FOI-R--0499--SE

May 2002

ISSN 1650-1942

Scientific report

Mats Sjövall

Object and Feature Recognition in a Digital Terrain Model

Issuing organization
FOI – Swedish Defence Research Agency
Command and Control Systems
P.O. Box 1165
SE-581 11 Linköping

Report number, ISRN
FOI-R--0499--SE

Report type
Scientific report

Research area code
4. C4ISR

Month year
May 2002

Project no.
E7030

Customers code
5. Commissioned Research

Sub area code
42 Surveillance Sensors

Author/s (editor/s)
Mats Sjövall

Project manager
Erland Jungert

Approved by
Lennart Nyström

Sponsoring agency
The Swedish Armed Forces

Scientifically and technically responsible
Erland Jungert

Report title
Object and Feature Recognition in a Digital Terrain Model

Abstract (not more than 200 words)
A method for object and feature recognition is useful in many different areas. The most important application is perhaps simulations and other types of 3D-visualisations where there is a need for determining location and properties of a large number of different objects. Various terrain analysis problems could be solved such as drivability, where the detection of flat areas could be used to determine whether the terrain is suitable for driving vehicles, or possibly for landing-sites for helicopters.
In this Master Thesis, a method for finding terrain-objects using a symbolic structure has been devised. The method is able to find long narrow, straight and winding ditches as well as ridges, hills and roads. This is accomplished by finding separate cross-sections unique to the terrain-object and then connecting these to complete objects. The cross-sections are found using patternmatching techniques.
The evalutation of the algorithms has been made on a data-set from a laser-radar surveillance done with a TopEyeTM system.

Keywords
Digital terrain model, laser radar, symbolic structure, drivability, ditch, hill

Further bibliographic information

Language English

ISSN 1650-1942

Pages 48 p.

Price acc. to pricelist

Utgivare	Rapportnummer, ISRN	Klassificering
Totalförsvarets Forskningsinstitut - FOI Ledningssystem Box 1165 581 11 Linköping	FOI-R--0499--SE	Vetenskaplig rapport
	Forskningsområde 4. Spaning och ledning	
	Månad, år Maj 2002	**Projektnummer** E7030
	Verksamhetsgren 5. Uppdragsfinansierad verksamhet	
	Delområde 42 Spaningssensorer	

Författare/redaktör	Projektledare
Mats Sjövall	Erland Jungert
	Godkänd av Lennart Nyström
	Uppdragsgivare/kundbeteckning Försvarsmakten
	Tekniskt och/eller vetenskapligt ansvarig Erland Jungert

Rapportens titel (i översättning)

Objekt- och särdragsigenkänning i en digital terrängmodell

Sammanfattning (högst 200 ord)

En metod för objekt- och särdragsigenkänning är användbar i många olika sammanhang. Den viktigaste tillämpningen är kanske simuleringar och andra typer av 3D-visualiseringar där det finns ett behov att bestämma position och egenskaper för ett stort antal olika objekt. Olika terränganalysproblem kan lösas, t.ex. framkomlighet där upptäckt av platta ytor kan användas för att bestämma om terrängen är lämplig för att framföra fordon och om det är möjligt att landa med helikopter.

I detta examensarbete har en metod för att hitta terrängobjekt som använder en symbolisk struktur tagits fram. Metoden kan hitta långa smala, raka och kurviga diken såväl som åsar, kullar och vägar. Detta åstadkoms genom att metoden hittar olika terrängsegment som är unika för terrängobjektet och sedan kopplar ihop dessa till kompletta objekt. Segmenten hittas med hjälp av mönstermatchningstekniker.

Utvärderingen av algoritmerna har gjorts på data från en flygning med laserradarsystemet TopEye™.

Nyckelord

Digital terrängmodell, laserradar, symbolisk struktur, framkomlighet, dike, kulle

Övriga bibliografiska uppgifter	Språk Engelska
ISSN 1650-1942	**Antal sidor:** 48 s.
Distribution enligt missiv	**Pris: Enligt prislista**

Contents

Chapter 1 - Introduction

1.1 Background

A method for object and feature recognition is useful in many different areas. The most important application is perhaps simulations and other types of 3D-visualisations where there is a need for determining location and properties of a large number of different objects.
Various terrain analysis problems could be solved such as drivability, where the detection of flat areas could be used to determine whether the terrain is suitable for driving vehicles, or possibly for landing-sites for helicopters.
Work has been done towards the design of a query-language for terrain-objects where feature-recognition is a vital part. If an efficient way of detecting objects can be found, this can be used to quickly create an index of possible objects in an area. Once the object has been found some more precise method could be used to determine the details of the object. The maximum size of the objects that are of interest here is up to a few hundred meters.
A very detailed high-resolution terrain model will be obtained by using an airborne laser scanner. Identification of terrain objects in this vast data-set is difficult and very resource-heavy. To represent the data-set more efficiently a symbolic structure (described in [2]) has been defined on which this work will be based.

1.2 Objectives

In previous work [1] an automated process for finding long, thin terrain-objects using a symbolic structure (the categories described in Section 1.4) has been devised. That work was somewhat limited in that only one simple test-filter that finds ditches was developed.
The objective of this work is to determine if this method can be used to find other types of terrain-objects, and possibly refine the filtering-process.

1.3 Laser Radar

The data used in this work was collected using an active sensor called a laser radar. The measurements were performed using a system called TopEye^{TM}, a topological

survey system carried by a helicopter. The system uses the laser radar to measure the distance to the ground, and by using the known position of the helicopter (using GPS), a coordinate for the measured point can be computed. In addition, the system receives a reflectance-value which is the magnitude of the reflected laser-beam, and this value will vary for different materials at the measured point.

1.4 Categories

Categories are one way to describe the terrain in a symbolic way. This is done by partitioning the topological data into quadratic tiles and for each tile matching the data to a specific category. Each tile is approximated by either one plane or by two planes, in which case the category has one of the basic category-forms shown illustrated in Figure 1.1. How this classification is done is described further in [3].

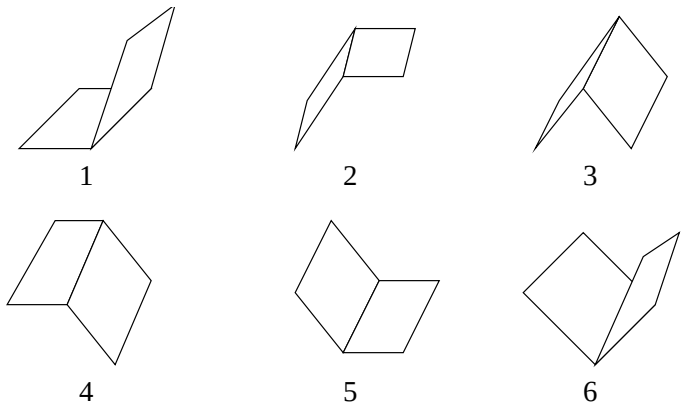


Figure 1.1: *Basic category-forms*

The direction of the inclinations and edges are limited to the points that can be seen in Figure 1.2. Each category is described by the following properties:

- **Inclination** The average inclination of the tile described by one direction pointing to the highest part of the tile.
- **Orientation** The edge that divides the tile into two planes is used to determine the orientation. It is described by a start-point and an end-point. The orientation can in certain cases represent an extreme point which can be any of the points except the corner-points. If the tile does not fit into one of the basic categories and could be described by only one plane, i.e. it has no edge, the orientation will have a value of zero.

- **State** This value determines if the orientation is positive or negative in relation to the inclination of the tile. For example, out of the basic forms in Figure 1.1, the categories 2, 3 and 4 are positive.

A flat category will have all values set to zero.
A more detailed description of the categories can be found in [1] and [2].

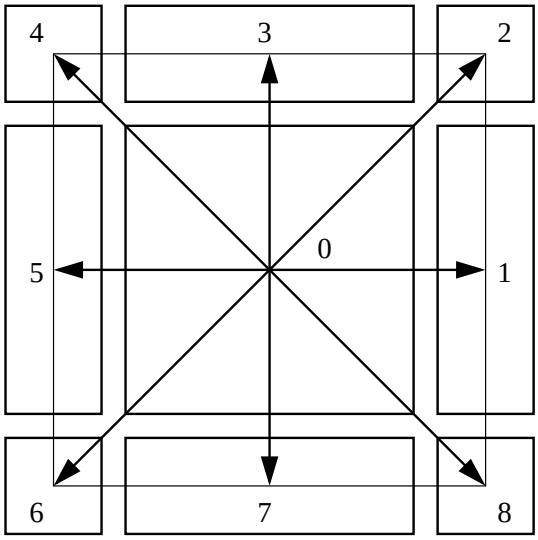


Figure 1.2: *The sub-parts of a tile, their integer encoding and the allowed inclination directions.*

Chapter 2 - Filter Algorithms

This chapter will describe the various algorithms that have been developed in this work. A filter is defined by a set of algorithms that are applied to a data-set to find only those structures that are matched by the specified parameters. The filter-process is divided into two main parts where the first is the matching of individual cross-section segments and the second is the connecting of these segments into objects.

2.1 Stage 1 - Finding segments

The basic idea is to use the categories to find the cross-section of the specific terrain object. This cross-section will be referred to as a segment.
For instance, if a ditch is to be found, a category with a downward-slope followed by zero or more flat categories must be found first (depending on how broad the ditch is), followed by an upward-slope. This will result in a segment of two or more categories as can be seen in the first example in Figure 2.1. A ditch-segment could also be limited to one tile, in which case the category would have no inclination and a negative edge as can be seen in the right-most example in Figure 2.1.

