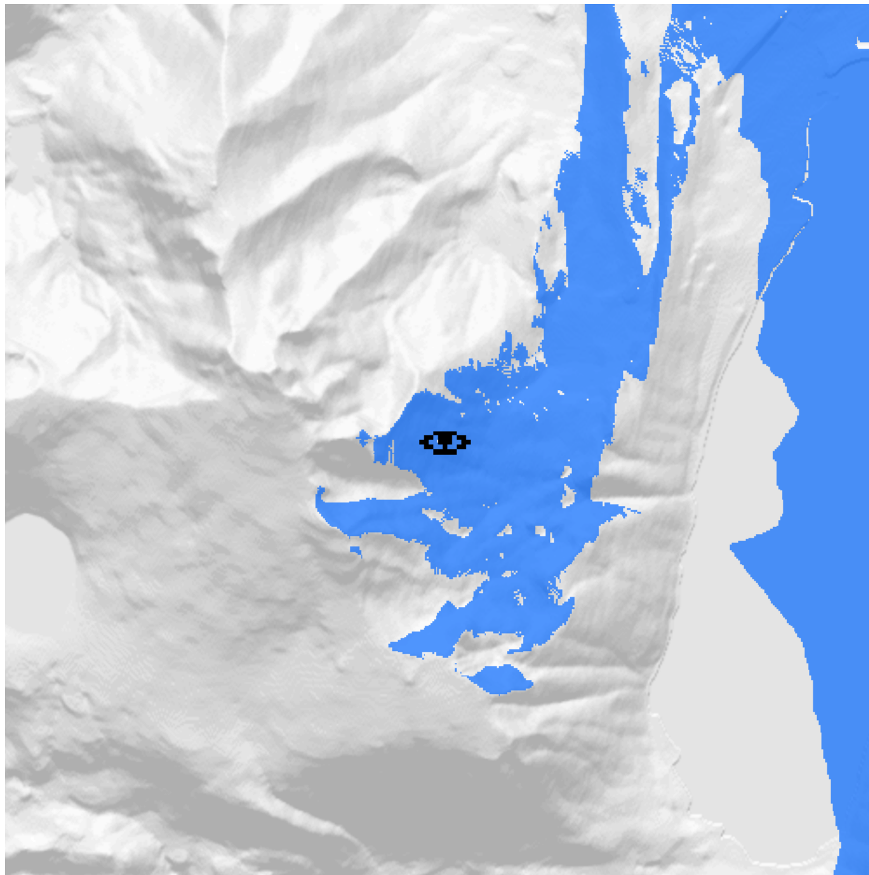


Distributed Viewshed Analysis

An Evaluation of Distribution Frameworks for Geospatial Information Systems

EMIL JOHANSSON
JACOB LUNDBERG

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY AND UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2016

Master's thesis 2016

Distributed Viewshed Analysis

An Evaluation of Distribution Frameworks
for Geospatial Information Systems

EMIL JOHANSSON
JACOB LUNDBERG

CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF
GOTHENBURG

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden 2016

Distributed Viewshed Analysis
An Evaluation of Distribution Frameworks
for Geospatial Information Systems
EMIL JOHANSSON
JACOB LUNDBERG

© 2016 EMIL JOHANSSON, JACOB LUNDBERG.

Academic supervisors:	Birgit Grohe
	Department of Computer Science and Engineering
	K.V.S Prasad
	Department of Computer Science and Engineering
Industry supervisor:	Calle Hanson
	Carmenta AB
Examiner:	Graham Kemp
	Department of Computer Science and Engineering

Master's Thesis 2016 Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A viewshed analysis over an area at Lake Tahoe, CA, USA. The eye denotes the position of the observer and the areas visible are marked in blue.

Gothenburg, Sweden 2016

Distributed Viewshed Analysis
An Evaluation of Distribution Frameworks
for Geospatial Information Systems
EMIL JOHANSSON
JACOB LUNDBERG
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Viewshed analysis is the process of computing what areas of a terrain are visible from a certain observation point. In this thesis we evaluated the performance of these computations on cloud clusters using the distribution framework Apache Spark. We implemented three commonly used viewshed algorithms; R3 which is slow but highly accurate as well as R2 and van Kreveld which are faster but less accurate. Two versions of each algorithm were implemented, one to run on a single multi-core machine and one to run on a server cluster using Spark. We compared the accuracy and running time of the different algorithms in order to determine when to use the different algorithms. Our results show that viewshed analysis does not perform well when implemented using Spark if real-time results are required. In fact the faster algorithms performed consistently worse on the cluster, even for very large input data. For the highly accurate, but slow, R3 algorithm we were able to achieve a 1.6x speedup using the distribution framework.

Keywords: viewshed, GIS, distributed, cluster, line-of-sight, Apache Spark

Acknowledgements

We would like to thank our supervisors from Chalmers, Birgit Grohe and K.V.S Prasad for their feedback and fruitful meetings. We would also like to thank Carmenta and the people working there for making us feel welcome and for supporting us trough our project, with a special thanks to our industry supervisor Calle Hanson.

Emil Johansson, Jacob Lundberg, Gothenburg, June 2016

Contents

List of Figures **x**

List of Tables **xiii**

1 Introduction **1**
- 1.1 Background . . . 1
- 1.2 Problem Description . . . 1
- 1.3 Previous Work . . . 2
- 1.4 Goals . . . 3
- 1.5 Limitations . . . 3
- 1.6 Overview of Thesis . . . 4

2 Theory **5**
- 2.1 Digital Elevation Model . . . 5
- 2.2 Line of Sight . . . 6
- 2.3 Viewshed Analysis . . . 8
- 2.4 Viewshed Algorithms . . . 9
  - 2.4.1 R3 . . . 9
  - 2.4.2 R2 . . . 11
  - 2.4.3 Van Kreveld . . . 12
- 2.5 Framework: Apache Spark . . . 15

3 Implementation **18**
- 3.1 R3 . . . 18
- 3.2 R2 . . . 20
- 3.3 Van Kreveld . . . 22

4 Results **23**
- 4.1 Correctness . . . 23
- 4.2 Accuracy Tests . . . 24
- 4.3 Timing Tests . . . 26
  - 4.3.1 Locally . . . 26
  - 4.3.2 Distributed . . . 28

5 Discussion **33**
- 5.1 Conclusion . . . 33

5.2 Future Work 34

Bibliography **35**

A Definitions **I**

List of Figures

1.1 A viewshed analysis over an area at Lake Tahoe, CA, USA. The eye denotes the position of the observer and the visible areas are marked in blue. 2

2.1 A raster is a two-dimensional matrix, that divides a map in cells. A sample map (a) is overlayed with a raster (b), creating a rasterized map (c). The higher the resolution in the raster, the more rows and columns in the matrix resulting in more and smaller cells, as shown in (d). Every cell in the matrix represents a subarea of the map and the height information for this subarea is stored in the corresponding cell in the raster. 6

2.2 Side view of a LoS between observer O and points A, B and C. Point A is visible as there is no part of the terrain between O and A that obstructs the ray from O to point A, the same is true for point C. Point B is not visible as the ray between O and B is obstructed by the hill located between the points. 7

2.3 The elevations contained in a raster-based DEM do not consider the curvature of the earth. When performing viewshed analysis, a correction value, δE, needs to be subtracted from each elevation value in the DEM. 8

2.4 Viewshed: A LoS is calculated from the observer O to all the points in the area. 9

2.5 The impact of DEM raster resolution on viewsheds calculated from the center of an area of interest, marked by an eye. The darker areas represent rectangular prisms placed on a flat surface and the diagonally striped cells are considered visible. 10

2.6 A LoS from (0,0) to (4,2), the dot is the target point for which to calculate visibility. The circles mark the x-crossings where height values will be sampled to use in calculation of the target point's visibility. . 11

2.7 An area-of-interest divided into eight octants. 11

2.8 The R2 algorithm calculates a LoS to points A and B, and stores LoS height information on all x-crossings. Point C's LoS height will be determined by the LoS that pass closest to point C. In this example the LoS to point A will determine the LoS height of point C, as $d_A \leq d_B$. 12

2.9 van Krevelds algorithm uses a sweep line, that rotates around the observer and calculates the visibility of a cell when it passes over the cell's center. . . . 13
2.10 An example binary search three that is used in the van Kreveld-algorithm. α is the angle between the cell and the observer and α_{max} is the highest angle on any cell that are positioned between the cell and the observer. . . . 14
2.11 Three different events for a point, where a sweep-line rotates counter-clockwise around the observer. Line A shows the sweep-line where it enters the point, line B shows the position of the sweep-line during the center-event, and the last line, line C, shows the exit-event. . . . 14
2.12 The area contained between the two angles, A_1 and A_2, represents one part of the whole viewshed-analysis in the parallelized van Kreveld algorithm. This part can be calculated independently, therefore making it possible to parallelize the algorithm. . . . 15
2.13 A sample configuration of a Spark cluster with five nodes. . . . 16
2.14 The components of a Spark application. . . . 17

3.1 Interpolation to get an approximation of the height of a theoretical point that lies between point y_1 and y_2. . . . 19
3.2 The visibility of point (1,1), denoted by the dot, will be calculated by several LoS. The circles denote the points along the LoS which will approximate (1,1) . . . 21
3.3 While an approximation of the height of point (1,1) will be calculated by LoS(2,3) it will not determine the visibility of point (1,1) since LoS(3,3) is closer. . . . 21
3.4 A raster where the gray area represents the cells on the edge of the area-of-interest. . . . 22

4.1 Comparison between our implementation of the R3 algorithm and the algorithm used at Carmenta. . . . 24
4.2 Differences when comparing the R2 and Kreveld algorithms to the R3 algorithm on a 2001x2001 raster DEM representing the city of Paris, France. A negative value indicates that the R3 algorithm reports a higher LoS height than the other algorithms, and vice versa. Values close to zero means that the two compared algorithms agree on the results. . . . 25
4.3 Running time when calculating a viewshed with the R2 algorithm locally, on a raster of size 16001 x 16001, with different number of threads. . . . 27
4.4 Comparison when running the R3 algorithm with different number of threads. . . . 28
4.5 Comparison when running the R2 algorithm with different number of threads. . . . 28
4.6 Comparison when running the van Kreveld algorithm with different number of threads. . . . 29
4.7 Comparison of memory consumption of the three algorithms . . . 29

4.8 Comparison of running times of the R2 algorithm on a 16001x16001 raster using Spark with varying amounts of executors per worker node. 30
4.9 Comparison of running times of the R3 algorithm, between a single multi-core CPU and a cluster using Spark 31
4.10 Comparison of running times of the R2 algorithm, between a single multi-core CPU and a cluster using Spark. 32
4.11 Comparison of running times of the van Kreveld algorithm, between a single multi-core CPU and a cluster using Spark. 32

List of Tables

4.1 Occurrences of differences in reported LoS height values when compared to the R3 algorithm. 25
4.2 Point-by-point comparisons of the visibility results of the R2 and van Kreveld algorithms to R3's results. The matching points are the number of points where the algorithms agree with R3 on visibility. . . 26
4.3 Error statistics when comparing the results of the R2 and van Kreveld algorithms to R3's results using a 2001x2001 raster. 26

1

Introduction

1.1 Background

Geospatial Information Systems (GIS) are used to capture, analyze and present data with geographic or spatial components. With the trend of mobile and web applications becoming more prevalent, the underlying hardware (e.g. mobile phones, tablets, light-weight web servers) used to access these systems might not provide sufficient computational capabilities required by GIS. It can therefore be advantageous to be able to distribute the more computationally heavy procedures to a machine with more capable hardware or to a distributed cluster of servers.

One such computationally heavy procedure is viewshed analysis, the goal of which is to calculate visible subareas for a particular area of terrain using elevation data. The set of visible points is called a viewshed. Figure 1.1 shows a viewshed analysis over an area at Lake Tahoe, CA, USA.

In this thesis we perform an evaluation on the feasibility of distributing GIS computations by distributing viewshed analysis from a single server to a server cluster. This evaluation is performed on behalf of Carmenta[1], a company specializing in GIS. The server cluster is built using existing frameworks and hosted on a cloud-computing service. Distributing computations in this manner is not new in and of itself, which is why we use existing frameworks rather than designing something completely new.

