Universiteit Leiden
Opleiding Informatica

Gathering and Managing Spatiotemporal Data

For Tracking Animal Behaviour

Name: J.H. van Staalduinen

Date: September 5, 2016

1st supervisor: Prof. dr. J.N. Kok

2nd supervisor: Prof. dr. H.J. van den Herik

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)

Leiden University

Niels Bohrweg 1

2333 CA Leiden

The Netherlands

Gathering and Managing Spatiotemporal Data
For Tracking Animal Behaviour

Jan van Staalduinen

Abstract

In this thesis, we investigate spatio-temporal data about the movements of animals. The thesis describes three phases in the process of studying these movements. First, we describe ten methods of gathering the data, including UvA-BiTS, ARGOS, mobile communication networks, LoRa, video feeds, LiDAR, RADAR, Time-of-flight measurement, human observation and vibration sensors. Second, we describe four methods of visualizing the data: moving points on a 2D map, a 3D graph, a Voronoi graph and a heatmap. We test these visualizations with two data sets, one containing real data about Przewalski's horses in Mongolia and one fictional, generated, data set about moving 'agents' on the Oostvaardersplassen. Finally, we propose two methods by which the data can be analyzed: average minimal distance over time and clustering. We use the nature reserve "The Oostvaardersplassen" in The Netherlands as a case to test our methods.

Acknowledgements

First of all, I would like to thank my supervisor Joost Kok for the weekly meetings and advise about the progress of my work. Among all the things he did, they were a lot of help in selecting, from a huge body of compelling research, topics to discuss in this thesis. I also thank Jaap van den Herik, for the very interesting orientating discussions about some fascinating alternative research subjects I could have chosen: it's a shame there was only place for one bachelor thesis.

Furthermore, I thank the people that joined me, each of them working on their own thesis, in livening up the desolate places that constitute study rooms during summer.

The board members of study association "De Leidsche Flesch" were very kind to provide us with an endless supply of thesis-fuel (coffee). Finally, I thank my family for trying to understand what I was working on and supporting me therein.

Contents

Chapter 1

Introduction

***In this chapter**, we will give a short introduction on the field of animal tracking, as well as the topic we will be discussing in this thesis. We describe the methodology we used and how it relates to a more standard methodology for data science. We present our problem statement and research questions and, finally, provide a brief summary of the conclusions of this thesis.*

1.1 Tracking animals

In recent years, the tracking of animals has made great leaps through the growing use of electronic trackers. There are three reasons for this progress: computing power increased, data gathering improved and processing methods enhanced. So, the knowledge on the behaviour of animals across the globe enriched and became more detailed each year.

That does not mean, however, that it is easy master the topic. The wide variety of different tracking methods and techniques make it difficult to select the proper subtopic that is most interesting to investigate. Additionally, as in general in the growing field of data science, the use of large sets of data to address a problem or obtain some insight into a situation might not be an obvious step to take.

During talks with the Dutch Foresting Service (Staatsbosbeheer) about the wetlands called the Oostvaardersplassen (see Figure 1.2), it became clear to us that they were developing plans with regard to the data they gathered. Furthermore, it is not only important to know the applicable methods of gathering data, but also how one could visualize them and make the data useful and/or interesting for both the management and the public. To this end, we wanted to know more about the tracking of animals and began research into the topic.

1.2 Methodology

As described by Van den Herik [vdH16](Dutch)[1], one methodology for doing research in the field of data science consists of 7 steps:

1. Gathering
2. Cleaning
3. Interpreting
4. Analysis
5. Visualising
6. Story telling
7. Emergence of new paradigm

These steps are visualised in Figure 1.1. From the fifth step, visualisations, three alternate routes can be taken: 1) the data gathering has to be changed, 2) from the visualisations, we can interpret the data and 3) by analysing patterns in the visualisations, conclusions can be distilled. In this thesis, we focus on three steps in this methodology: 1) Gathering, 4) Analysis and 5) Visualising.