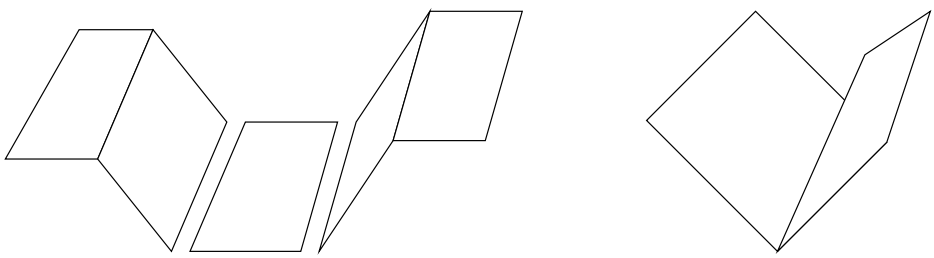


Figure 2.1: *Some possible ditch-segments*

If this is done for a large area, and a ditch exists, several ditch-segments will be found. However, several erroneous segments will also be found since this cross-section will not only be found in a ditch. For example, a small hole in the ground could possibly generate this type of segment.
How to deal with these incorrect segments is described in Section 2.2.

The filters are defined by specifying a “string” of tiles describing the feature that is to be found. Each tile in this string is made up of a set of allowed categories.
Several categories of the same type can appear consecutively, which will be specified with a minimum and maximum length of each character. Furthermore, a minimum

and maximum total length of the segment can be specified. See Chapter 3 on how this is implemented.

When the string has been specified, it can be searched for in the data-set using simple pattern-matching techniques. This search has to be carried out both horizontally and vertically to find all relevant patterns.

Often the filter-string will consist of three sub-segments: The first is the start-pattern, which in the ditch-example will be categories representing a downward-slope. The second is the middle-pattern which could, for example, be made up of a flat category with a minimum length of zero (no middle segment) and a maximum length of the width of the ditch. The third is the end-pattern, which in our example is the categories describing an upward-slope. Further discussions of different types of cross-sections can be found in Chapter 4.

The categories are not rotation invariant which means that if the search is to be carried out horizontally from west to east, the categories representing a downward-slope must have an inclination to the west.
The filter must be rotated 90 degrees when the search is to be carried out vertically. Then, if the search is done north to south, the downward-slope will be a category with an inclination to the north.
The notion that only a horizontal and a vertical search are sufficient is discussed in [1].

2.2 Stage 2 - Connecting segments to objects

When the first stage of the filtering has been completed, a lot of segments have been found. Some of these segments are possibly part of the terrain object that is being searched for, and some are definitely not.
To find out which segments are part of an object, adjacent segments must be connected into one single terrain-object. Several methods of how to do this will be described in this part.
All connect-algorithms use restrictions of the objects' width and height to determine which objects should remain and which should be removed. The width is calculated by taking the average width of all the segments in the object. The length is then calculated by simply dividing the total number of categories by the width. The result will be a reasonably good estimate of the total length of the object measured in number of

category tiles. If for example the filter will find an object with a total length of 5.5 and the tile-size is 2 meters the objects length measures in meters will be 5.5 x 2 = 11 meters.

2.2.1 Simple connect

The simplest method is to find all hits among the categories that touch each other forming a continuous object. A minimum number of categories in the object can be specified to filter out small, irrelevant objects.

This can be implemented with a recursive algorithm. In the following example the search is started at coordinates x and y. The recursion will continue until a complete object is found. The recursion will stop if no filtered tile is found at the specified co-ordinate so to be able to find all objects in the data-set this function has to be applied to every tile to find the starting-point of each object. The function will return the size of the object so that a decision can be made whether the object should be filtered or not.

```
simple(x,y) {
 size=0
 if is_filtered_stage_1(x,y) and          Check if the tile is part of a segment
    not is_filtered_stage2(x,y) then {    found in stage 1 and not already
                                          connected in this stage.
  size=1
  set_filtered_stage_2(x,y)               Mark the tile as connected.
  size = size + simple(x+1,y)             Continue searching to the east.
  size = size + simple(x-1,y)             Continue searching to the west.
  size = size + simple(x,y-1)             Continue searching to the north.
  size = size + simple(x,y+1)             Continue searching to the south.
 }
 return size;
}
```

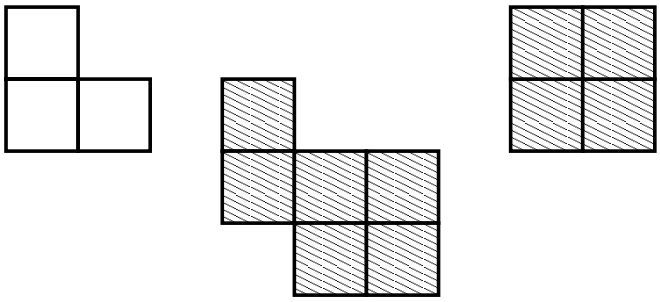


Figure 2.2: *Illustration of the simple connect algorithm*

Figure 2.2 shows an example of the simple connect algorithm where a minimum size of 4 is used. Two separate objects are found which are represented by the shaded tiles.

2.2.2 Connect segments

The second algorithm will connect adjacent segments. The search will first be performed from top to bottom and only connect the adjacent segments that were found in the horizontal search in stage 1. Only segments directly beneath the previous segment will be considered adjacent. Similarly, a search has to be made from left to right only connecting segments that were found in the vertical search in stage 1.
With this algorithm, error-tolerance can easily be implemented by instead of halting the search when a missing segments is found, continue and keep track of how many segments are missing. The search could then be halted when the number of missing segments exceed a certain tolerance-value.

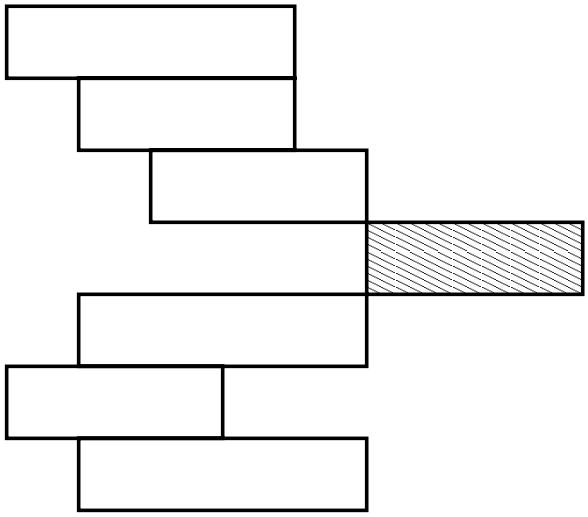


Figure 2.3: *Illustration of the vertical segment-search*

The algorithm for a vertical search of horizontal segments is illustrated in Figure 2.3. The white rectangles represent the detected object, and the shaded rectangle the rejected segment. In this example a certain amount of error is permitted resulting in line 4 being skipped and a new segment on line 5 being found directly below the segment on line 3. The rejected segment is not part of the object since it is not below any segment already found in the object. The result is one single object.

2.2.3 Connect edges and inclinations

The two algorithms described above are not concerned with the underlying category-structure since they only operate on information about the categories filtered out in stage 1. The following algorithm, however, will take into account the details of the categories that have been filtered.

This algorithm is similar to the simple algorithm described above in that it finds adjacent categories but with some added conditions of when to continue searching and when to stop.

The idea is to follow the edges (and inclinations) to find which segments should be part of the object. An inclination is seen as an edge where the direction is 90 degrees from the inclination. In the case of a tile with both an inclination and an edge the edge takes precedence. In each step of the search process a check is made to make sure that the current edge matches the previous edge. To expect an exact match of the edges is in most cases unrealistic so a tolerance of +/- 45 degrees will be allowed. If the check is successful, the direction this step will choose will be strictly based on where the edge is pointing (only one direction will be chosen unless there is an edge which does not divide the tile symmetrically, in which case there will be two directions depending on the angle of the edge).

The case where the tile contains an edge will be illustrated in Figure 2.4 by a category with an edge from point “5” to point “8” (with directions as specified in Figure 1.2).

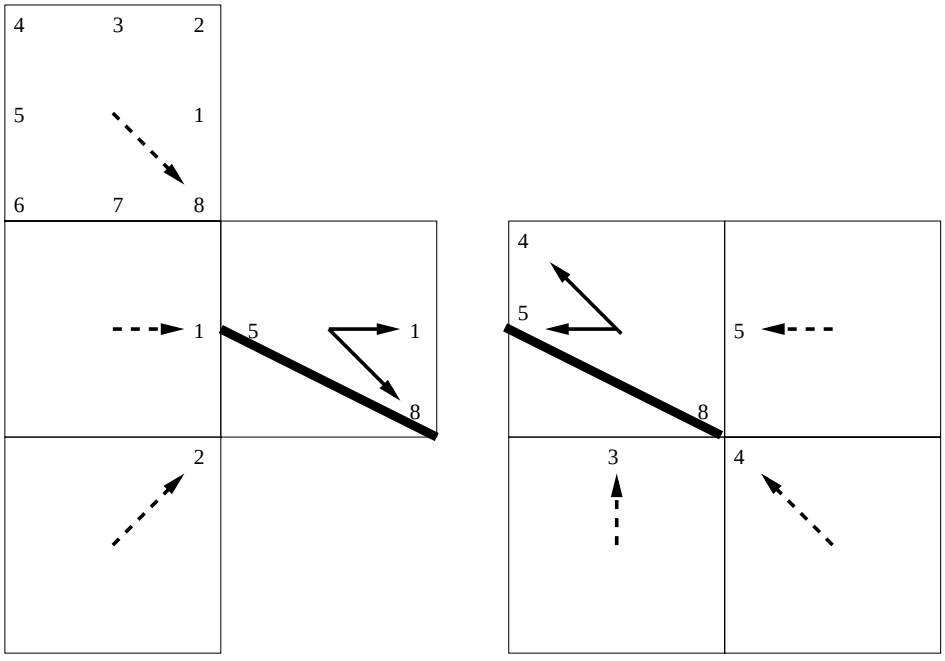


Figure 2.4: *Illustration of how the edge-connect algorithm searches edges.*

When the search reaches this category a check is made about whether the search should stop here or continue. To determine further action, the direction in which the search reached this category is compared with the points of the edge. In this example the first point is “5” so we check if the direction this category was reached is “8”, “1” or “2”. “1” is 180 degrees opposite from “5”. “2” and “8” are 45 degrees next to “1” (the tolerance discussed above). If that condition holds true then the searching continues at the category in the direction of point “8” (the other end of the edge). Furthermore, the point opposite of the starting-point (“5”) is “1” and should also be searched so that the search has been performed with regards to the whole edge and not only in the direction of the end-point. In the illustration in Figure 2.4, the dashed arrows point to the allowed direction of the previous search step, and the solid arrows to the directions in which searching should continue. The same procedure will be used when starting with the other point of the edge so that if the previous direction is “3”, “4” or “5”, searching should continue in directions “4” and “5”.
A special case will happen when there is an edge as in Figure 2.5 in which case the search will continue at the end-point of the edge as above and the point adjacent to it, furthest from the start-point.