1.2 Problem Description

The aim of this thesis is two-fold. The first part consists of implementing different algorithms for calculating viewsheds in a distributed environment. The algorithms used are described in Section 2.4. The idea is to examine the feasibility of running

[1]http://www.carmenta.com

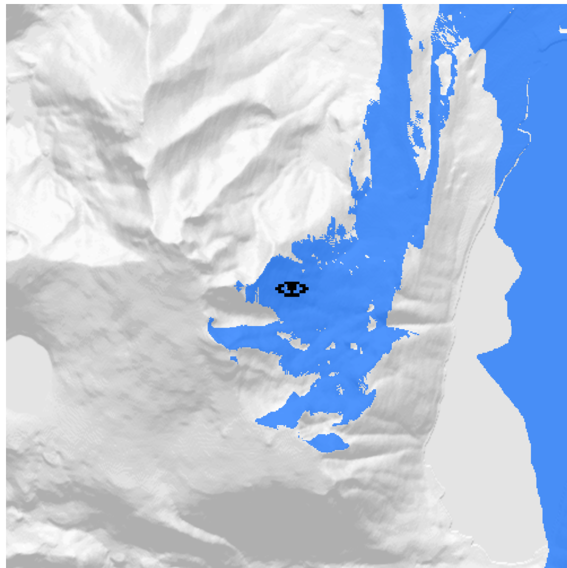


Figure 1.1: A viewshed analysis over an area at Lake Tahoe, CA, USA. The eye denotes the position of the observer and the visible areas are marked in blue.

viewshed analysis on a cluster by using existing frameworks. Some questions we answer are what kind of speed-up, if any, one can expect; how the cluster size (number of nodes) affects the running-time of the viewshed analysis and what changes one might need to make, in order to reduce potential overhead incurred by a distributed solution. The second part is focused on the overall usability of distribution and the chosen frameworks. What would be a suitable application for using a cluster? What determines whether an application is suitable for distribution? What are some of the aspects one should take into consideration when deciding to use distribution on clusters? To answer these questions we need to run timing tests to be able to investigate the feasibility of using distribution, as well as gather information regarding the suitable size of a cluster. We also need to run tests comparing the visibility results of the different algorithms, in order to compare their accuracy.

1.3 Previous Work

Ware, Kidner and Rallings [1] anticipated that the demand for parallel processing when performing geospatial analysis would increase. This idea has been confirmed by more recent work, for instance by Thai and Olasz [2] and Abdul, Potbar and Chahaun [3]. They all agree that this form of distribution is a worthwhile method for increasing the performance of GIS calculations.

According to Tabik et al. [4], performing viewshed analysis is highly useful in a large number of applications, civilian as well as military. It is also well suited for parallelisation as lines-of-sight can be evaluated independently inside an area-of-

interest [5]. Furthermore, since changes in terrain happen slowly, elevation data is fairly static. This means that replicating the elevation data to several distributed nodes is straightforward and does not incur the high messaging overhead commonly found in distributed applications. This makes viewshed analysis a good candidate for distribution.

Axell and Fridén [6] describe in their Master's thesis (also performed at Carmenta) how they were able to obtain faster computations for performing viewshed analysis by utilising GPU parallelisation, compared to a multi-core CPU.

Our thesis examines potential increases in performance when using distributed frameworks and cluster computing. These performance increases are especially useful when concurrently performing multiple viewshed analyses on the same view since even a small reduction in execution time will be amplified over multiple executions. Example use cases where this is applicable is calculating a route that is the least visible from a number of predetermined observation positions or finding the placement of a given number of observers that yields the optimal visibility coverage of a particular area.

1.4 Goals

Our goals with this thesis are to implement algorithms for viewshed analysis with the distribution framework Apache Spark. We also need to measure the accuracy of the implemented algorithms, making sure that they achieve adequate results. In order to compare our distributed solution to an implementation on a multi-core CPU, we need to measure the computation times of running viewshed analyses on both solutions. The last step is to draw conclusions regarding the measured overhead incurred by the distribution framework as well as the size of the cluster and also more general conclusions on the usability of distribution frameworks in GIS applications.

1.5 Limitations

The height data used in all benchmarking will be a raster representation of a Digital Elevation Model (DEM) [7]. A raster is a grid of cells, like a matrix, with elevation data stored in each cell. Other formats for storing the height data are not considered in this thesis.

There are several different algorithms for performing viewshed analysis, in this thesis we consider three of the most commonly used ones: R2, R3 and van Kreveld. Another widely used algorithm is the Xdraw algorithm described by Franklin et al. [8]. Xdraw is among the faster viewshed algorithms but is too inaccurate. According

to Franklin et al. it suffers from a problem where points could influence visibility results even though they should not [8]. Therefore, we choose to not consider it in this thesis in favour of algorithms with higher degrees of accuracy.

Viewshed analysis can be used for applications other than human sight, such as determining where a radio signal emitted from a certain point can be received. These other applications might introduce new considerations for the analysis. In the radio example, the analysis needs to take into account the diffraction of the radio waves. In this thesis we will only consider viewshed analysis in the context of human sight.

We will not investigate either security or fault tolerance in this thesis.

1.6 Overview of Thesis

The structure of this report is as follows: Chapter 2 contains theory about viewshed analysis, as well as a review of the selected algorithms that will be used in this thesis. A detailed description on how we implemented the different algorithms can be found in Chapter 3. Chapter 4 contains the results we gathered, both in terms of accuracy and execution time. Benchmarking was done both locally and on the cluster, to be able to compare the results. These results are then discussed and analysed in Chapter 5, which also contains conclusions regarding the feasibility of using clusters when performing viewshed analysis.

2

Theory

This Chapter presents the theory behind viewshed analysis and the different algorithms that have been used in this thesis. Information about the distribution framework is also be provided.

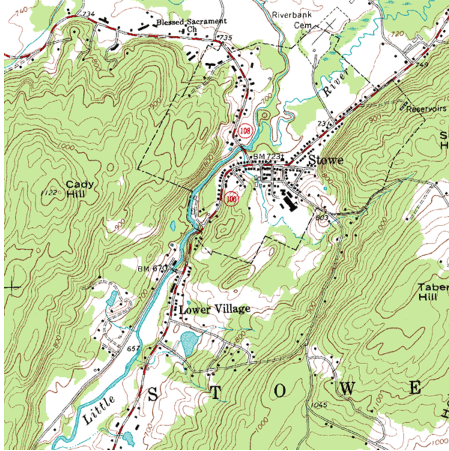
2.1 Digital Elevation Model

A topographic map containing elevation data of an area is called a Digital Elevation Model (DEM), which is a 2.5-dimensional representation of an area. A DEM can be represented with either vector or raster data, with raster-based DEMs being the most common way to represent elevation data [9].

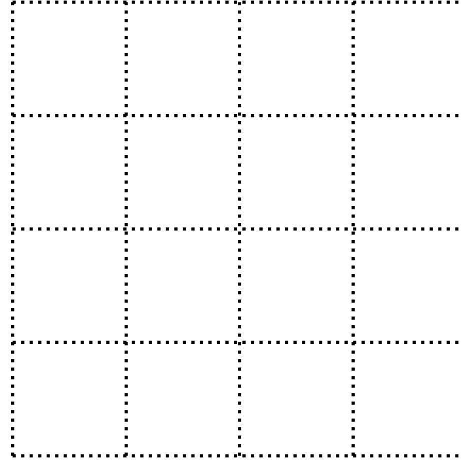
Vector data is visualized with a Triangulated Irregular Network (TIN) [7], which has irregularly positioned triangles that represents a surface. The irregularity comes from the size and positions of the triangles. In an area with little or no difference in elevation, few triangles are needed to represent this area. Conversely, for an area with large differences in elevation, more triangles are needed to sufficiently represent that area.

A raster is a grid of squares, where every cell in the grid represents a small subarea of the map, which is visualized in Figure 2.1. The value stored in a cell is the height information for that subarea. The higher the resolution of the raster, the smaller a subarea represented by a cell will be. This means that a higher resolution will give a more accurate representation of the area. The downside is that the raster will require more memory to store, as the raster will contain more data points. Since the value stored in a cell represents the height of a small subarea it will not be an exact value for every physical point in a terrain (of which there are infinitely many). There are different methods to sample the terrain in order to decide the elevation for a cell. For example, one could use the elevation of the point in the center of a cell, an average over the whole subarea, the minimum elevation in the subarea, the maximum elevation in the subarea or the elevation of the subarea's center. When performing viewshed analysis one needs to consider the sampling method used by

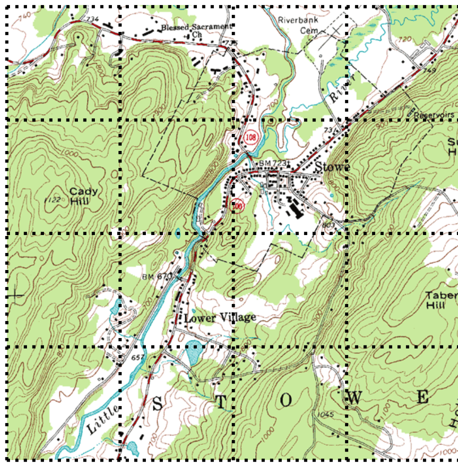
different input data sets since different methods will provide different results.



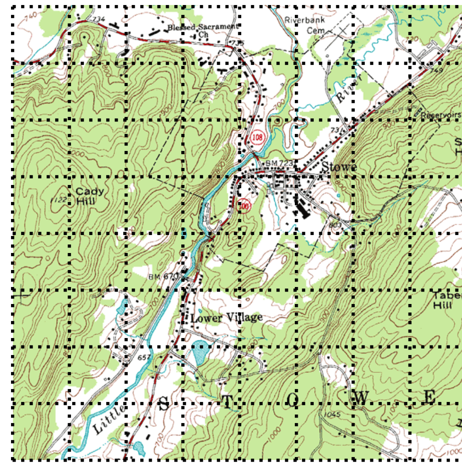
(a) Sample map[1]



(b) Empty raster



(c) Resolution: 4x4



(d) Resolution: 8x8

Figure 2.1: A raster is a two-dimensional matrix, that divides a map in cells. A sample map (a) is overlayed with a raster (b), creating a rasterized map (c). The higher the resolution in the raster, the more rows and columns in the matrix resulting in more and smaller cells, as shown in (d). Every cell in the matrix represents a subarea of the map and the height information for this subarea is stored in the corresponding cell in the raster.

2.2 Line of Sight

Line-of-Sight (LoS) is a procedure where the goal is to calculate the visibility from one point to another point. An example of LoS can be seen in Figure 2.2. The procedure can be visualized by creating a ray in one direction from the observer to

[1] Sample map taken from the public domain USGS Digital Raster Graphic file o44072d6.tif for the Stowe quadrangle, VT, USA

its closest point, this point is always visible, as there are no points that can influence its visibility. Another ray is created from the observer to the next closest point, if this ray is obstructed by any points that are closer to the observer the second point is deemed not visible, otherwise the second point is visible. In Figure 2.2, point A is visible as the ray created between that point and the observer is not obstructed by any points closer to the observer. Point B is not visible as its ray is obstructed. Point C is visible as its ray is not obstructed by any points between point C and the observer.

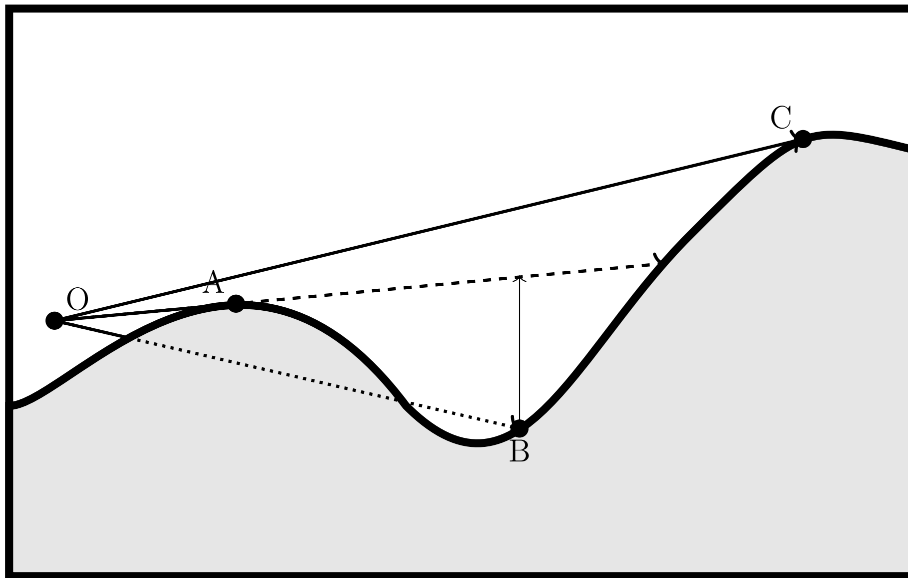


Figure 2.2: Side view of a LoS between observer O and points A, B and C. Point A is visible as there is no part of the terrain between O and A that obstructs the ray from O to point A, the same is true for point C. Point B is not visible as the ray between O and B is obstructed by the hill located between the points.

The check for obstructing rays gives a result in the form of a Boolean, either the cell is visible or not. By calculating the difference in height between a non-visible point and the lowest height needed for that point to be visible, one can calculate the necessary increase in height needed to make something located on that point visible. This can for example be used for determining if an aircraft flying at a certain altitude above the terrain is visible from the observer.

The curvature of the earth needs to be considered when performing LoS calculations using raster DEMs since the rasters contain elevations that are relative to some fixed level, typically sea level, and treats this fixed level as if it were flat. This problem is exemplified in Figure 2.3, which shows how the target point of a LoS calculation is lowered relative to the observation point because of earth's curvature. In order to address this, the elevation of each point in the raster needs to be corrected by subtracting a correction value based on the respective point's distance from the observer. A simple approximation of the correction value, δE, that works well for

distances significantly smaller than earth's radius is

$$\delta E = \frac{D_O^2}{2R_E} \tag{2.1}$$

where D_O is the distance from the observation to the target point and R_E is earth's effective radius [8]. For areas-of-interest with maximum distances of a few hundred meters this correction can be omitted, since the difference in elevation will be insignificant.