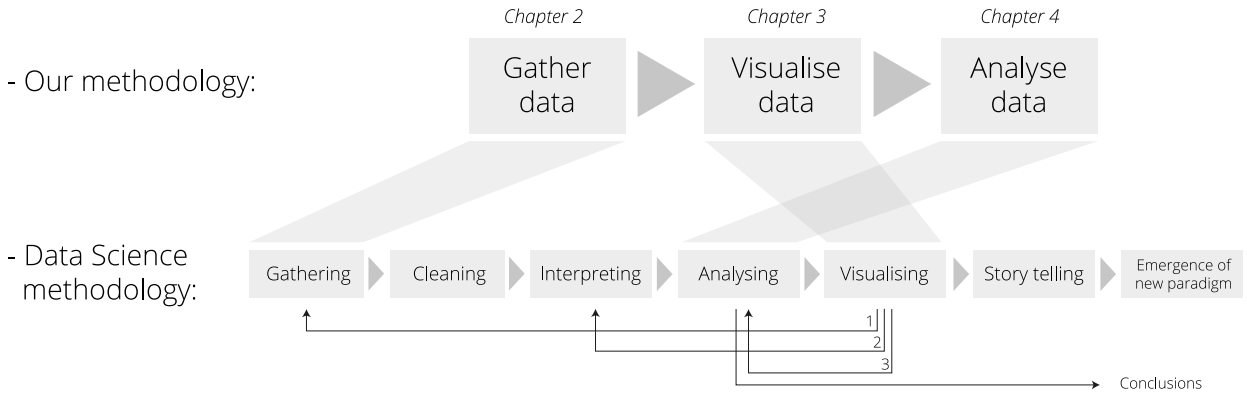


Figure 1.1: Our research methodology.

The research for this thesis was thus divided into three phases. In the first phase, we tried to identify different technologies that can possibly be used to track (larger) animals across a wetland. Then, for each technology we determine how well it performs and how suited it is for our case. In the second phase we looked at some interesting ways in which the data can be visualised to aid the management or spark interest for the wetlands in the public. In the third phase we analysed the data using the power of a computer.

[1]The description of this methodology in the cited essay omits the first step 'Gathering' as it assumes the data set is already present.

This trichotomy is reflected in the chapters. In chapter two, we describe the techniques for tracking animals. We go into some proven, tried-but-true, methods such as human observation and GPS trackers, but also some methods that are less obvious such as a video feed that might be used to produce a stream of GPS data. In chapter three, we introduce two sample data sets, one of which we created ourselves, and use these to test a number of visualisation options. The third phase of our research is reflected in the fourth chapter, where we analysed the data using the computer.

In the fifth chapter, we conclude this thesis by providing summary and conclusions of our work and propose some directions for further research.

1.3 Problem statement

As explained in Section 1.1, we are looking to gain knowledge about the field of animal tracking and the use of spatio-temporal data therein. We express this goal in the following problem statement:

Problem statement: *How can we use spatio-temporal data of animals in wetlands to aid management and interest the public?*

As we described in Section 1.2, the methodology we chose leads to three questions around which we have focused our research to find an answer to the stated problem:

Research question 1: *How can we gather spatio-temporal data of animals in wetlands?*

Research question 2: *Which methods can be used to visualize the spatio-temporal data?*

Research question 3: *Which methods can be used to analyse the spatio-temporal data?*

In our final chapter, Chapter 5, we will combine the knowledge we gained in research for this thesis to answer these questions and thereby distill a conclusion regarding the problem statement. In the next section, Section 1.4, we will already give a summarised version of this chapter.

1.4 Summary of conclusions

In Chapter 2, we find that mobile operations networks, LoRa or the UvA-BiTS system work best for tracking animals and gathering the spatio-temporal data. Using a video feed might be interesting, but this idea needs further development.