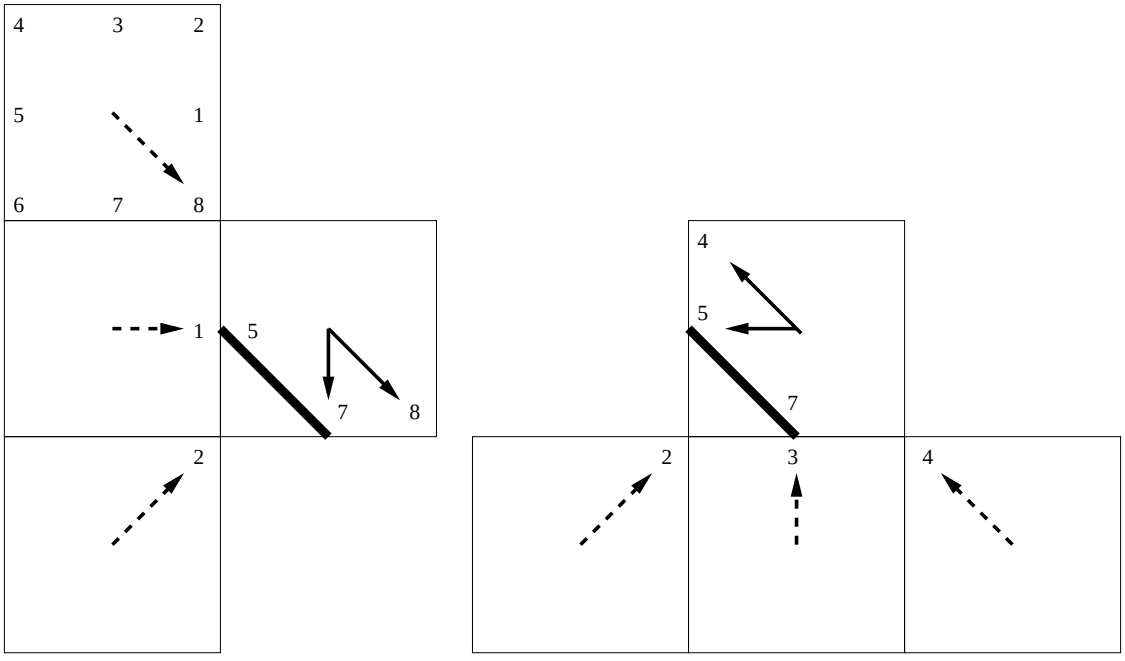


Figure 2.5: *Illustration of the special case of the edge-connect algorithm.*

Similarly, the same procedure is used for inclinations (only where no edge exists) where the directions to follow will be 90 degrees from the inclination. This is illustrated in Figure 2.6 using the same representation as above with the inclination rep-

resented by the thick arrows. There will be two cases here as well, one check from the “left” and one from the “right”.

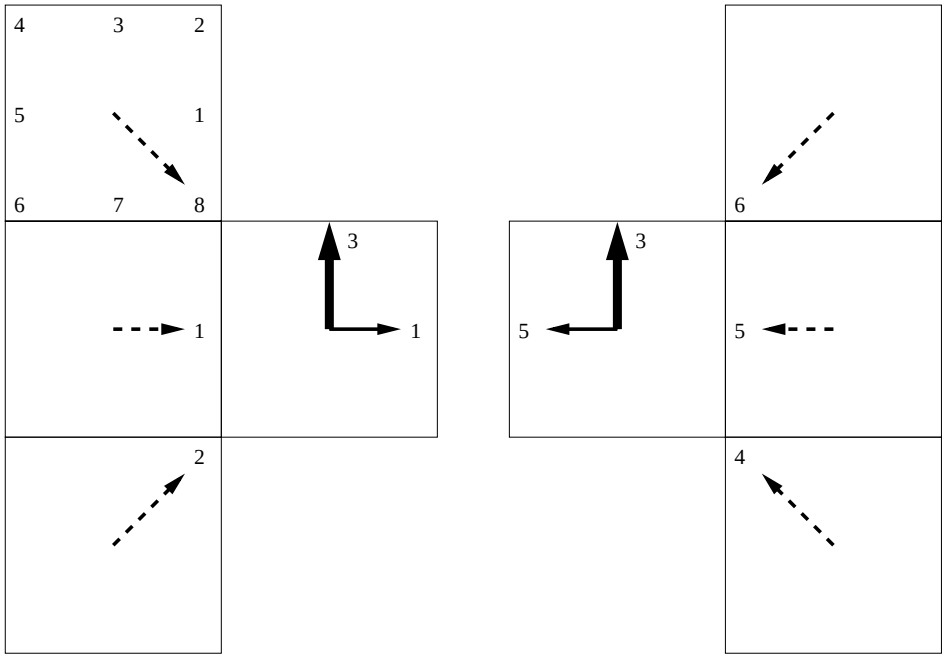


Figure 2.6: *Illustration of how the edge-connect algorithm searches inclinations*

Searching will also continue if the current category is part of the same segment as the previous one.
This algorithm is well suited for implementation as a recursive algorithm, which also is the case in the experiment system in Chapter 3.

2.3 More operations

Some additional filter operations will be described in this part. These filters can be applied before or after the filtering in stage 2.

2.3.1 Logical AND

This filter will combine horizontal and vertical matches with a logical AND-operation. Only those categories that are part of both a horizontal and a vertical segment will remain after filtering. This is illustrated in Figure 2.7 where there are both horizontal and vertical segments and the shaded tiles represents the result of this filter-operation.

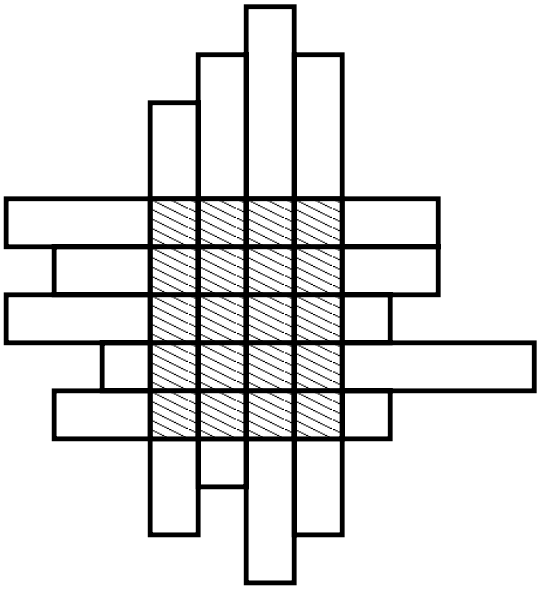


Figure 2.7: *Illustration of the Logical AND algorithm*

2.3.2 Rectangle

This filter will search the categories for rectangular objects with a specified width and height. An error tolerance is easily implemented so that a certain percentage of the rectangle must match.

Chapter 3 - The Experiment System

To evaluate and test the algorithms, a test-program has been written in Java [11]. This program is able to display categories and apply filters specified in text-files.

3.1 The program

The Java-program that has been developed is more of a framework for experimenting with different algorithms and filters than a finished end-product. The results of the filters are only shown in the visualization window and can not be exported outside of the program. Internally the different terrain-objects that are found are stored in a data-structure which means that it is possible, without too much work, to dump this data to an external file. The basic operation of the program can be seen as a flow-chart in Figure 3.1.

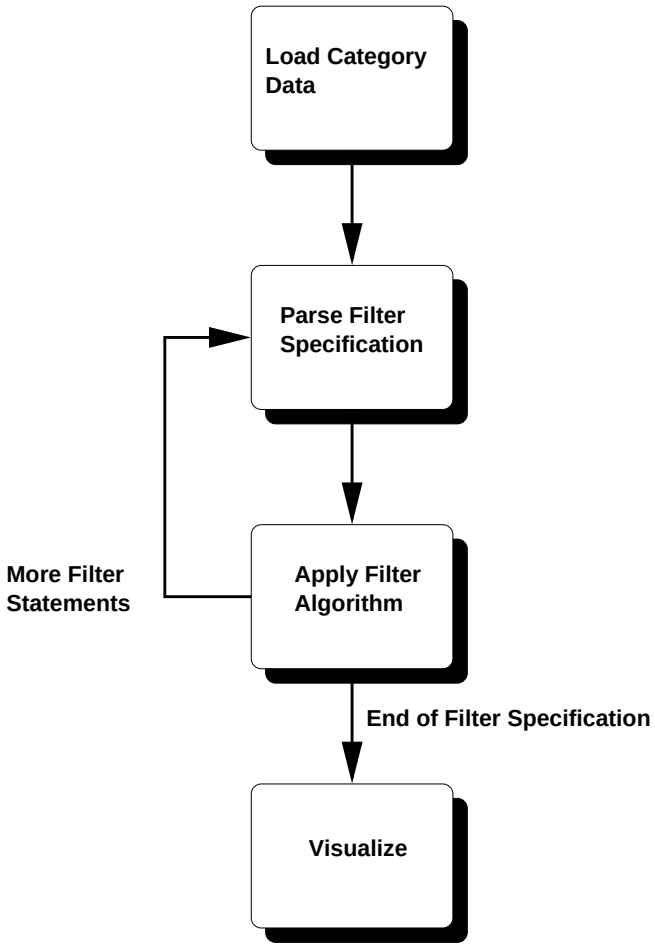


Figure 3.1: *Basic operation of the Java-program*

3.2 Category representation

A visual representation of the somewhat abstract categories has been devised. This representation is shown in Figure 3.2. Inclination is represented by a green line pointing upwards in the direction of the slope. The green line is replaced by a black arrow in this document. Orientation is represented by a black or white line representing the edge. A positive state is indicated by drawing the orientation in white and negative in black. The figure below also shows what a cross-section could possibly look like (this is not something that is part of the program but only shown here to better understand the categories).