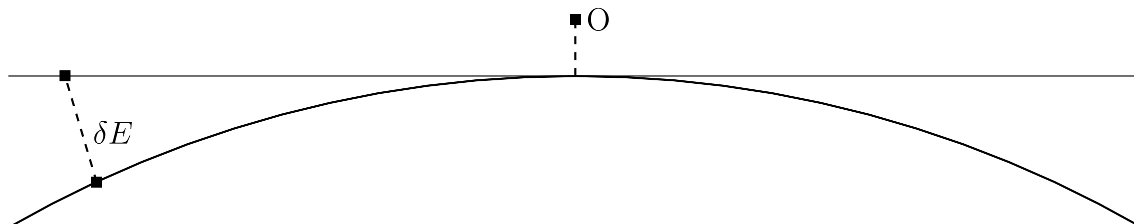


Figure 2.3: The elevations contained in a raster-based DEM do not consider the curvature of the earth. When performing viewshed analysis, a correction value, δE, needs to be subtracted from each elevation value in the DEM.

2.3 Viewshed Analysis

A viewshed analysis calculates the visibility for all points in an area-of-interest. One way to perform this analysis is to calculate multiple LoS from a specific observation point, to all other points in the region of interest. The idea is visualized in Figure 2.4. A LoS is created between the observer and all other points in the region. All points are calculated independently to determine which points, in a specific area, are visible. This is a naive method, described further in Section 2.4.1.

The resolution of a raster DEM used to model an area of terrain will have an impact on any viewshed analysis performed on the DEM. The most obvious impact comes from the generation of the raster, since each cell can only contain one height value. The value will be an approximation of the area the cell covers. A higher resolution raster DEM will be able to provide a closer approximation to the physical terrain since each cell represents a smaller area.

Higher resolutions will also have an impact on the viewshed analysis even if each cell is simply split into a number of smaller cells with the same elevation as the original cell. This can be seen in Figure 2.5, which shows viewsheds calculated with different raster resolutions for an area of interest consisting of a surface with a number of rectangular prisms of equal height capable of obstructing sight. As the resolution increases and each cell becomes smaller, the resulting viewshed is altered. Some areas that were not considered visible are now deemed visible and vice versa.

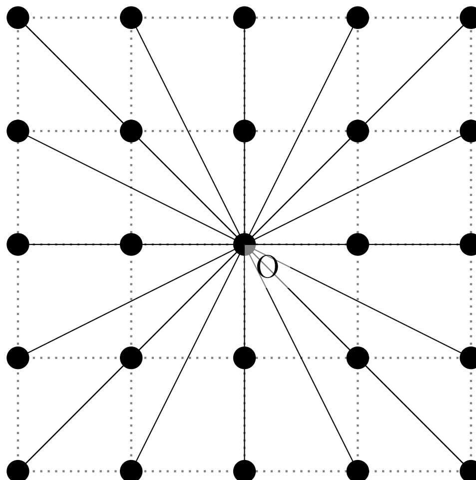


Figure 2.4: Viewshed: A LoS is calculated from the observer O to all the points in the area.

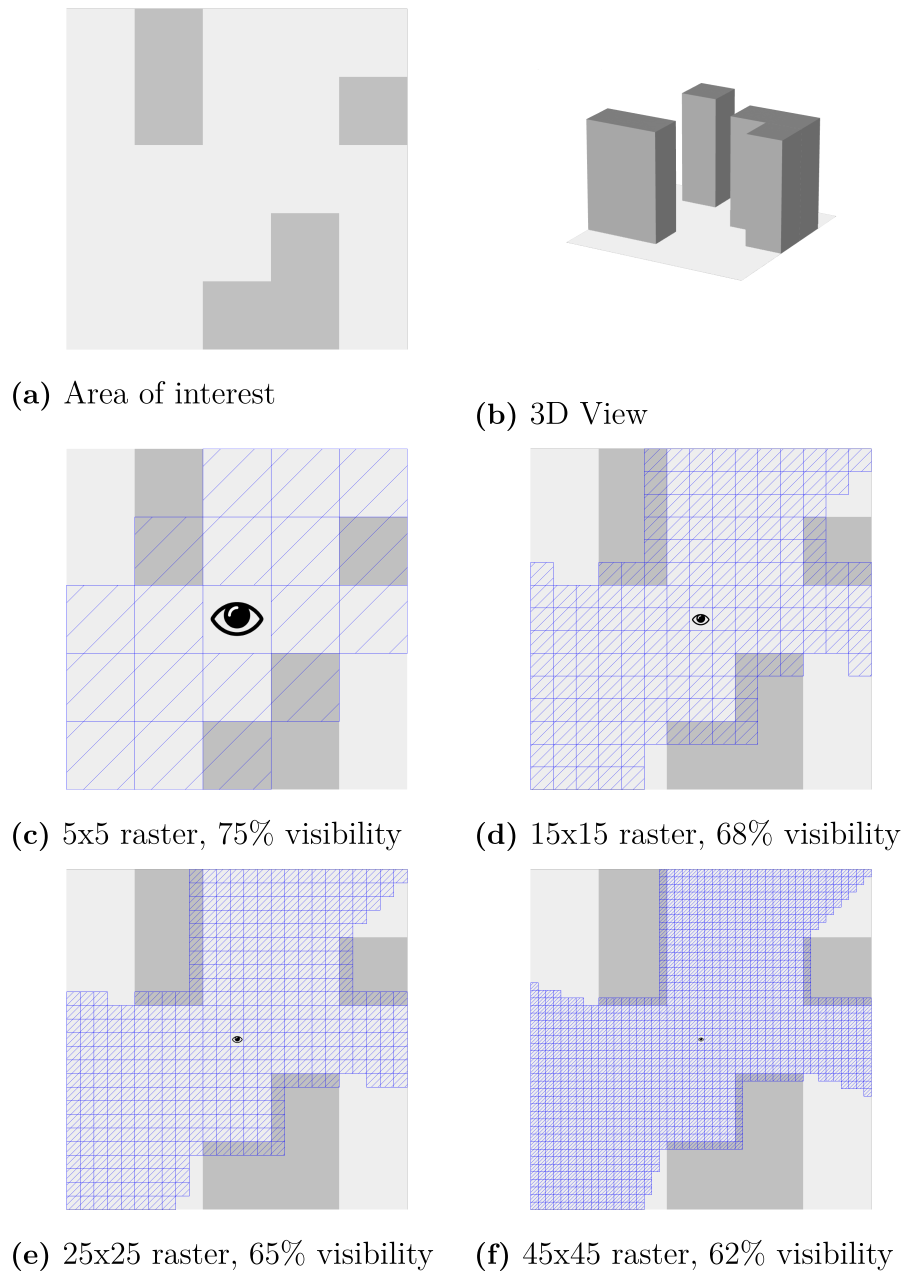
2.4 Viewshed Algorithms

There exist several different algorithms for performing viewshed analysis, each with their own trade-offs. This section aims to give a general idea of how the algorithms used in this thesis work and a brief look at their respective strengths and weaknesses.

2.4.1 R3

The R3 algorithm, described by Franklin et al. [8], is straightforward and highly accurate with relatively low performance, as it scales poorly with the size of the raster. The time complexity is $\mathcal{O}(R^3)$, where R is the side length of the area-of-interest measured in number of points. The idea of the algorithm is to calculate a LoS from the observation point to every other point in the region of interest. Each LoS is traversed from the observation point toward the target point, this concept is visualized in Figure 2.6. How the algorithm traverses a LoS is determined by the position of the target point relative to the observation point. Consider the partitioning of an area-of-interest in Figure 2.7, if the target point is in octant I, IV, V or VIII the elevation is sampled at every whole step along the x-axis, called an x-crossing. For target points in octants II, III, VI and VII the elevation is sampled at every whole step along the y-axis, called a y-crossing. At each step, the slope of a line from the observation point is calculated and the maximum slope encountered thus far along the LoS, s_{max}, is stored. When the LoS reaches the target point, the slope of a line from the observation point to the target point, s_t, is calculated and is compared to s_{max}. The target point is visible from the observation point iff $s_t > s_{max}$.

Since the x-crossings or y-crossings of a LoS will often not fall directly on points in



(a) Area of interest

(b) 3D View

(c) 5x5 raster, 75% visibility

(d) 15x15 raster, 68% visibility

(e) 25x25 raster, 65% visibility

(f) 45x45 raster, 62% visibility

Figure 2.5: The impact of DEM raster resolution on viewsheds calculated from the center of an area of interest, marked by an eye. The darker areas represent rectangular prisms placed on a flat surface and the diagonally striped cells are considered visible.

the raster, the elevation at the crossings needs to be estimated based on the heights of the points on the intersected line closest to the intersection. There are a number of different ways these heights can be estimated with each being useful in different use cases. For instance, if it is important to avoid false positives for visibility, i.e. that a point is marked as visible when it actually is not, an appropriate estimate is to use the maximum height of the two neighboring points.

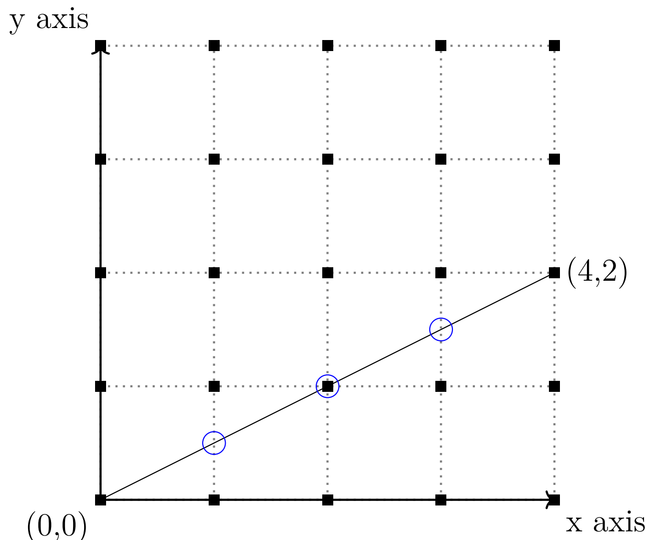


Figure 2.6: A LoS from (0,0) to (4,2), the dot is the target point for which to calculate visibility. The circles mark the x-crossings where height values will be sampled to use in calculation of the target point's visibility.

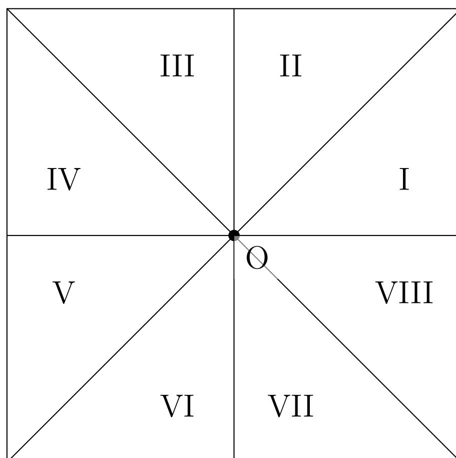


Figure 2.7: An area-of-interest divided into eight octants.

2.4.2 R2

In an effort to reduce the complexity of the R3 algorithm Franklin et al. developed another algorithm, called R2 [8]. The R2 algorithm aims to provide a way to calculate a viewshed in quadratic complexity, $\mathcal{O}(R^2)$, where R is the side length of the area-of-interest measured in number of points. The algorithm will calculate a LoS to points that are on the edge of the viewshed, e.g. point A and B in Figure 2.8, while also calculating the LoS height information for every x-crossing, or y-crossing depending on which quadrant the edge point belongs to, between the observer and the target point. This height information will be used to calculate the visibility of

intermediate points, i.e. point C in Figure 2.8, but only if the distance from the LoS to point C is smaller than the distance of the neighboring LoS to point C. If two LoS have the same distance from a crossing to a point to be calculated, both LoS will calculate the visibility of that point. These conflicts can be handled in different ways, for example to calculate the average or letting each consecutive result overwrite the previous.

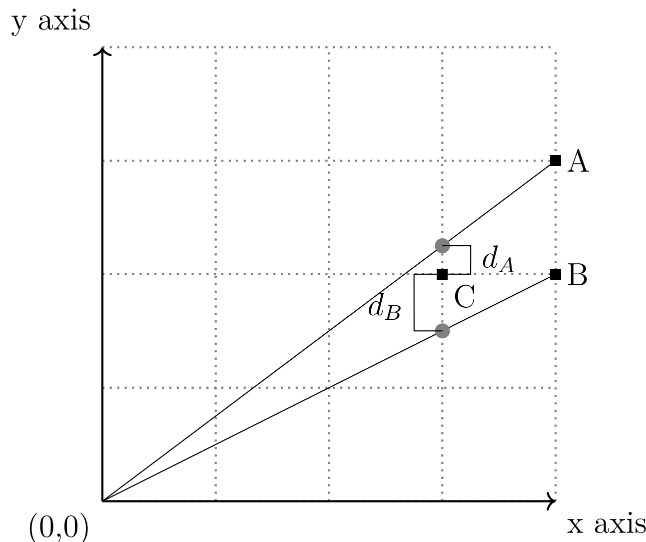


Figure 2.8: The R2 algorithm calculates a LoS to points A and B, and stores LoS height information on all x-crossings. Point C's LoS height will be determined by the LoS that pass closest to point C. In this example the LoS to point A will determine the LoS height of point C, as $d_A \leq d_B$.

2.4.3 Van Kreveld

The algorithm introduced by van Kreveld [10] uses a line that sweeps over an area where a viewshed is to be calculated, as can be seen in Figure 2.9. This is combined with a self-balancing binary search tree (BST) used to keep track of which cells the sweep line currently intersects. The BST (which can be of any type, such as AVL or Red-Black), is sorted by the horizontal distance between a cell and the observer. An example of a BST used by the van Kreveld algorithm can be seen in Figure 2.10. By sorting in this manner, the cell in the left-most leaf in the tree is closest to the observer (smallest distance) and the right-most cell is furthest away (largest distance). All cells have three specific events connected to them: an enter-event, which is when the sweep line first enters a cell; a center-event, which is when the sweep-line intersects the center of the cell; and an exit-event, which is when the sweep line exits the cell. These events are visualized in Figure 2.11. When the sweep line encounters an enter-event it will insert the corresponding cell into to the

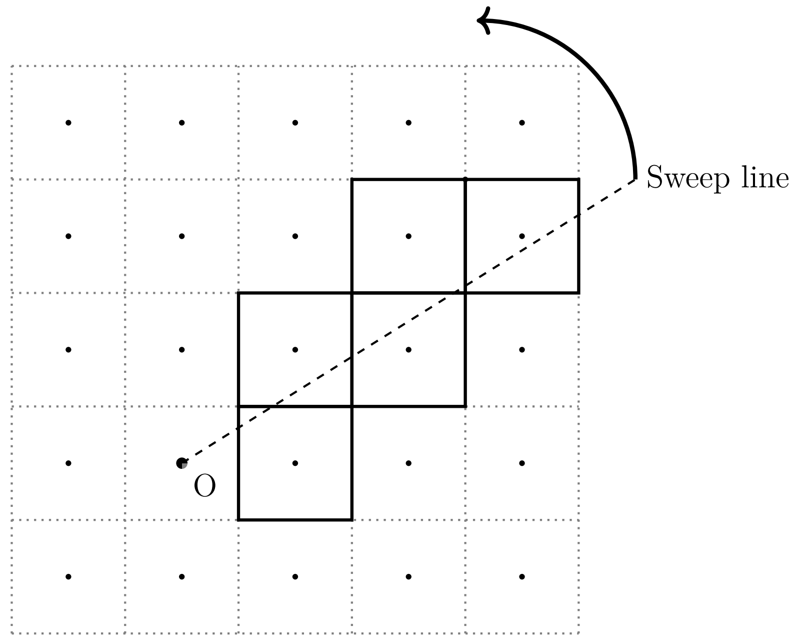


Figure 2.9: van Krevelds algorithm uses a sweep line, that rotates around the observer and calculates the visibility of a cell when it passes over the cell's center.

BST, if the BST is empty the new cell will become the root node. If the event is an exit-event the cell will be removed from the BST.

The last event, the center-event, is when the sweep-line is on the center of a cell and this is the event where the actual visibility computation occurs. Every node in the tree also stores information regarding the maximum angle that any node has calculated from its subtree. The angle for the current cell will be calculated and compared with the maximum value, retrieved from the tree, from nodes that have a lower distance to the observer than the current cell, i.e. the cells that lie between the observer and the current cell. If the current cell's angle is higher than the maximum value from the tree, the cell is deemed visible, otherwise it is not. If the cell is not deemed visible, a calculation is performed to gather how much higher the cell needs to be, to be considered visible. Pseudocode for how the algorithm works can be seen in Listing 2.1.

Listing 2.1: Van Kreveld-algorithm

```
// Calculate angle for enter/center/exit-events
// For all points add events to priorityQueue

While(pQ not empty):
    take head of queue
    check event:
        Enter: add point to tree
        Exit: remove point from tree
        Center: Check if point is visible
```

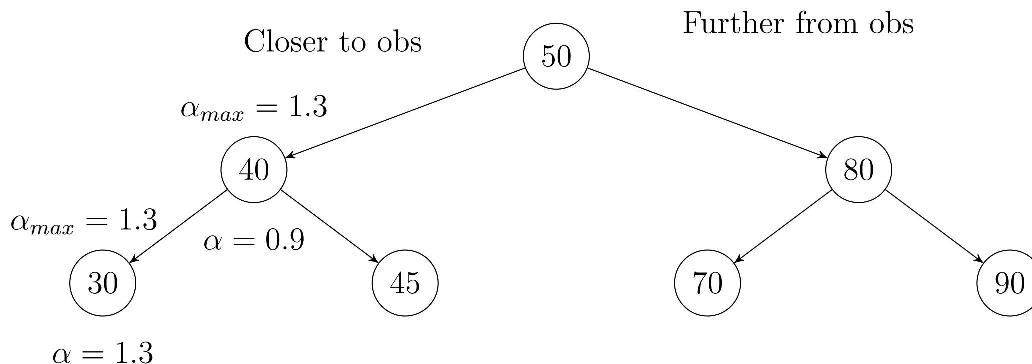


Figure 2.10: An example binary search three that is used in the van Kreveld-algorithm. α is the angle between the cell and the observer and α_{max} is the highest angle on any cell that are positioned between the cell and the observer.

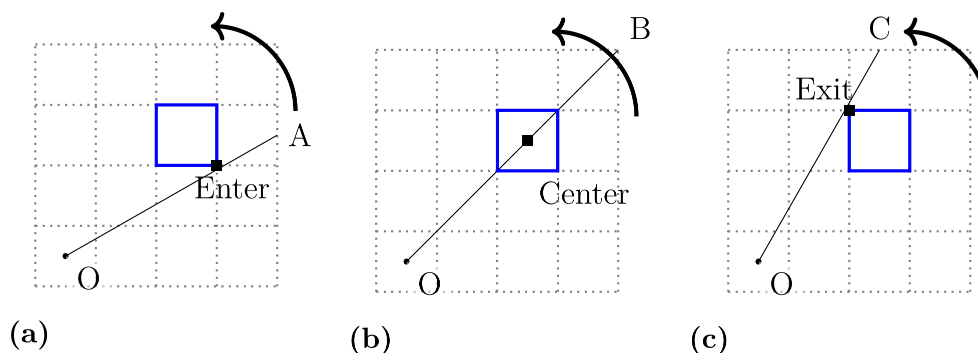


Figure 2.11: Three different events for a point, where a sweep-line rotates counter-clockwise around the observer. Line A shows the sweep-line where it enters the point, line B shows the position of the sweep-line during the center-event, and the last line, line C, shows the exit-event.

Ferreira et al. describe in their paper how to parallelize the van Kreveld algorithm [11]. After creating a representation of the cells with calculated values for the different events, one can divide the list of all enter events and parallelize the calculation on different threads. As can be seen in Figure 2.12, a part of the raster is contained between the two angles A_1 and A_2. Cells whose enter-event or exit-event are between these two angles will be considered one part of the algorithm and can be calculated independently. The list of these cells will be computed by one thread, creating a separate priority queue and binary search tree for that part.

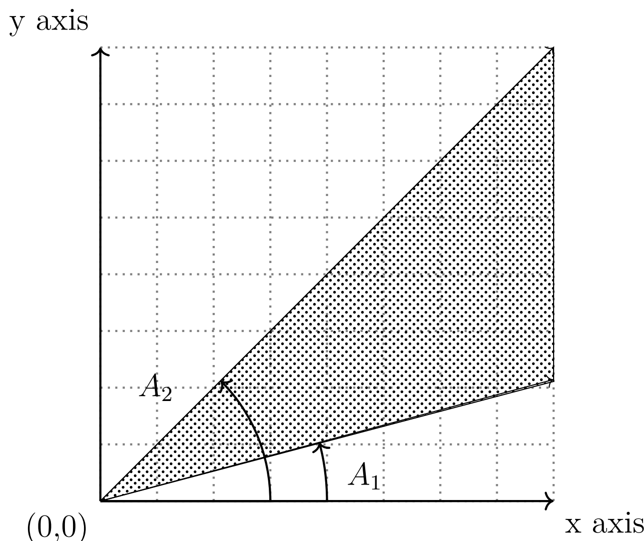


Figure 2.12: The area contained between the two angles, A_1 and A_2, represents one part of the whole viewshed-analysis in the parallelized van Kreveld algorithm. This part can be calculated independently, therefore making it possible to parallelize the algorithm.

2.5 Framework: Apache Spark

To distribute the calculations of the different viewshed algorithms we used the distribution framework Apache Spark. Spark is described as a "fast and general engine for large-scale data processing" [12]. Spark uses Resilient Distributed Datasets (RDDs) [13] which is an immutable dataset, that Spark distributes to the worker-nodes. Spark can perform two types of operations on an RDD, *transformations* and *actions*. Transformations are when functions are applied to each element in an RDD. Due to the partitioning feature of the RDDs, transformations can be applied in parallel. The other operation that Spark can perform on RDDs are called actions. Actions on RDDs will return a value, for instance the count action will return the number of entries that exist in a RDD. Spark uses lazy evaluation, meaning that transformations will not be performed until an action is triggered. Transformations and actions are combined to create a Spark job.

One can achieve better performance from Spark by utilizing its configuration options, as well as optimizing how and where data is stored. Spark provides APIs for four different programming languages: Java, Scala, Python and R.

A cluster is needed to perform distributed computations using Spark, Figure 2.13 shows an example of a Spark cluster configuration. Nodes in a Spark cluster can have two different roles: cluster managers and workers. A cluster manager is responsible for managing the resources of the cluster, i.e. worker nodes, and can either be Spark's built-in standalone manager, or YARN or Mesos. The latter two are general cluster managers that also support other distribution frameworks. The standalone

manager is suitable for simpler applications where advanced resource scheduling is not needed. If one wants to run different distribution frameworks alongside Spark on the same cluster, one needs to use YARN or Mesos. Worker nodes are responsible for performing the computations, i.e. the transformations and actions on the RDDs.

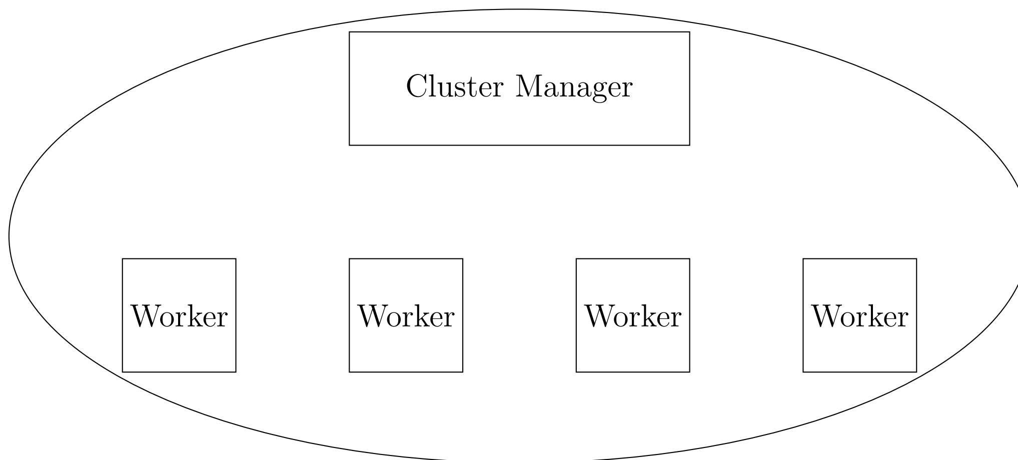


Figure 2.13: A sample configuration of a Spark cluster with five nodes.

Spark applications consist of two components: a driver program and at least one executor, Figure 2.14 illustrates the architecture of these components. The driver manages the connection to the Spark cluster and all work done before distribution, such as setting up common resources shared by all executors. The driver program does not need to be run on the cluster as long as full two-way communication is possible. However, since a lot of communication takes place between the driver and the cluster, it is advantageous to place the driver as close as possible to the cluster physically to minimize latency introduced by network communication. An executor is a process running on a worker node that performs the actual transformations and actions of a Spark job. Each executor is unique to a certain application and exists only during the lifetime of the application. However, it is possible for several different applications to run its own executors on the same worker. A Spark job is divided into tasks which consist of particular transformations or actions performed on partitions of an RDD. An executor can be configured to use more than one CPU core on the worker node on which it resides, if this is the case an executor can perform multiple tasks concurrently.