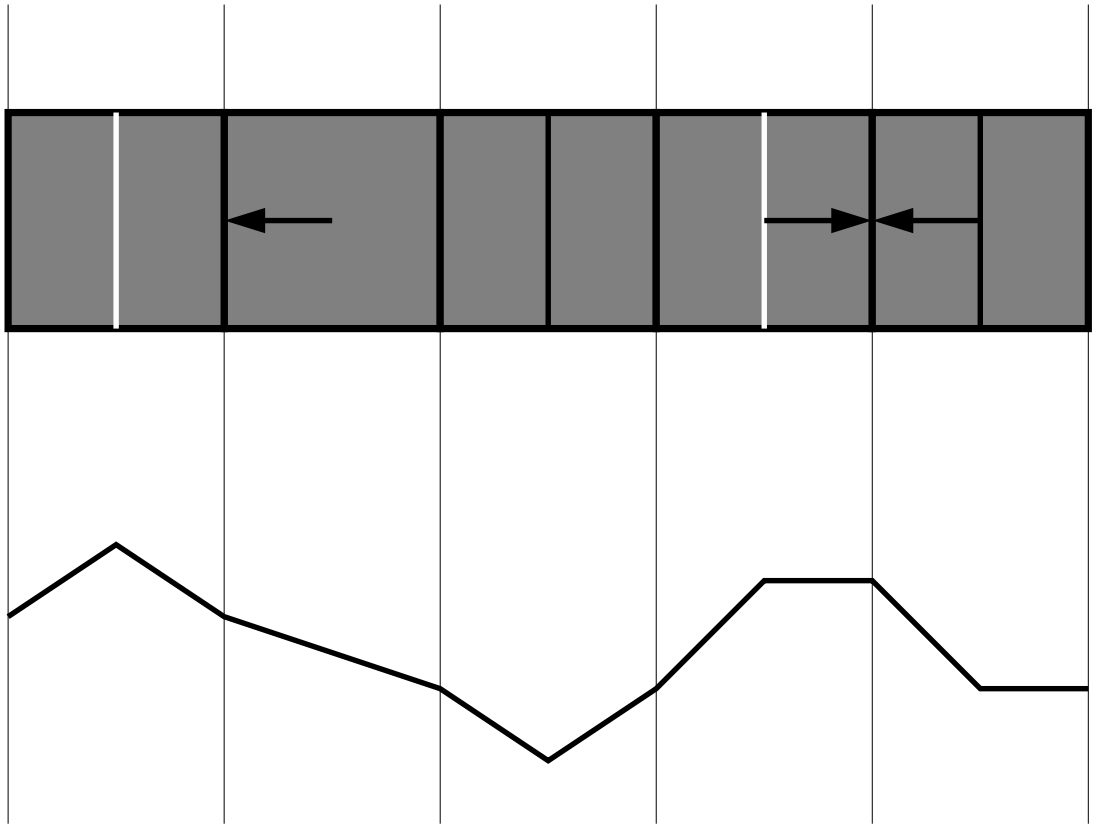


Figure 3.2: *The visual representation of 5 different categories with the cross-section underneath.*

3.3 Filter specifications

The filters are specified in a plain text-file that is parsed by the Java-program and applied to the data. The filters are applied in the order they are specified in the file.

Each segment will be searched for horizontally, after which all categories in the filter-specification are rotated 90 degrees and searched for vertically, which means that all filters have to be specified for a horizontal search, left to right.
All lengths and widths are specified in number of category-tiles, where each tile is 2 by 2 meters.
Lines starting with the string // will be considered comments.

3.3.1 Segment

A filter-segment as has been described in Section 2.1 is started with a *beginsegment* statement and is ended with *endsegment*. All text between these statements represents the "string" that is to be searched for. An optional parameter for the segment is *invert* which will invert the entire segment, i.e. positive state will become negative state etc. This can, for instance, be used to quickly transform a ditch-filter into a ridge-filter and vice-versa.
As all segments will be searched for both vertically and horizontally, this can be overridden by the statements *horizontal* and *vertical* to allow search in one specified direction.

3.3.2 Sub-segment

A segment can contain several sub-segments (or "characters") surrounded by *beginsubsegment* and *endsubsegment*. The *beginsubsegment* statement takes three parameters. The first two are required: minimum length and maximum length, which is the number of consecutive categories that should match this sub-segment. The third is the optional keyword *exclude* which means that this subsegment should not be part of the final segment although it still has to match. The subsegment can also contain the statement *maxerrors* that has one parameter that is the maximum number of not matched categories allowed. If this is not specified, no errors are allowed.

3.3.3 Category

The subsegment contains a set of allowed categories which are specified with the statements *begincategory* and *endcategory*. A category is specified with the statements *inclination*, *state* and *orientation* which can have several allowed values on one line. The keyword *not* is used to allow only the values *not* specified. The keyword *any* will allow all possible values.

3.3.4 Connect

After the segment-specifications, the post-processing algorithms can be specified. The statement *connect* will connect categories using the algorithm specified as the first parameter. Allowed algorithms are *edge*, *simple* and *segment* (see Section 2.2). Parameters two to five are minimum/maximum length and minimum/maximum width of the final object. These parameters are required, but by using a value of -1 the check can be disabled. If the segment-algorithm is used, a maximum error (specified in percent) can be set as an additional parameter.
The edge- and simple-algorithms are implemented as recursive algorithms which could lead to problems if the recursion depth is too great although this should not happen unless a filter is specified to find very large object in which case the resolution of the tiles should be altered.
Objects that are not in a straight line and that change direction (i.e. a winding stream) should be no problem for the edge-algorithm since it follows the direction of the edges and inclinations. The simple-algorithm ignores the directions as long as the categories are adjacent. The segment algorithm first searches from top to bottom followed by a search from left to right. This means that the algorithm only handles objects that are laid out in a straight line so objects that changes direction and are curved will not be found.

3.3.5 Additional statements

The statement *h-and-v* will use the logical-*AND* algorithm to combine horizontal and vertical segments. This statement takes no parameters.
The statement *findrectangle* will find rectangles of a specified size. The first parameter is the width, the second the height and the third the percentage of errors allowed. Note that this algorithm will not find the best but the first matching rectangle searching from left to right, top to bottom. More than one rectangle can be found but once a matching rectangle has been found, further searching will exclude that rectangular area.

3.3.6 Example

The following is a filter example that only contains one sub-segment which means that the complete segment will have a possible length of 1 to 3 categories (the minimum and maximum length of the sub-segment). The filter is only made as an example of how different statements are used and should not be seen as a filter that will

find any specific terrain-structure. More filter-specifications can be found in Appendix A.

```
beginsegment                          Start the segment
  invert                              Invert the segment, in this case the real
                                      inclinations searched for will be 2, 3 and 4
  beginsubsegment 1 3                 Start a subsegment with a minimum length
                                      of 1 and a maximum length of 3
    begincategory                     Start a category
        inclination 6 7 8             Allow inclinations 6,7 and 8
        state any                     Allow any state
        orientation not 14 15 16      Allow all orientations except 14, 15 and 16
    endcategory
  endsubsegment
endsegment
connect edge 10 -1 -1 -1              Connect categories using the edge-algorithm
                                      and a minimum length of 10
```

3.4 Test-data

The data used to test the filters have been taken from laser-radar flights over the FOI-area in September 2000 and over Kvarn, northwest of Linköping in August 2000. The available data were already sorted into files, each file covering an area of 100 by 100 meters, with the full data-set covering an area of 800 by 800 meters.
For each measured point there is an X, Y and Z coordinate that represents the global position. In addition to the coordinates, the reflectance value is stored in the raw data-files. The reflectance is the amplitude of the reflected laser pulse but this value is not used in this work, although a possible use for this is discussed in Section 5.1.3.
The data was first run through a ground-segmentation algorithm to find the ground only, i.e. removing all trees and houses. This was done using the method described in [4] with a grid-size of 0.5 m which means that for each 100 by 100 square meters there would be 200 by 200 data-point. The resulting data was then categorised as described in [2], with each category being 2 by 2 meters resulting in a data-set of 50 by 50 categories for each square. This means that the data reduction rate is 50x50/ 3x200x200 = 25/1200 = 1/48.

Chapter 4 - Results

This chapter will describe the different filters that have been developed. All filter-specifications can be found in Appendix A. The figures showing results of the filtering are screen-dumps from the Java-program. The zoomed-in images are 100 by 100 meters and contain the visualisation of all categories. The zoomed-out images are 800 by 800 meters and do not contain the details of the categories.

4.1 Ditch

This filter finds narrow ditches and is also very tolerant to changes in direction of the ditch. The specification contains three different segment-types:

- The first is a single-tile flat category with a negative edge (i.e. all orientations except extreme points).
- The second is a segment with a down-slope, flat and up-slope structure. This part finds segments that are part of a ditch that has a northeast-southwest orientation. Figure 4.1 shows all possible categories for each sub-segment and since the north-south (“37”) edge, as well as the inclinations “1” and “5” are part of the start and end sub-segment, segments with a north-south direction will also be found.
- The third segment is similar to the second. The only difference is in the orientation which is northwest-southeast.