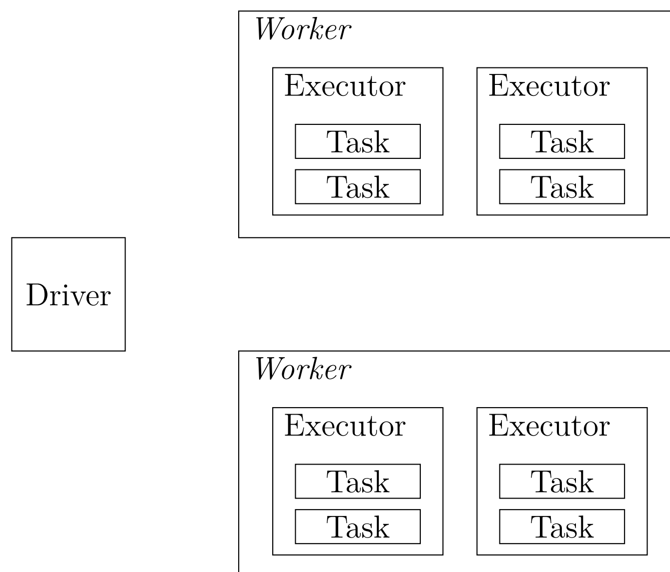


Figure 2.14: The components of a Spark application.

3

Implementation

The viewshed algorithms addressed in this thesis are described in general in Chapter 2, but there are many ways in which each algorithm can be implemented. These implementations can have different characteristics with regards to e.g. computation time and memory requirements. This chapter describes the implementations we made as part of this thesis, describing how they differ from the base algorithms and motivations for the design choices we made.

One thing all our implementations have in common is that they do not only return a Boolean result whether a point is visible or not, they will also report how much a point needs to be raised in order to be considered visible. For example, a value of zero corresponds to a visible point and a value of one corresponds to the number of meters (in this case, one meter) that a points needs to be elevated to be visible from the observation point. All implementations created in this thesis were made in the programming language Java, both for the local implementations and our distributed solutions.

3.1 R3

The R3 algorithm, described in Section 2.4.1, is a rather naive approach to viewshed analysis. In order to handle a LoS that does not fall directly on a point when it intersects an x or a y-crossing, our algorithm interpolates the height from the two points closest to the intersecting point, y_1 and y_2. The code for this calculation can be seen in Listing 3.1, and a descriptive figure can be seen in Figure 3.1

Listing 3.1: Interpolation, for LoS intersection on a x-crossing.

```
// y  = y−coordinate for LoS on crossing
// y1 = closest integer ≤ y
// h1 = height information for coordinate (x,y1)
// y2 = closest integer ≤ y
// h2 = height information for coordinate (x,y2)

intersectHeight =   abs(x−y1) * h2 +
                    abs(x−y2) * h1
```

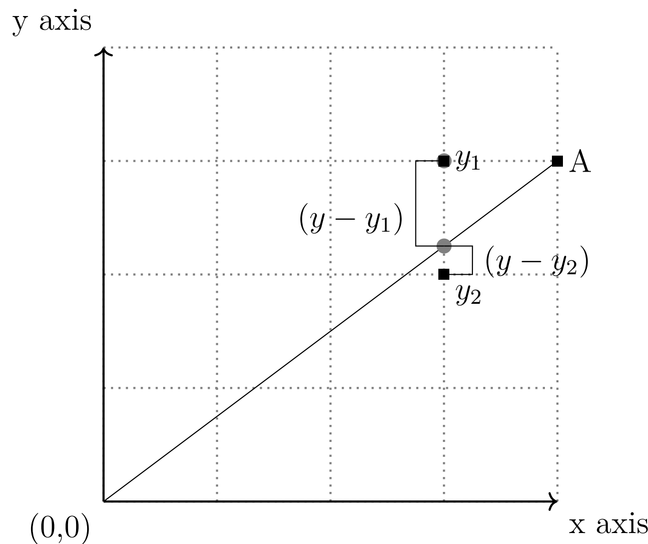


Figure 3.1: Interpolation to get an approximation of the height of a theoretical point that lies between point y_1 and y_2.

The y value is the exact position of the intersection in an x-crossing, y_1 is the y value rounded up to the closest integer, to get the height information from that point in the raster. y_2 is the y value rounded down. The value we get from taking `abs(y-`y_1`)` is the distance between y and y_1, which is a value between zero and one. This value is then multiplied with the height value from the other point (y_2) and then added to the same calculation for y_2. This way the point that is closest to the intersecting point will have the most influence on the resulting height value.

Interpolation is the preferable choice of approximation when the most accurate result is needed [8]. The downside is that it is a more expensive computation compared to for example taking the minimum or maximum value. Maximum and minimum is used when one wants to obtain results that are more conservative towards either visibility or non-visibility. For example, by taking the maximum the LoS-results will probably be higher than the correct result, which would mean that a point that is deemed visible, is visible with a high probability. In contrast, by taking the minimum, a point that is deemed not visible, is not visible with a high probability.

Our implementation of the R3 algorithm in Spark consists of a main function that will read a raster and transform it into a list of points, for all of the cells in the raster. These are later used as keys in a key-value lookup to get the result corresponding to each cell. This list is transformed into an RDD. To calculate the viewshed we created a function that takes a coordinate as input and returns a key-value pair. The value is the visibility height for that cell. The algorithm was then run by applying this function to every element in the RDD, effectively performing a LoS calculation for every point in the raster. The result is a key-value map with a point as the key, and the value is either zero for a visible cell, or a positive decimal value for how much higher a specific cell needs to be elevated, to be considered visible.

3.2 R2

The R2 algorithm calculates visibility for all points on the grid a LoS passes and not only those it actually intersects. This fact, combined with the fact that each LoS is being calculated independently from all other LoS, means that many cells will have their visibility calculated several times. This is illustrated in Figure 3.2. The highest number of visibility calculations for all nodes can be calculated with the formula $4R^2 - 12R + 8$, where R is the side length of the area-of-interest measured in number of points. Each point can only have one visibility value so the results of the different calculations need to be merged. In a distributed setting this merge can be very costly in terms of computation time since conflicts can occur on different nodes. This means that all results have to be aggregated on a single node for the merge to complete successfully, which causes a lot of overhead in the form of communication and data transfer between nodes.

In order to address the problem of duplicates, we modified the R2 algorithm so that each LoS keeps track of the destination of its' neighboring LoS. When a LoS calculates the height of a point, it uses the information about the neighboring LoS to compute the distance from the point to each line. Only if none of the neighboring lines are closer to the point, the current LoS will calculate the visibility for that point. This way the number of duplicates is greatly reduced and since each line needs no information about the actual results of its neighboring lines, each LoS can still be calculated in parallel. Figure 3.3 provides an example of this modification.

The implementation of R2 in Spark is similar to that of R3, with two exceptions. Instead of creating a list with all coordinates we only used coordinates for the cells on the edge of the area-of-interest, visualized in Figure 3.4. The visibility function for R2 takes a coordinate as input and returns a list of key-value pairs as a result. This list corresponds to the result of all cells that are on the LoS from the observer to the edge cell. To remove duplicates we also applied a combining function on the result. As the algorithm only saves results from a calculation if the active LoS is the one that is closest to the cell that is to be calculated, we know that any duplicates have the same distance between the calculated cell and the LoS, therefore

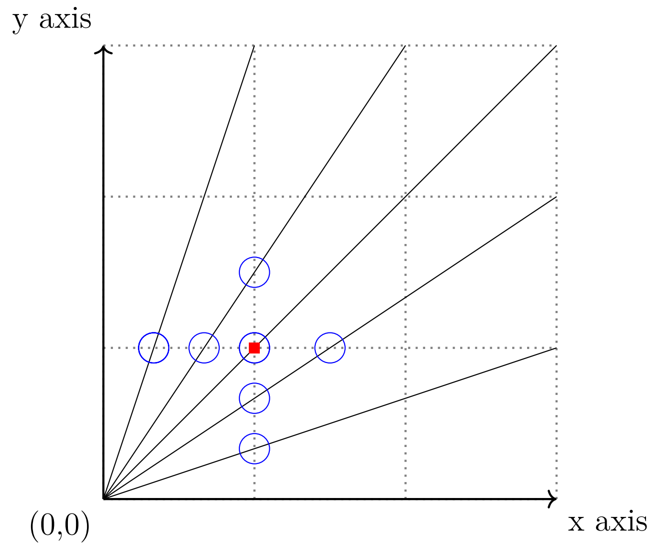


Figure 3.2: The visibility of point (1,1), denoted by the dot, will be calculated by several LoS. The circles denote the points along the LoS which will approximate (1,1)

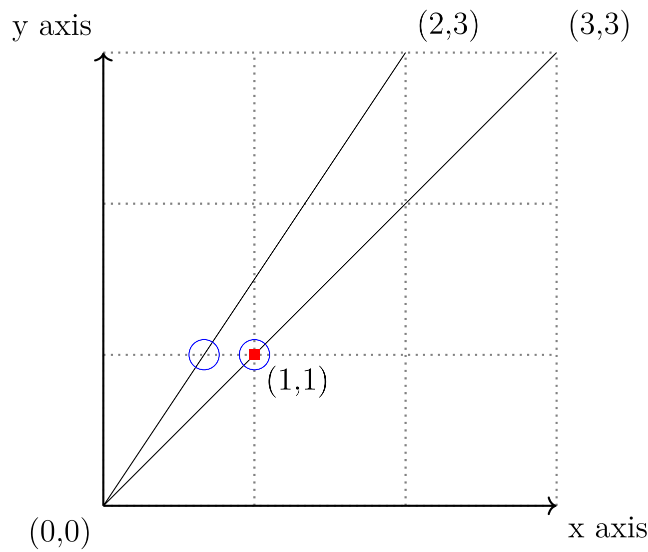


Figure 3.3: While an approximation of the height of point (1,1) will be calculated by LoS(2,3) it will not determine the visibility of point (1,1) since LoS(3,3) is closer.

the combining-function takes the average of the two reported results.

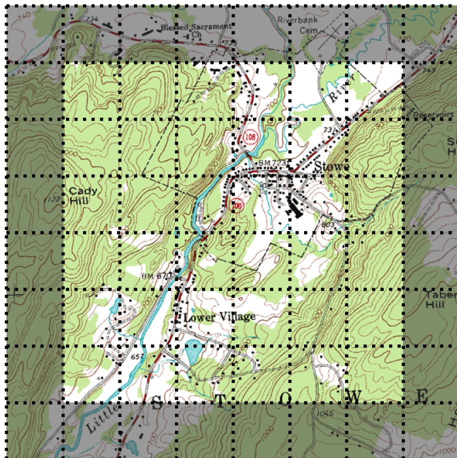


Figure 3.4: A raster where the gray area represents the cells on the edge of the area-of-interest.

3.3 Van Kreveld

Our implementation of the van Kreveld algorithm in Spark uses a different approach for parallelization than described in 2.4.3. Rather than partitioning the raster based on the angles of individual points in the raster, our implementation uses the coordinates of the points. The function we implemented takes a start and an end coordinate as input, these coordinates represent the part of the raster that should be calculated. The function then creates its own priority list of events, as well as a binary search tree. The algorithm would then be computed, but only between the designated start and end coordinate. The reason for using coordinates instead of angles, as shown in Figure 2.12, is that there is no need to calculate the angles for all cells before dividing the raster, which makes it possible to calculate the angles in parallel as this is done after the partitioning. The result from running the algorithm is a list of key-value pairs of all the cells in this part of the raster, with the key corresponding to the coordinate of the cell and the value corresponding to the height-information calculated by the algorithm.

The van Kreveld algorithm differs somewhat from the R2 and R3 algorithms when calculating visibility heights of points. The R3 and the R2 algorithms use interpolation, as shown in Listing 3.1, whereas the van Kreveld algorithm uses the maximum height of obstructing points. The effect of this is that points that the van Kreveld algorithm deems visible, will most certainly be visible. Furthermore, van Kreveld might report points as not visible that the two other algorithms will report as visible.

4

Results

This chapter describes the results we gathered from running tests on our implementations. The results include accuracy and time complexity and are used to compare the algorithms as well as comparing local computations with running them on a cluster. The hardware used locally was a computer with 16 GB RAM, running an Intel i7 870 at 2.93 GHz [14], the JVM was set to use 10 GB of RAM. The Spark tests were performed on the Google Cloud Engine platform [15] using a cluster consisting of five *n1-highmem-4* instances [16], with one instance acting as the cluster manager, using YARN, and the rest as workers.

4.1 Correctness

To gather results and to test our implementations of the R2 algorithm and the van Kreveld algorithm we compared the results between them and the R3 algorithm. The reason for this is that the R3 algorithm, as previously stated, calculates an individual LoS to every cell in the raster, meaning that approximation errors are kept to a minimum. To be able to use results from the R3 algorithm we first needed to assure that it would give accurate results. To do this we compared it to the algorithm used by Carmenta, which we used as a base case. The comparison is visualized in Figure 4.1, where points visible only according to Carmenta's algorithm are coloured red, points deemed visible only by our R3 are coloured light blue. The darker area denotes points where both algorithms agree are visible. The reason for the large amount of points visible only to our algorithm is that the algorithm used by Carmenta only calculates viewshed for a circular area and not the entire area-of-interest. There are some points inside the circle that the algorithm used by Carmenta deems visible, which the R3 algorithm does not. This is probably because of some of the optimisations that are implemented in the Carmenta algorithm. Overall the two algorithms agree on a vast majority of points, which means that our implementation can be considered correct and accurate.