These three segments-types are sufficient to find ditches orientated in all directions since the filters are also rotated 90 degrees.
The minimum width of a ditch-segment is one category-tile and the maximum width is three category-tiles. Figure 4.2 shows the result of the segment-filter where quite a few false-hit segments occur as well.
The segments are then connected using the connect-edge algorithm with a minimum allowed length of 10 (which equals 10x2 = 20 meters). This will remove most of the smaller objects, most likely not to be ditches. The results of this algorithm can be seen in Figure 4.3 where three ditches clearly can be seen. Two of these are in reality ditches beside a road, and the third is a small stream as can be seen in Figure 4.4. Unfortunately some possible ditches will be removed because of many missing segments. This is most likely because the ditches aren’t deep enough.

This filter can easily be inverted with the *invert* keyword. The result will be a narrow ridge as shown in Figure 4.5.

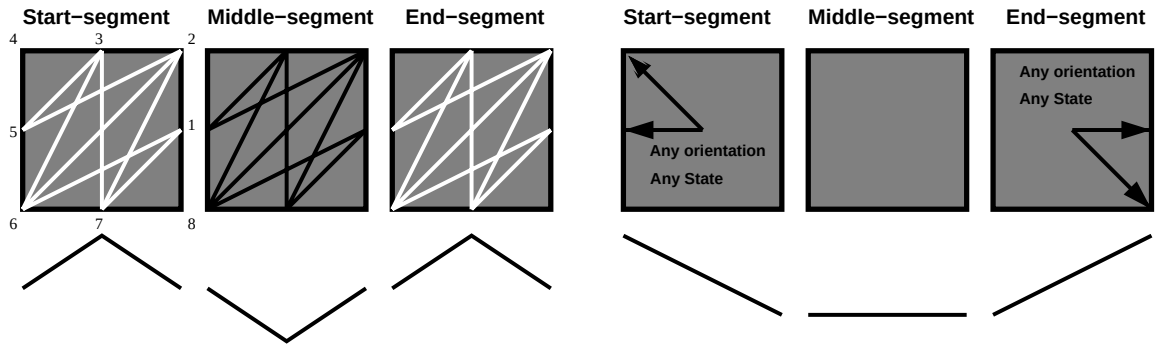


Figure 4.1: *Categories allowed in the ditch-segment that represents a north-east/ south-west orientation, with the corresponding cross-section beneath.*

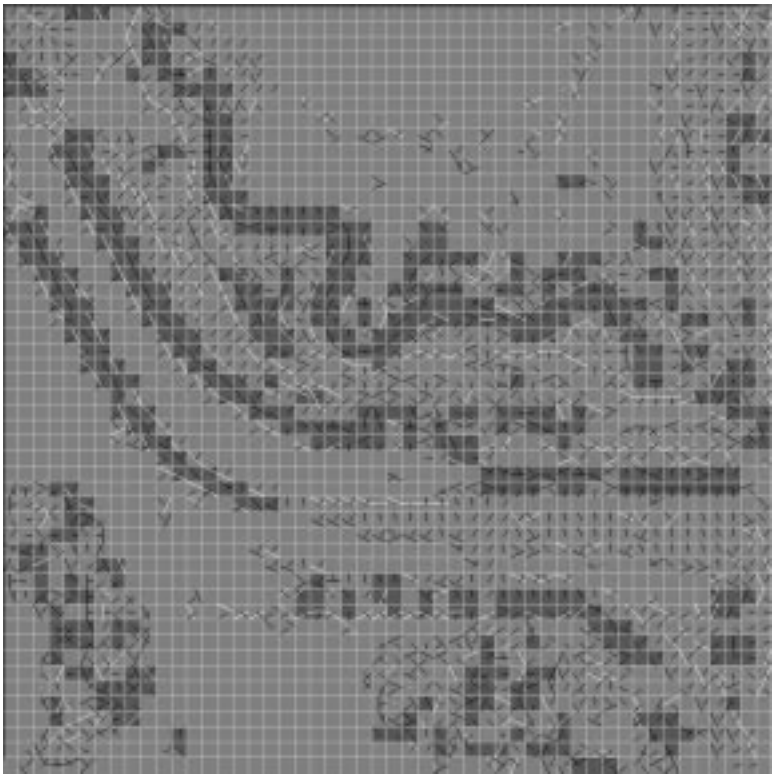


Figure 4.2: *The ditch-filter with all filtered segments including false hits.*

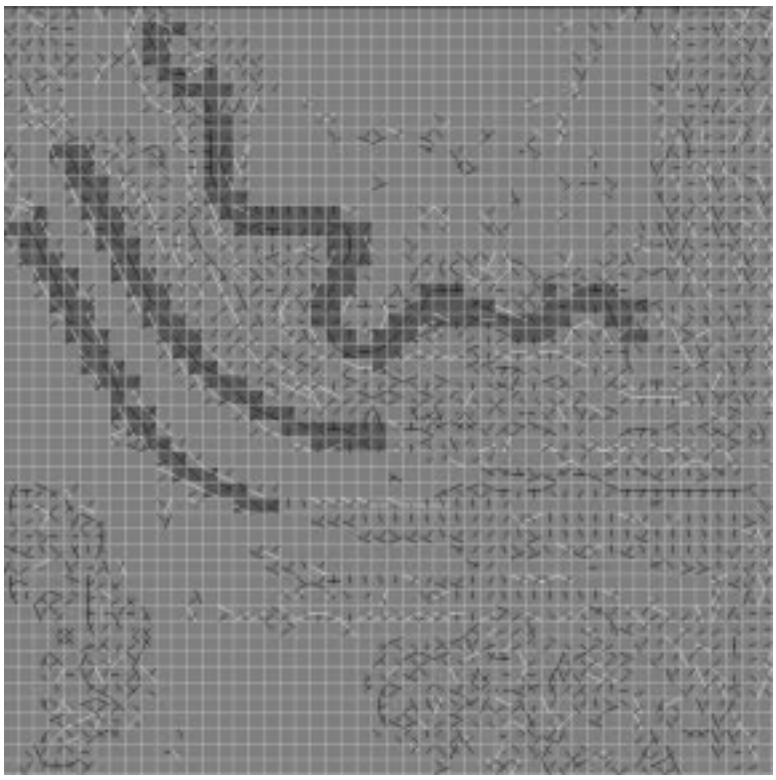


Figure 4.3: *The ditch filter with segments connected using the edge-connect algorithm.*

Figure 4.4: *The small stream in real life.*

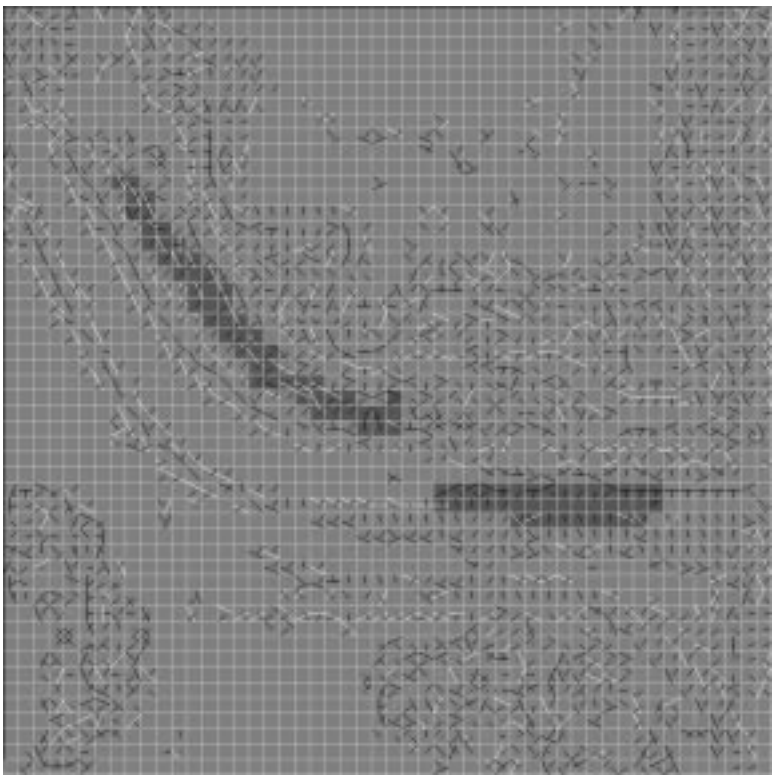


Figure 4.5: *Narrow ridges found by the inverted ditch-filter.*

4.2 Ridge

This filter is similar to the ditch filter, but the scale is changed to find larger structures. Compared to the ditch-filter, it is somewhat simplified since the more precise definition of categories is not needed for an object of this size. Only one segment is specified and the allowed categories are illustrated in Figure 4.6. The start-segment is an up-slope with two to four category-tiles. The middle-segment consists of one to four category-tiles of any kind, and the end-segment is a down-slope with two to four category-tiles. The total segment length is limited to five to twelve category-tiles. Finally, the connect-edge algorithm is applied with a minimum length of 50 tiles and a minimum width of 10 tiles. The final results can be seen in Figure 4.7.

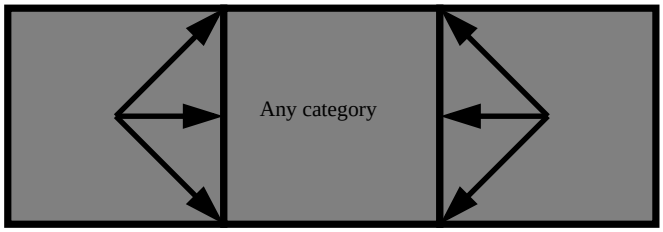


Figure 4.6: *Categories in the ridge-filter.*

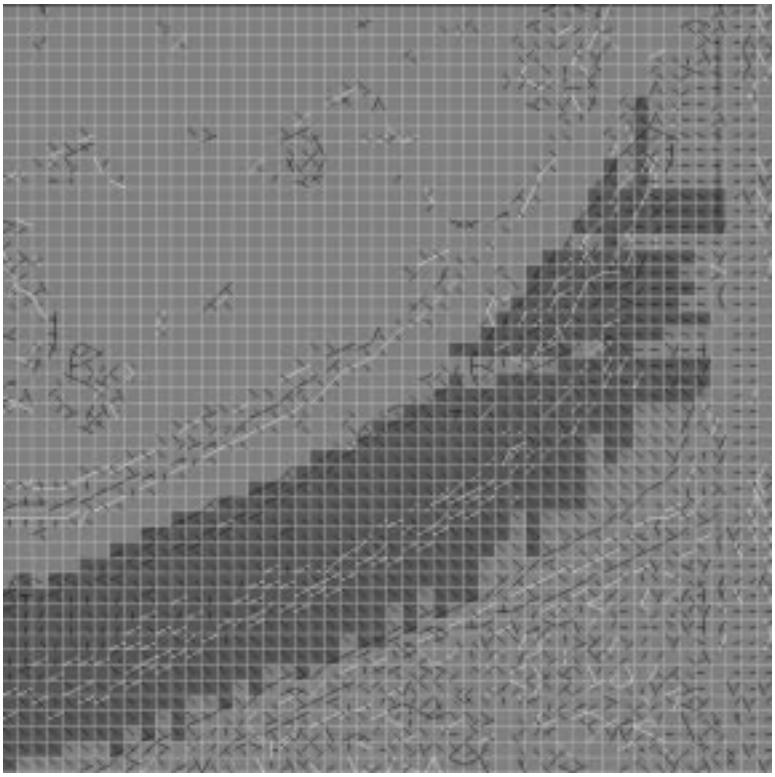


Figure 4.7: *The result of the ridge-filter.*

4.3 Hill

This filter finds small round hills and uses a different approach in the description of the filters.

Instead of using the cross-segment of a hill, this filter will find the round edges around it. This is possible by looking for a more specific category-string. An ideal hill will have categories like those in Figure 4.8. By searching for the segment that has the categories with inclinations “8”, “7” followed by “6”, the north edge of the hill will be found. By doing similarly for all edges, an outline of the hill will appear. Post-processing for this filter is done by applying the simple-connect algorithm. This could be more refined by making sure that all four different edges are part of the object, but this has not been done in this work. Figure 4.9 will show a hill being detected in the real data-set.
Since the segment-search is only done horizontally and vertically this technique will only work for this kind of round structure. If, for example, an oval hill is to be detected with this method, only objects that are totally aligned horizontally and vertically would be found. To be able to use this method with any type of object, the segment-search has to be done not only horizontally and vertically but in all directions.

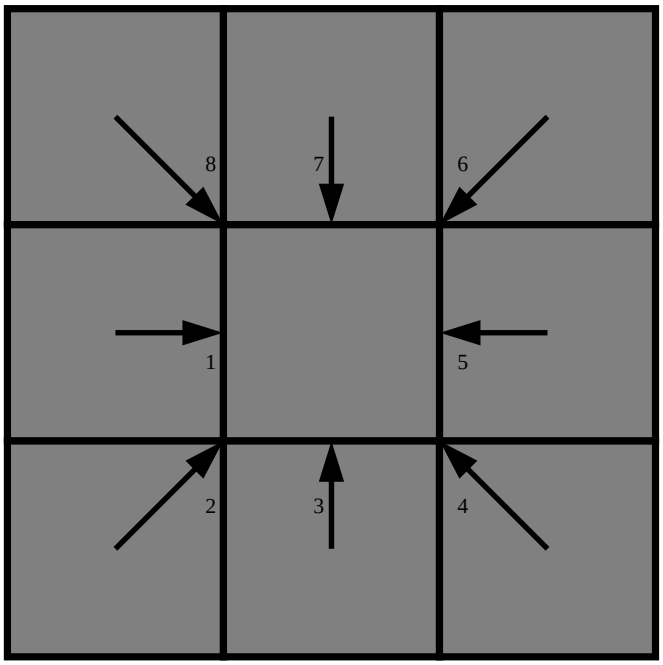


Figure 4.8: *An ideal categorisation of a hill.*

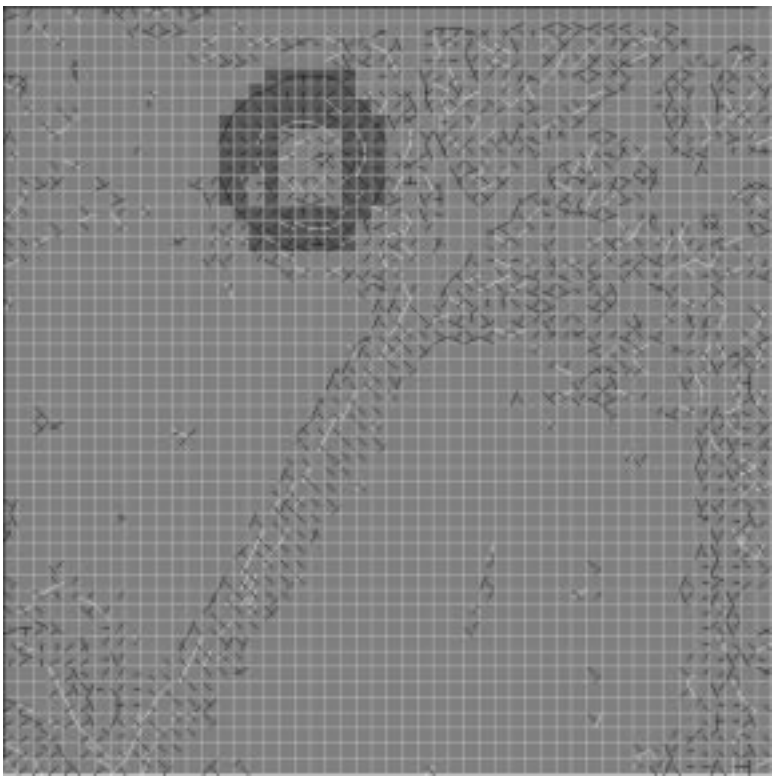


Figure 4.9: *The results of the hill-filter.*

Figure 4.10: *The hill in real life.*

4.4 Pond

This filter will be able to find a little pond with a somewhat round shape. This is done by looking at the cross-section, which ideally will look something like Figure 4.12. Since a long flat segment is bound to have errors in it, a certain error tolerance is allowed. To ensure that the object is “closed” the cross-section has to be found both horizontally and vertically by using the post-processing filter *h-and-v.* The edge-connection algorithm is then applied to allow a minimum length and width of 15 categories. The final result can be seen in Figure 4.11 and a photograph of the actual pond in Figure 4.13.

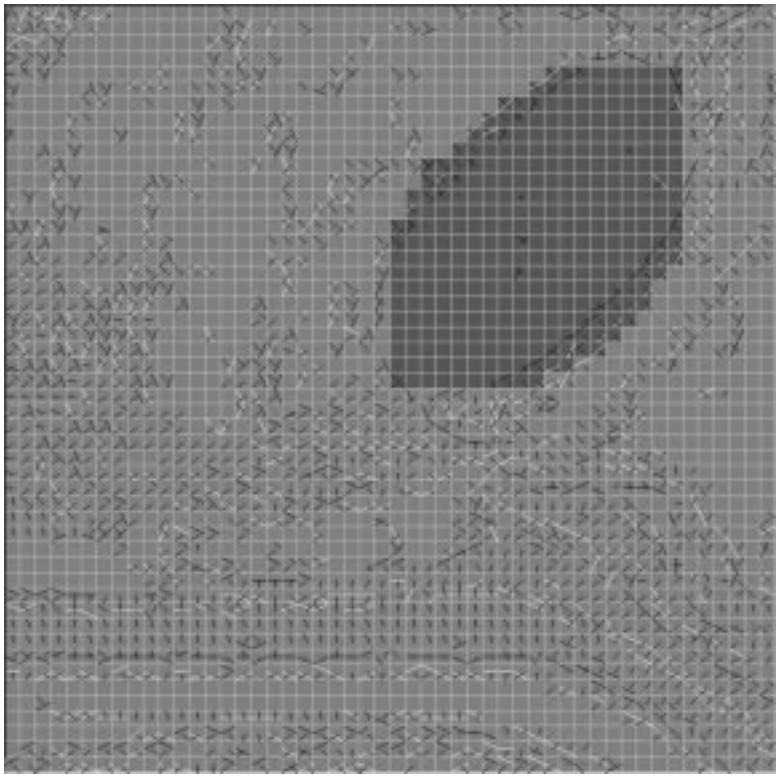


Figure 4.11: *The result of the pond-filter.*

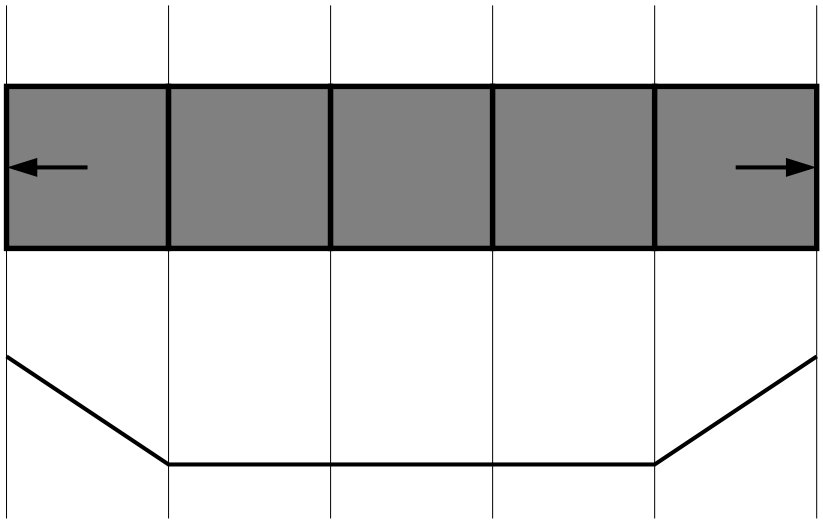


Figure 4.12: *The pond segment and cross-section.*

Figure 4.13: *The pond in real life.*

4.5 Flat

This is a very simple filter that will filter out all flat categories, i.e. categories with no inclination, no orientation and no state. The results can be seen in Figure 4.14 where a lot of interesting terrain formations are present, most notably the roads and the large field to the east.