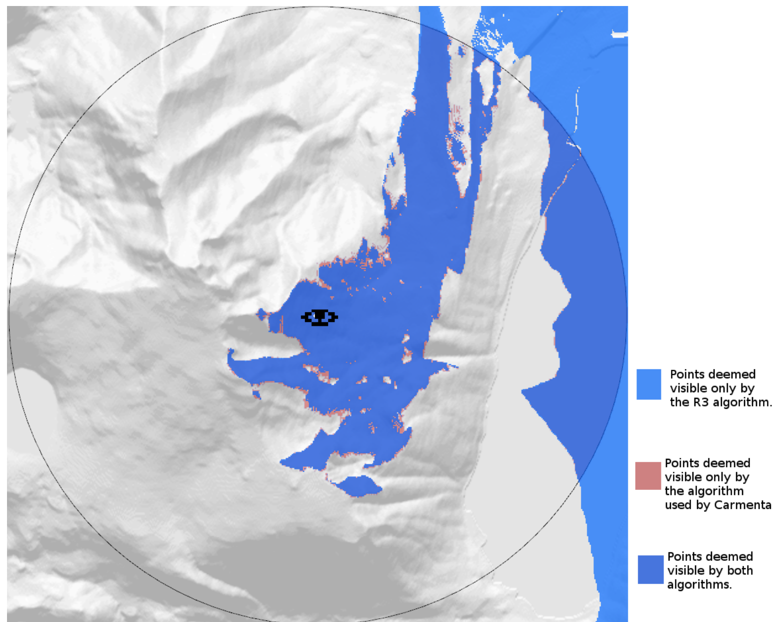


Figure 4.1: Comparison between our implementation of the R3 algorithm and the algorithm used at Carmenta.

4.2 Accuracy Tests

In Figure 4.2 the graph visualizes the comparison between the R2 and Kreveld algorithms to the R3 algorithm, the same data can be viewed in Table 4.1. Desirable values are zero, or close to zero, as this means that there are little to no difference in the reported results from the algorithms. For R2, the vast majority of the values are concentrated around zero which means that we have achieved high accuracy in our implementation of the R2 algorithm. The van Kreveld algorithm also have most values concentrated around zero, although slightly more spread out than for R2. An interesting result is that the van Kreveld comparison does not display the same symmetric behaviour as R2. This can likely be attributed to the different ways the algorithms estimate the height of obstructing points that lie between real points in the raster. Van Kreveld uses the maximum height of the two closest real points whereas R2 and R3 both use interpolation.

Table 4.2 shows a comparison on how many points the algorithms agree are visible or not. Our results show that both the R2 and van Kreveld algorithms have a high degree of agreement compared to R3, around 99%, but R2 has a slight edge.

Table 4.3 shows error metrics when comparing the R3 algorithm to both the R2 algorithm and the van Kreveld algorithm. The error metrics of the R2 algorithm are comparable to the values presented in the paper by Franklin et al. [8]. When evaluating the results from our tests, we can gather that both algorithms have sufficient accuracy, though R2 reports somewhat better results than the van Kreveld algorithm.

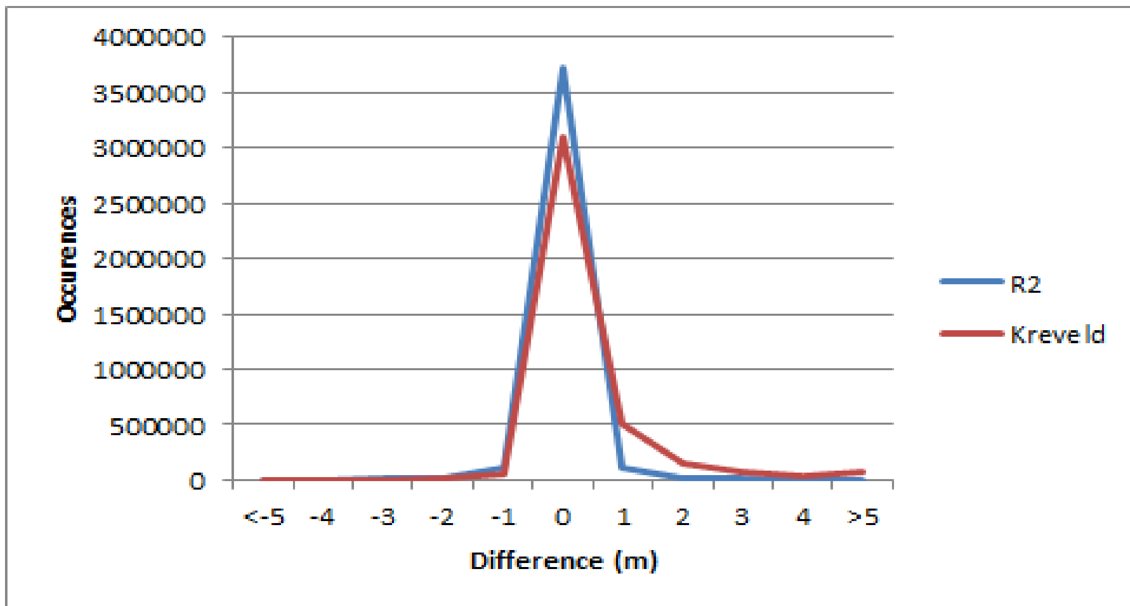


Figure 4.2: Differences when comparing the R2 and Kreveld algorithms to the R3 algorithm on a 2001x2001 raster DEM representing the city of Paris, France. A negative value indicates that the R3 algorithm reports a higher LoS height than the other algorithms, and vice versa. Values close to zero means that the two compared algorithms agree on the results.

	R2	Kreveld
$\leq$-5	4232	1419
-4	4300	2046
-3	8877	4475
-2	22096	11292
-1	106401	45741
0	3717201	3100835
1	103539	515917
2	21061	144766
3	8113	63221
4	4107	37520
$\geq$5	4073	76768

Table 4.1: Occurrences of differences in reported LoS height values when compared to the R3 algorithm.

	R2	Kreveld
Total Points	4004000	4004000
Matching Points	3977861 **(99.3%)**	3953654 **(98.7%)**

Table 4.2: Point-by-point comparisons of the visibility results of the R2 and van Kreveld algorithms to R3's results. The matching points are the number of points where the algorithms agree with R3 on visibility.

	R2		Kreveld	
Mean error	-0.0026	meters	0.4455	meters
Min error	-13.88	meters	-11.01	meters
Max error	13.05	meters	45.89	meters
Std deviation	0.49	meters	1.50	meters

Table 4.3: Error statistics when comparing the results of the R2 and van Kreveld algorithms to R3's results using a 2001x2001 raster.

4.3 Timing Tests

To be able to draw conclusions on the performance between the different algorithms and the different implementations we performed timing tests. The recorded times exclude the time it takes to read and parse the raster.

During the initial timing tests we faced a problem where we did not have enough RAM for storing each LoS result. To solve this problem, and to ensure that the times we reported were the running time of the actual algorithm, we chose to save the result from every individual LoS in the same variable. Thereby reducing the memory problem, while still running the whole algorithm. The downside of this solution is that the result from a previously computed LoS will be overwritten by the next.

4.3.1 Locally

The results from running the R2 algorithm on our local implementation are shown in Figure 4.3. The times are an average over ten runs performed on the multi-core CPU described previously. As can be seen, the execution time gets lower and lower depending on the number of threads the algorithm executes on.

The results from running the R3 algorithm with different numbers of threads can

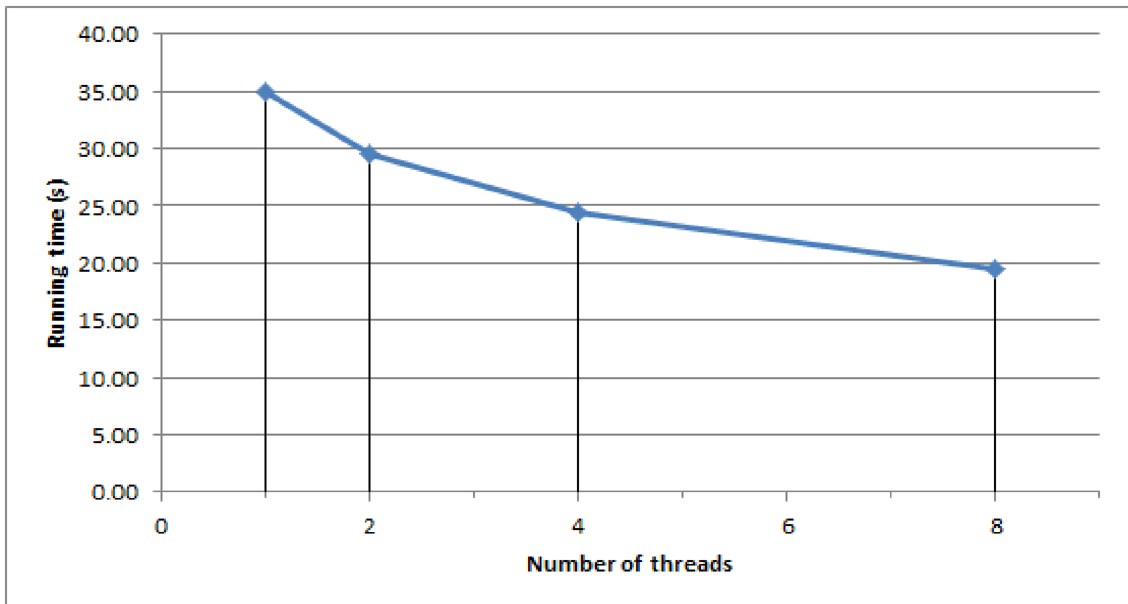


Figure 4.3: Running time when calculating a viewshed with the R2 algorithm locally, on a raster of size 16001 x 16001, with different number of threads.

be seen in Figure 4.4. It is easy to see the difference in complexity between the two algorithms. The time it takes to run the R3 algorithm on a raster with size 2001 x 2001 is comparable to running the R2 algorithm on a raster with size 16001 x 16001, a raster of almost 64 times the size.

All three algorithms were implemented so that they could be run in parallel. The running time when executing the algorithms with different numbers of threads are shown in Figure 4.4, Figure 4.5 and Figure 4.6. The results show that running an algorithm with more threads will decrease the overall running time, which in turn implies that we have succeeded in our parallel implementation of the algorithms. The R2 algorithm receives a smaller speed-up when using multiple threads compared to both the R3 and van Kreveld algorithms. This is probably due to that the running time of the algorithm is so short that the overhead incurred by running the algorithm in multiple threads have a measurable impact on the total running time.

One problem with the van Kreveld algorithm is that it uses a lot of RAM. Memory usage is visualized in Figure 4.7. For a raster with a side length of 5001 cells, the van Kreveld algorithm uses more than three times the RAM compared to the other algorithms. Due to restrictions on our testing hardware, the largest raster we were able to run the van Kreveld algorithm on had a side length of 6001. By comparison, on our cluster we were able to run a viewshed analysis with the van Kreveld algorithm on a raster with side length of 10001 cells. This implies that by using a cluster's ability to scale, one could successfully perform calculations on large rasters that would otherwise not be possible on a single machine.

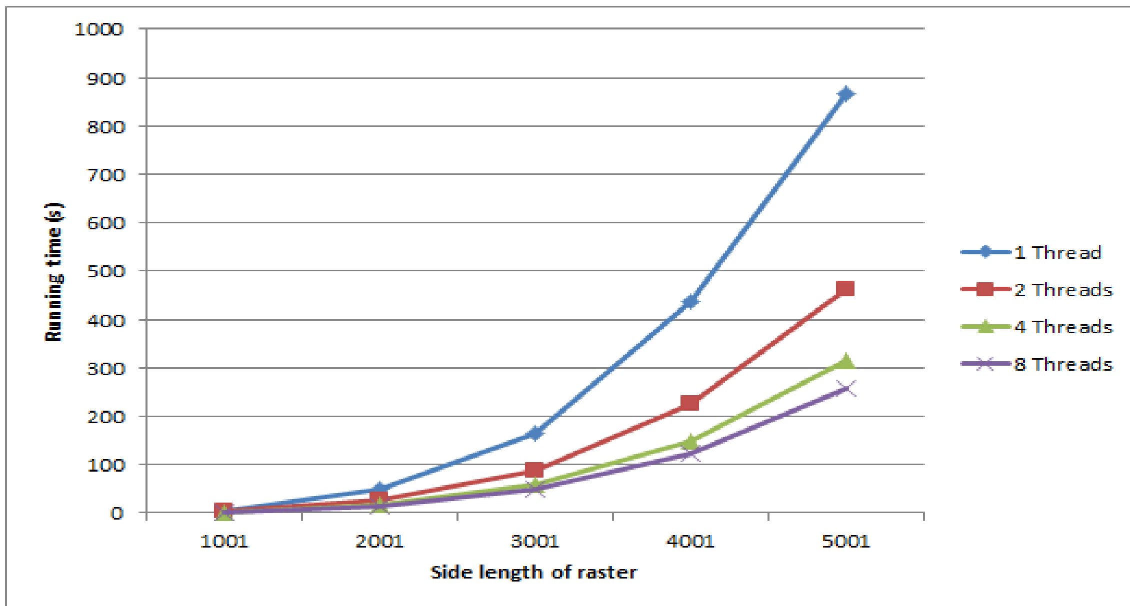


Figure 4.4: Comparison when running the R3 algorithm with different number of threads.

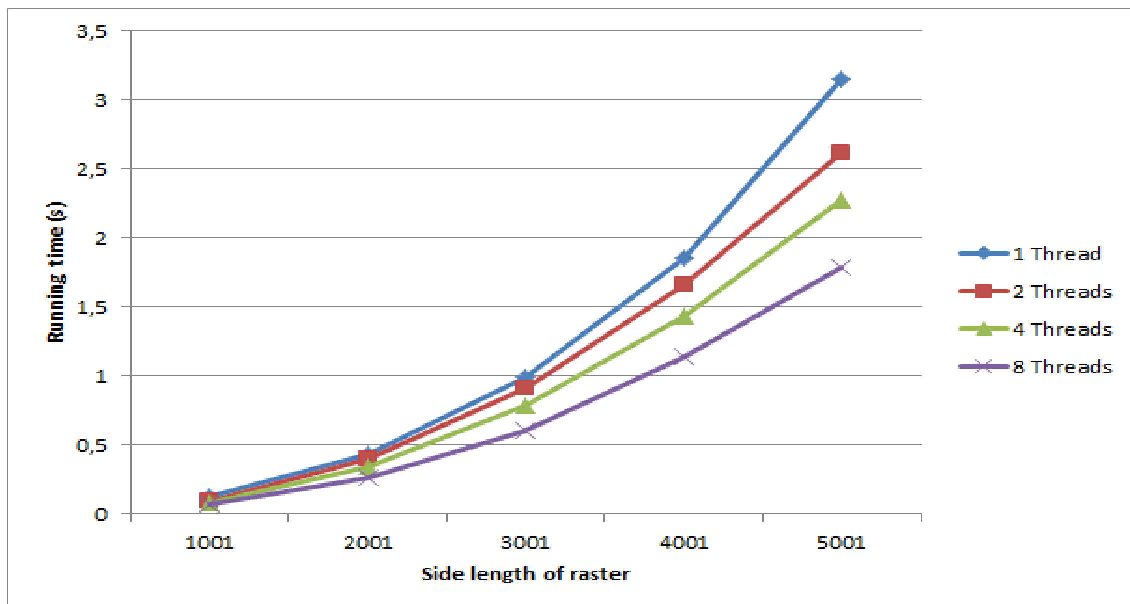


Figure 4.5: Comparison when running the R2 algorithm with different number of threads.

4.3.2 Distributed

In order to evaluate how the configuration of the Spark cluster impacts the running time of viewshed analysis we ran a series of tests of the R2 algorithm using a 16001x16001 raster with varying number of CPU cores per executor, which determines how many executors can run on a single worker node, and number of cores in total. The results of these tests are displayed in Figure 4.8. We found that all the

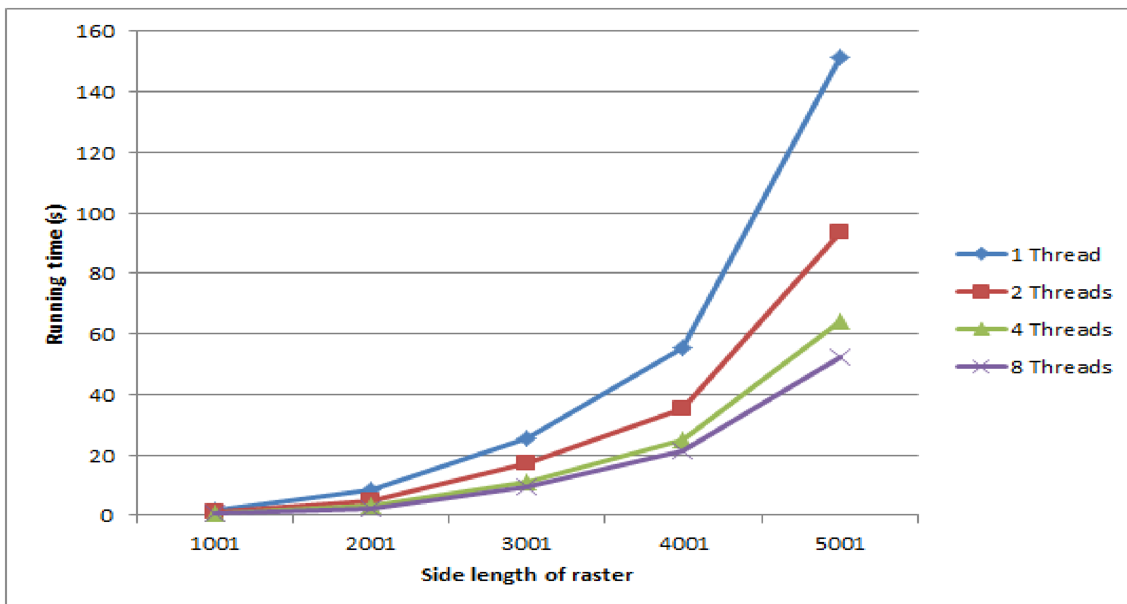


Figure 4.6: Comparison when running the van Kreveld algorithm with different number of threads.

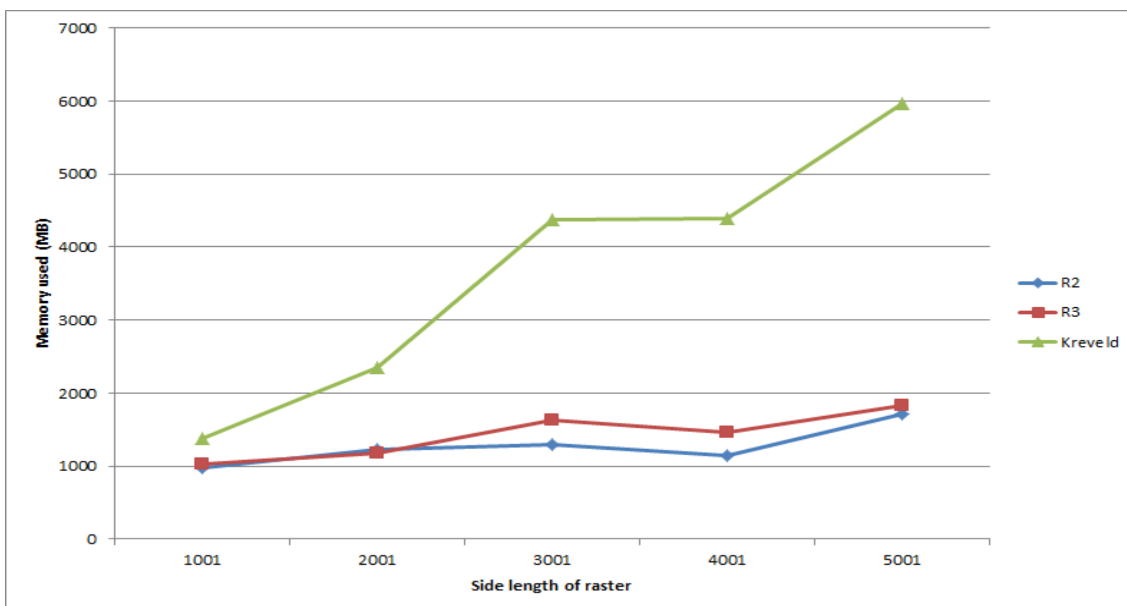


Figure 4.7: Comparison of memory consumption of the three algorithms

tested variations of cores per executor behaved similarly with regards to the running time. Using two cores per executor, resulting in two executors on each worker, yielded slightly better results and was therefore used in all subsequent timing tests performed on the Spark cluster.

Even though the R3 algorithm has a higher complexity than the R2 algorithm, we still wanted to gather results from running it. The reason for this is that we wanted to compare running times on a more complex algorithm between the local imple-

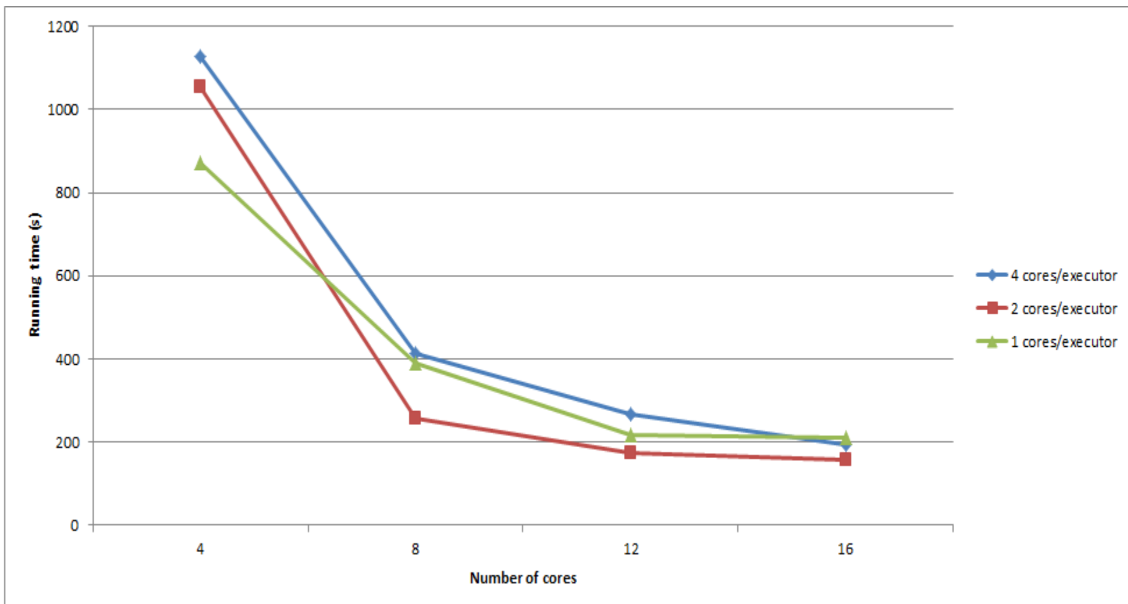
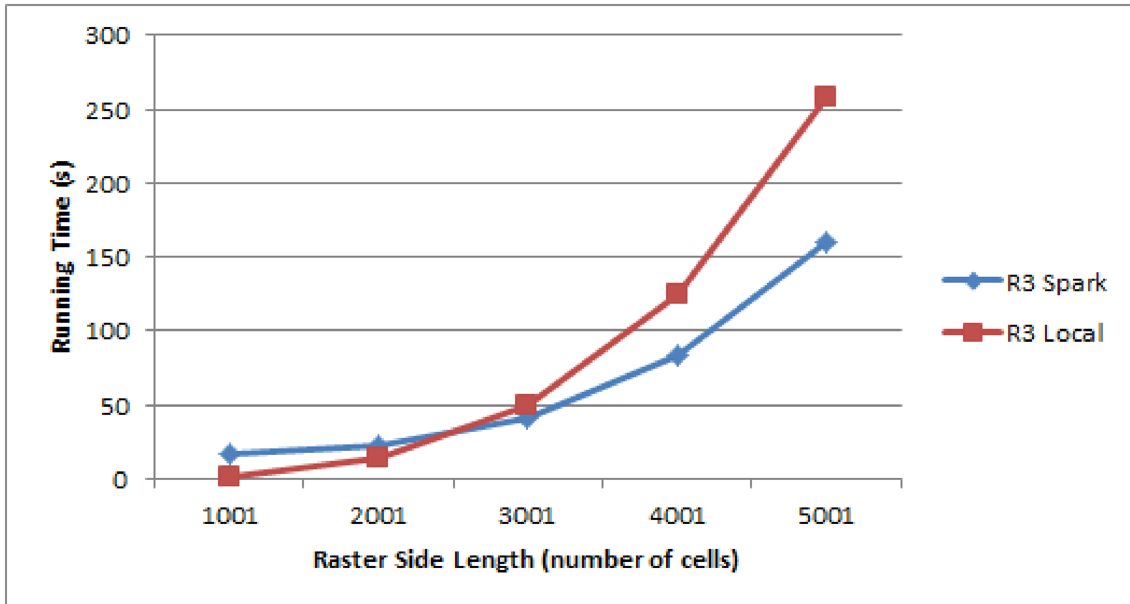


Figure 4.8: Comparison of running times of the R2 algorithm on a 16001x16001 raster using Spark with varying amounts of executors per worker node.

mentation and the Spark implementation, to be able to draw conclusions whether other, more demanding, computations can benefit from being run on a cluster. The results of this comparison can be viewed in Figure 4.9. According to the results, Spark calculates the viewshed faster when compared to the local implementation. What one has to keep in mind when comparing the running times is that in this instance Spark was run on a cluster with a total of 16 virtual cores and the local implementation was run with 8 threads on a CPU with 8 virtual cores. Even though the hardware differs, it is still interesting to consider that Spark outperforms the R3 local implementation. We did the same comparison for the R2 algorithm, which can be viewed in Figure 4.10. It is apparent that the local implementation outperforms our distributed solution, and there is nothing that points towards a theoretical break-point for larger rasters where the distributed solution would perform better than the local one.