Figure 4.14: *The flat categories (shown in grey) in the data-set in an area of 800 by 800 meters. The black rectangle is 100 by 100 meters and is the result of the flat-area filter.*

4.6 Flat-area

This filter will apply the *findrectangles*-algorithm to the flat-filter. The size of the area is 50 by 50 categories which equals 100 by 100 meters. An error tolerance of 5 percent is allowed. Results can be seen in Figure 4.14. Note that since the ground-segmentation algorithm has been applied to the data the ground could be covered by trees, so for this filter to applicable in any practical drivability application, no ground-segmentation should be done or the tiles must be classified with respect to the classes “open field” and “forest”.

4.7 Road

This filter is capable of finding roads. The principle is to find the segments that start and end with a category that is not flat, and with several flat categories in between. The enclosing categories should not be part of the segment. This is done by using the *exclude* parameter to the *beginsubsegment* statement of the non-flat categories.

What will remain are segments with only flat categories. These segments are connected by using the connect-segment algorithm. A minimum length of 10 is used with an error percentage of 20. The filter applied to the complete data-set is shown in Figure 4.15. Several shorter road-parts can be seen in several places and are most likely flat areas where houses were removed by the ground-segmentation. Since the connect-segments algorithm does not effectively handle changes in direction, some parts of the road are missing, see also Section 5.1.2 for further discussions on the road-filter.

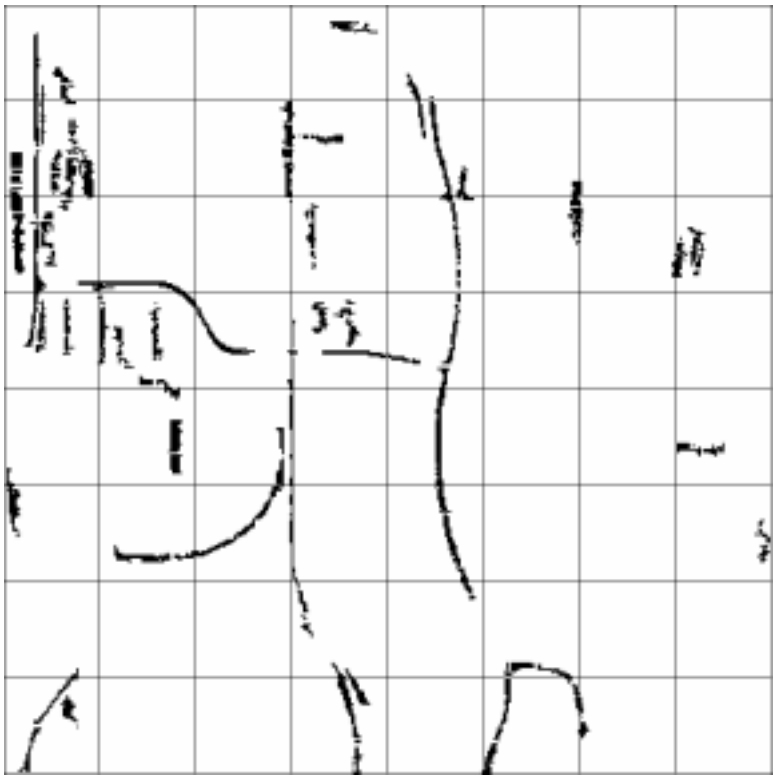


Figure 4.15: *The results of the road-filter.*

Chapter 5 - Discussion

This chapter will discuss some of the problems, the algorithms, the results and possible future improvements of the work.

5.1 Categories

5.1.1 Limitations

Since the categories are approximated by a maximum of two planes this could result in the disappearance of certain features. For instance, a ditch located on the slope of a hill could be difficult to detect. If the slope is more dominant than the ditch, the categories on the slope could take precedence over the categories representing the ditch.

5.1.2 Height

An additional parameter that has been subject to experiments is the height of the categories. This parameter corresponds to the maximum distance between the highest and the lowest point of the category. This data is not a formal part of the category-definition but is nonetheless present and has been experimented with. The most interesting use of this parameter is to consider categories below a minimum height as flat categories. As the category algorithm is designed now, the minimum height of a non-flat category is 0.2 meters. Because of the resolution of the original data, the height in the categories are rounded off into one decimal. In all the filters described above, no filtering has been made in regards to the category-height, but this should not be necessary since the filters are applied to a data-set that in most parts describe an area which is very flat. Making it more flat would be to loose a lot of detail. When the filters were applied to another data-set, the Kvarn-area, which in most parts can be classified as rough forest, the road-filter in particular did not perform very well. To solve this problem a pre-filter was applied that converts all categories that have a height less than 0.3 meters into a complete flat-category. Even the bumpy forest-roads were detected using this technique as can be seen in Figure 5.1

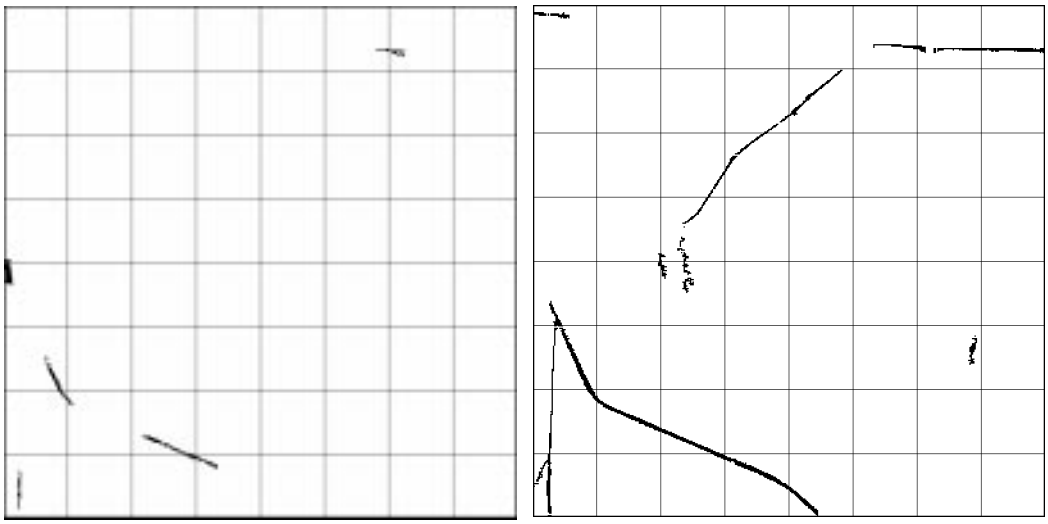


Figure 5.1: *Left: The standard road-filter applied to the Kvarn-area. Right: The categories with a height of less than 0.3 meters were converted to flat categories before the filter was applied.*

5.1.3 Improvements

Here, some possible enhancements of how the category-data is applied that can possibly help to improve the filtering technique, are presented. These improvements needs to be thoroughly evaluated since this has not been done in this work.

- **Angle** The filters could be refined further by calculating the angle of the inclinations.
- **Intensity** The road algorithm for example, could be greatly improved by using the intensity information from the laser-radar data.

5.1.4 Used categories

Most of the information coded into the categories were used in the construction of the filters. The most useful part of the categories is the inclination which is used by almost all the filters. The orientation (edge) and state is mostly useful when looking for smaller objects like the narrow ditches. A use for the extreme point (i.e. where the orientation is 11, 22, etc.) is yet to be found.

5.1.5 Multi-resolution

The effects of reducing the resolution of the categories should be studied further. In this work only categories with a size of 2 by 2 meters have been used.

5.2 Algorithms

5.2.1 Edge-connect

Some ideas were rejected during the development of the algorithms. Especially the edge-connect algorithm was rewritten several times. The initial idea of the edge-connect algorithm was, for each segment, to calculate the direction of the entire segment, i.e. north, north-east, etc. The algorithm would then search in the direction of the segment. This worked quite well but was very inefficient and too complex. The current method, which only takes into account the direction of each category, is less complex, more efficient and thus yields better results.

5.2.2 Segment-search

The search-algorithm that in the first step of the filtering process finds the segments was initially implemented in the simplest possible way:

- Search for the first category in the segment-string.
- Continue matching the rest of the string.
- Backtrack to after the first category if a mismatch is found.

A significant improvement was gained by using some ideas from [9] and [10]. The improved algorithm works as follows:

- Search for the last category in the segment-string.
- Continue matching the rest of the string backwards.
- If a mismatch is found continue after the last category.

When this was implemented, a significant reduction in computation time was achieved.

5.2.3 Fuzzy-matching

Fuzzy-matching by using theories from [6] and [7] were considered but eventually rejected since the algorithms were not directly applicable to this work.

5.2.4 Efficiency

A simple evaluation of the algorithms efficiency can be made by measuring the time spent by the Java-program when applying the filters. The timings were done on a Sun UltraSPARC-II 500 MHz and the filters were applied to the whole FOI-area of 800x800 meters. The results can be seen in Table 5.1.