Figure 4.11 shows a comparison between the van Kreveld algorithm run locally and on the cluster. The two implementations have comparable performance, the reason that the measuring points for the local implementation stop at 6001 cells is that the algorithm demanded too much internal memory, and was unable to calculate rasters of greater size.



(a) Measurements of running times of the R3 algorithm, between a single multi-core CPU and a cluster using Spark.

(b) Polynomial regression based on the findings presented in Figure 4.9a.

Figure 4.9: Comparison of running times of the R3 algorithm, between a single multi-core CPU and a cluster using Spark

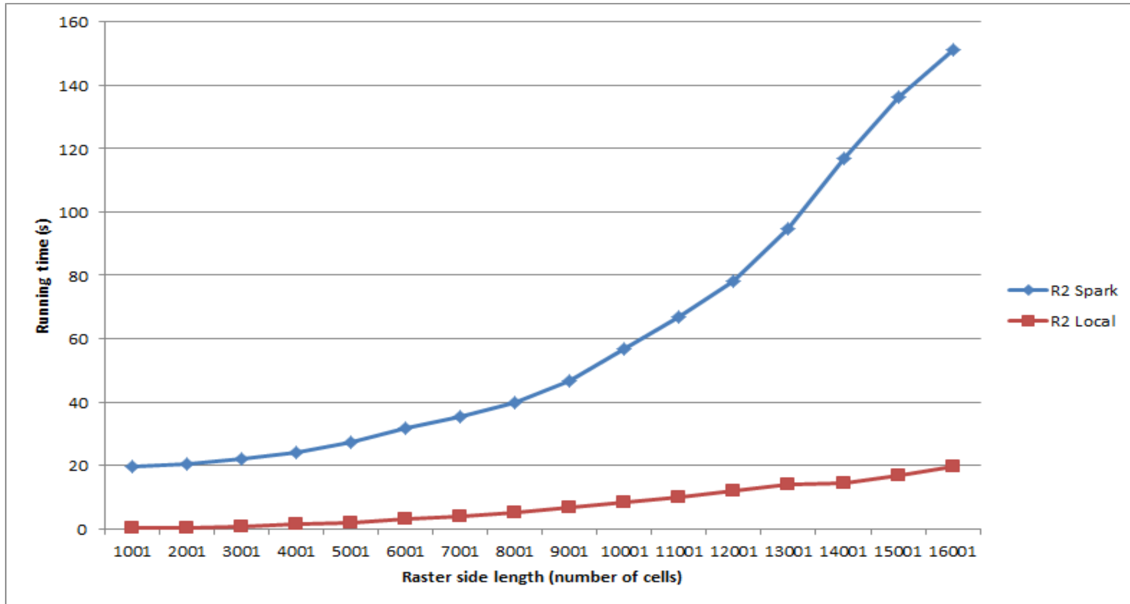


Figure 4.10: Comparison of running times of the R2 algorithm, between a single multi-core CPU and a cluster using Spark.

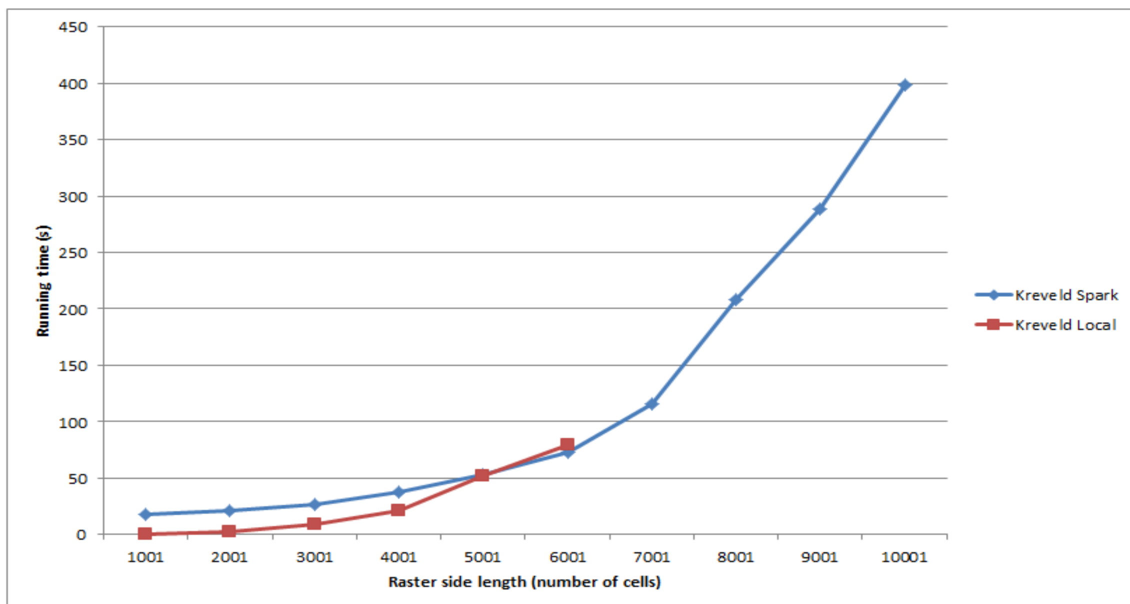


Figure 4.11: Comparison of running times of the van Kreveld algorithm, between a single multi-core CPU and a cluster using Spark.

5
Discussion

Comparing our findings with those from Axell and Fridén [6] shows that it is not probable to achieve better, or even equal, performance when doing viewshed analyses on a cluster compared to on a GPU, at least not when using raster with side length up to 16001 cells. This can be attributed to the fact that the computational power of modern-day GPUs are exceedingly high and the speed of networks has not had the same evolution as GPUs.

Considering that running a viewshed analysis on a GPU is so much faster compared to running the same analysis on multiple CPUs, we think that to utilize the power of a distributed solution one needs to use it for specific, and suitable, use cases. One of these use cases would, for example, be finding a subset of points that has the highest visibility, so called visibility index. First, to gather this result one has to run (exponentially) more computations, as this is equivalent to running viewshed analyses regarding all points in the area as observers. Secondly, as the result only consists of a subset of points there is no need to aggregate the whole result to one single node, thereby correctly utilizing the distributed aspect of this system.

We think that Spark works best when the total number of calculations that the application performs are very large, which is one difference between the R2 and the R3 algorithm. This is highlighted in Figure 4.9 where a comparison between the local and the distributed implementation of the R3 algorithm is shown. As previously mentioned in Chapter 4 there is a difference in hardware when computing the results, but one also has to consider that it is usually both cheaper and easier to rent the computational power of 16 virtual cores than it is to buy the hardware yourself.

5.1 Conclusion

This thesis has examined the feasibility of using a distributed system to perform viewshed analysis. As can be seen in Chapter 4 our implementation of the R2-algorithm achieved high accuracy, but the results of the running-time speak against using distribution for these kinds of computations. The reason for this seems to be

the overhead incurred by sending data over a network, which compared to internal communication, with for example a GPU, is a rather slow medium.

We can also conclude that the idea of using distribution in GIS is a sound one, though one has to carefully select appropriate use cases for a distributed solution, in an effort to properly utilize the power of distribution and to minimize any overhead introduced by the distributed solution, e.g. transmission overhead.

5.2 Future Work

As previously mentioned it is important to examine the use cases when distributing GIS computations. We think that the process of finding a subset of points with the highest reach, i.e. the points where an observer would cover as big an area as possible, would be a suitable use case for distribution. It would be interesting to see the performance one could achieve on a medium sized cluster, while performing these calculations.

According to the results we gathered, performing viewshed analysis on a cloud cluster will not perform better than using a GPU. What would be interesting to examine is to perform analyses on rasters larger than 16001x16001 cells. To be able to do this, one either needs to have more capable hardware in terms of available RAM, or to optimise the algorithms with respect to how much memory they use.

Bibliography

[1] J. A. Ware, D. B. Kidner, and P. J. Rallings, "Parallel distributed viewshed analysis," in *Proceedings of the 6th ACM international symposium on Advances in geographic information systems*. ACM, 1998, pp. 151–156.

[2] B. N. Thai and A. Olasz, "Raster data partitioning for supporting distributed GIS processing," *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 40, no. 3, pp. 543–551, 2015.

[3] J. Abdul, M. Potdar, and P. Chauhan, "Parallel and distributed GIS for processing geo-data: An overview," *International Journal of Computer Applications*, vol. 106, no. 16, pp. 9–16, 2014.

[4] S. Tabik, A. R. Cervilla, E. Zapata, and L. F. Romero, "Efficient data structure and highly scalable algorithm for total-viewshed computation," *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, vol. 8, no. 1, pp. 304–310, 2015.

[5] L. Lu, B. Paulovicks, M. Perrone, and V. Sheinin, "High performance computing of line of sight viewshed," in *Multimedia and Expo (ICME), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1–6.

[6] T. Axell and M. Fridén, "Comparison between GPU and parallel CPU optimizations in viewshed analysis," Master's thesis, Chalmers University of Technology, 2015,
Summary: *The authors make a comparison between running viewshed analysis on a multi-core CPU and a GPU, and concludes that the latter is more efficient. They manage to observe a speed-up of 3.1x on a GPU compared to a CPU. We used the results from this paper to compare our distributed solution to the one run on a GPU.*

[7] S. Mayhew, *A dictionary of geography*. Oxford University Press, USA, 2015.

[8] W. R. Franklin, C. K. Ray, and S. Mehta, "Geometric algorithms for siting of air defense missile batteries," *A Research Project for Battle*, no. 2756, 1994,
Summary: *A paper that describes what a viewshed is, and what algorithms*

are used to calculate a viewshed. The problem they aim to solve is to develop an algorithm for viewshed analysis that has less complexity than $\mathcal{O}(R^3)$. The result of this is the R2 algorithm. We used this paper to deduce how to implement the R3 and the R2 algorithm, and also for general understanding of viewshed analysis.

[9] J. Albrecht, *Key concepts and techniques in GIS.* Sage, 2007, ch. 9.3.

[10] M. van Kreveld, "Variations on sweep algorithms: Efficient computation of extended viewsheds and classifications," in *Proceedings of 7th International Symposium on Spatial Data Handling*, vol. 13, 1996, pp. 13A.15–13A.27,
Summary: *A paper that aims to calculate viewsheds with the use of plane sweep algorithm. The author, Marc van Kreveld, is the source of both the algorithm and the name of the algorithm; the van Kreveld algorithm. We used this paper as a guide when implementing the van Kreveld algorithm.*

[11] C. Ferreira, M. V. Andrade, S. V. Magalhães, W. R. Franklin, and G. C. Pena, "A parallel sweep line algorithm for visibility computation." in *Proceedings of XIV GEOINFO*, 2013, pp. 85–96,
Summary: *The authors developed a parallelized version of the iterative van Kreveld algorithm. This is done by dividing the viewshed area into partitions, and calculate every partition in parallel. Their solution is used in our implementation of the van Kreveld algorithm.*

[12] Apache Spark. Apache Spark home page. (Visited on 2016-05-04). [Online]. Available: http://spark.apache.org/

[13] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation.* USENIX Association, 2012, pp. 2–2.

[14] Intel Corporation. Ark. Intel® Core™ i7-870 Processor. (Visited on 2016-05-09). [Online]. Available: http://ark.intel.com/products/41315/Intel-Core-i7-870-Processor-8M-Cache-2_93-GHz

[15] Google Cloud Platform. (Visited on 2016-05-09). [Online]. Available: https://cloud.google.com/

[16] Machine Types. (Visited on 2016-05-31). [Online]. Available: https://cloud.google.com/compute/docs/machine-types#predefined_machine_types

A

Definitions

GIS - Geospatial Information Systems - Information systems that analyze and present geographic or spatial data.

DEM - Digital Elevation Model - A digital representation of terrain with elevation information.

TIN - Triangulated Irregular Network - A digital data structure that represents a DEM with the use of triangles.

Raster - Used to represent a DEM and is a grid of squares, where every cell in the grid represents a small subarea.

LoS - Line of sight - The process of calculating whether a target point is visible from an observer's point of view.

Viewshed - A representation of an area that shows which points are visible, and which are not.

Apache Spark - An open-source distribution framework maintained by the Apache Software Foundation.

RDD - Resilient Distributed Dataset - Dataset used by Spark.

Carmenta - The company where this thesis was performed, Carmenta specializes in GIS. www.carmenta.com