Filter	Time (sec)
Ditch	10
Hill	4
Pond	4
Flat	3
Flat-Area	58
Road	4

Table 5.1: *The overall execution time (excluding loading of data) for some different filters.*

As could be expected, the more complex filters (e.g. the ditch) take more time than the simpler ones (e.g. the flat-filter). The *findsquare*-algorithm used by the Flat-Area filter is very inefficient which results in a very long execution time.
A more detailed study shows that most of the execution time is spent in the first part, i.e. the segment-matching algorithm. If any further optimizations are to be made, this is where to concentrate the work.

5.3 Objects

5.3.1 Problems

With the methods used, some erroneous objects may theoretically be found because no regards as to the context of the object is accounted for. For example, if there are two ridges running in parallel, the area between them could possibly be interpreted as a ditch. This could be solved by assigning a priority value to each filter and then

applying them in order. When an object is detected, remove it from the data-set and continue with the next filter that has a lower priority. This could mean that if the ridges are detected prior to the ditches the problem in the above example could disappear.

5.3.2 Object Properties

The filters will find the width and the length of the objects. The width is however not very exact. In most cases, the width will be larger than in reality since surrounding categories (for example, the down-slope and up-slope in the ditch) are not fully part of the object. A ditch, for example, could be detected as having a width of 2 (one down-slope and one up-slope) which means the ditch would be 2x2=4 meters wide, when the ditch in reality is only one meter wide. To overcome this problem a more exact mathematical method may be applied directly to the found ditches, e.g. in cases where the purpose is to determine whether the ditch is possible to cross by a certain type of vehicle.

5.3.3 Combination of objects

It should be possible to combine different types of filters to find other types of objects. A pass, for example, could be detected by finding a gap between two ridges or hills.

5.3.4 Refine objects

Even though the algorithms are designed to handle certain errors or uncertainties, the complete object is not always found, either because of the limitations of the categories or simply because of errors in the original data. This could be solved by looking at adjacent objects and determining if they in fact are part of the same terrain formation. This has not yet been done, and in case this work should continue this problem should be given a high priority.

Chapter 6 - Conclusion

This work has demonstrated that it is possible to find different kinds of objects by applying filters made up by the categories. Some refinement of the methods described in [1] has been made and a quite advanced framework for experimenting with different filters has been designed and implemented.

Future work must include improvement of the filters and further types of terrain-objects should be possible to identify. Some further ideas for improvement have been suggested in Chapter 5, with the most important part being the refinement of objects in Section 5.3.4.
By using the formal filter-specification which has been described in Section 3.3 it is possible to more adequately and correctly describe the terrain-objects which will be the foundation for further work on the development of a query-language.
By using this formalization for filter description it has been possible to demonstrate the use of the filters for determination of e.g. ditches, ridges, ponds, roads etc. We believe that it is possible to develop further filter types for other types of terrain features.
Another application that will be subject to further research will be to refine the technique in order to also determine terrain driveability which is an important problem that require more efficient techniques.

References

[1] Johan Fransson, Fredrik Lantz and Erland Jungert, “A Terrain Data Model With a Symbolic Structure For Identification of User Defined Terrain Objects”, Submitted to the 2002 Spring American Geophysical Union Meeting (Laser Altimetry Session),Washington DC, May 28-31, 2002.

[2] Fredrik Lantz and Erland Jungert, “Dual aspects of a Multi-Resolution Grid-Based Terrain Data Model with Supplementary Irregular Data Point”, Proceedings of the 3rd Int. Conf. on Information fusion, Paris, July 0-13, 2000.

[3] M. Elmqvist, E. Jungert, F. Lantz, Å. Persson, U. Söderman, “Terrain Modelling and Analysis Using Laser Scanner Data”, Proceedings of ISPRS Workshop on Land Surface Mapping and Characterization Using Laser Altimetry, Annapolis, MD, Oct 22-24, 2001, pp 219-226.

[4] Magnus Elmqvist, “Ground Estimation of Laser Radar Data using Active Shape Models”, Presented at the OEEPE workshop on Airborne Laserscanning and Interferometric SAR for Detailed Digital Elevation Models, Stockholm, Sweden, March 1-3, 2001.

[5] Simon Ahlberg, Christina Carlsson and Pontus Hörling, “Måligenkänning och terrängmodellering med laserradardata”, FOA R-99-01303-408-SE, 1999.

[6] Martien Molenaar and Tao Cheng, “Fuzzy spatial objects and their dynamics”, ISPRS Journal of Photogrammetry and Remote Sensing, 55 164-175, 2000.

[7] Didier Dubois, Henri Prade and Claudette Testemale, “Weighted Fuzzy Pattern Matching”, Fuzzy Sets and Systems vol.28, no.3 p.313-31, 1988.

[8] Robert R Hoffman, “What is a Hill? An analysis of the meaning of generic topographic terms”, Army Engineer Topographic Labs., 1985.

[9] Thomas Wang, “A Fast String Scanning Algorithm with Small Startup Overhead”, http://www.concentric.net/~Ttwang/tech/stringscan.htm, 1999.

[10] Christian Charras and Thierry Lecroq, “Exact String Matching Algorithms”, http://www-igm.univ-mlv.fr/~lecroq/string/index.html, 1997.

[11] David Flanagan, “Java in a Nutshell”, ISBN 1-56592-487-8, 1999.

Appendix A - Filter Specifications

A.1 Ditch

```
// Single tile, flat negative edge
beginsegment
//invert
  horizontal
  beginsubsegment 1 1
    begincategory
        inclination 0
        state 2
        orientation not 00 11 22 33 44 55 66 77 88
    endcategory
  endsubsegment
endsegment

//SOUTH-EAST
beginsegment
//invert
  beginsubsegment 1 1
    begincategory
      inclination 0
      state 1
      orientation 26 27 35 36 37 17 25 16
    endcategory
    begincategory
      inclination 4 5
      state any
      orientation any
    endcategory
  endsubsegment

  beginsubsegment 0 1
    begincategory
      inclination 0
      state 0 2
      orientation 0 26 27 35 36 37 17 25 16
    endcategory
  endsubsegment

  beginsubsegment 1 1
    begincategory
      inclination 0
```

```
      state 1
      orientation 26 27 35 36 37 17 25 16
    endcategory
    begincategory
      inclination 1 8
      state any
      orientation any
    endcategory
  endsubsegment
endsegment

//SOUTH-WEST
beginsegment
//invert
  beginsubsegment 1 1
    begincategory
      inclination 0
      state 1
      orientation 13 37 38 47 48 57 58 14
    endcategory
    begincategory
      inclination 5 6
      state any
      orientation any
    endcategory
  endsubsegment

  beginsubsegment 0 1
    begincategory
      inclination 0
      state 0 2
      orientation 0 13 37 38 47 48 57 58 14
    endcategory
  endsubsegment

  beginsubsegment 1 1
    begincategory
      inclination 0
      state 1
      orientation 13 37 38 47 48 57 58 14
    endcategory
    begincategory
      inclination 1 2
      state any
      orientation any
    endcategory
```

```
  endsubsegment
endsegment

connect edge 10 -1 -1 -1
```

A.2 Ridge

```
beginsegment 5 12
  beginsubsegment 2 4
    begincategory
      inclination 1 2 8
      state any
      orientation any
    endcategory
  endsubsegment

  beginsubsegment 1 4
    begincategory
      inclination any
      state any
      orientation any
    endcategory
  endsubsegment

  beginsubsegment 2 4
    begincategory
      inclination 4 5 6
      state any
      orientation any
    endcategory
  endsubsegment
endsegment
connect edge 50 -1 10 -1
```

A.3 Hill

```
beginsegment 5 12
  beginsubsegment 1 4
    begincategory
      inclination 2
      state any
      orientation any
    endcategory
```

```
  endsubsegment

  beginsubsegment 1 4
    begincategory
      inclination 3
      state any
      orientation any
    endcategory
  endsubsegment

  beginsubsegment 1 4
    begincategory
      inclination 4
      state any
      orientation any
    endcategory
  endsubsegment
endsegment

beginsegment 5 12
  beginsubsegment 1 4
    begincategory
      inclination 8
      state any
      orientation any
    endcategory
  endsubsegment

  beginsubsegment 1 4
    begincategory
      inclination 7
      state any
      orientation any
    endcategory
  endsubsegment

  beginsubsegment 1 4
    begincategory
      inclination 6
      state any
      orientation any
    endcategory
  endsubsegment
endsegment

connect simple 5 -1 -1 -1
```

A.4 Pond

```
beginsegment
  beginsubsegment 1 4
    begincategory
      inclination 4 5 6
      state any
      orientation any
    endcategory
  endsubsegment

  beginsubsegment 8 20
    maxerrors 2
    begincategory
      inclination 0
      state any
      orientation any
    endcategory
  endsubsegment

  beginsubsegment 1 4
    begincategory
      inclination 1 2 8
      state any
      orientation any
    endcategory
  endsubsegment
endsegment

h-and-v
connect edge 15 -1 15 -1
```

A.5 Flat

```
beginsegment
  beginsubsegment 1 1
    begincategory
        inclination 0
        state 0
        orientation 0
    endcategory
  endsubsegment
endsegment
```

A.6 Flat-area

```
beginsegment
  beginsubsegment 1 1
    begincategory
        inclination 0
        state 0
        orientation 0
    endcategory
  endsubsegment
endsegment
findrectangle 50 50 6
```

A.7 Road

```
beginsegment
  beginsubsegment 1 1 exclude
    begincategory not
      inclination 0
      state 0
      orientation 0
    endcategory
  endsubsegment

  beginsubsegment 1 6
    begincategory
      inclination 0
      state 0
      orientation 0
    endcategory
  endsubsegment

  beginsubsegment 1 1 exclude
    begincategory not
      inclination 0
      state 0
      orientation 0
    endcategory
  endsubsegment
endsegment
connect segment 10 -1 -1 -1 20
```