In Chapter 3, the testing of data visualisation provided us with interesting graphs. In particular, the Voronoi diagram proved capable of nicely visualising the distances between the horses. In combination with the analyses

of the average distances as produced in Chapter 4, this might show a useful way of continually monitoring flocking behaviour in a group of animals. The heatmaps we generated are useful to monitor another aspect of the nature reserve. They can be used to 1) find much used routes the animals take and 2) reveal areas that are very attractive to the animals.

In Chapter 4, we see that monitoring distances in flocks of animals and finding clusters of animals might be interesting to test on real-life data, but that our fictional data set is too predictable to show interesting behaviour.

Case: The Oostvaardersplassen

Figure 1.2: The Oostvaardersplassen is a nature reserve in the Netherlands, managed by the State Forestry Service. It is located in a polder created in 1968. This means that is a relatively new ecological structure. One of the goals of this reserve is to re-create a "truly wild'' nature as would be found in the Netherlands some two hundred years ago, before the Industrial age. Another is to gain knowledge of self-preserving ecosystems.

Chapter 2

Animal Tracking Technologies

***In this chapter**, we describe the methods available for tracking animals. We make a distinction between methods that use a tracking device that is planted on animals ('direct' tracking) and methods that monitor and animals from afar ('indirect' tracking). We evaluate four direct tracking methods and six indirect tracking methods. In evaluating direct tracking methods, we focus on the technology by which information is sent to the processing station instead of on the tracking device itself. After each evaluation, we give the advantages and drawbacks of that method. Finally, we combine this knowledge to distill a conclusion.*

2.1 Two categories of tracking methods

Technologies for tracking animals can be divided into two main categories: (a) direct tracking, whereby animals are tracked by placing a tracking device on the animal and (a) indirect tracking, whereby animals are tracked from afar (see Figure 2.1).

(a) Direct tracking (b) Indirect tracking

Figure 2.1: A direct tracking system (a) works by placing a device on an animal, while an indirect tracking system (b) uses a device that tracks the animal from afar.

We start by looking at different technologies to transmit gathered data to a base station in Section 2.2. We do not list all possible tracking devices that could be bought and fitted to an animal, because they all basically fulfil the same function: find the geographic location and send it to a base station. As they do differ in the way they send their data to this base station, we highlight four available options.

Intuitively, indirect tracking might be preferred over direct tracking because it causes less of a violation for the animal: the animal does not have to be caught to place a transmitter, we could track it from a distance. Furthermore, using indirect tracking we can probably monitor multiple animals whereas a tracking device placed on a single animal only traces that single animal. We explore six possibilities of indirect tracking in Section 2.3.

Finally, in Section 2.4, we combine and distill the information we found and explained in these sections. We conclude by comparing how suitable the methods are in our case. Included in this section is a table of the advantages and drawbacks of each method. The advantages and drawbacks of each method are also summarised at the end of their sections.

2.2 Direct tracking methods

In this section, we describe tracking methods that use a tracking device placed on animals. We will focus on the technologies available to send the tracking data from the tracker to a receiver and not go into detail about the large body of different commercially available GPS tracking devices. There are quite a number of technologies available for sending the gathered data to a base station or processing unit. We describe four such technologies: A) UvA-BiTS, B) ARGOS, C) LoRa and D) Mobile communications networks.

A) UvA-BiTS

At the University of Amsterdam, a research program is focused on developing and maintaining a system for tracking birds, called UvA-BiTS (University of Amsterdam Bird Tracking System). The research program started with a paper in 2013 by Willem Bouten [BBSBC13]. Since then, the system has been used and is in use to track birds from a number of different species.[1]

The system consists of trackers, base stations and relays (see Figure 2.2). The GPS trackers are sufficiently small to be worn 'as a backpack' by medium-sized birds. They contain a small memory unit, on which spatiotemporal data is stored during periods the bird is out of reach of the base or relay stations. In the next few paragraphs, I will describe the system components in further detail. First, I will describe the sensor that is placed on the animal. Second, I will describe the relay station used to transfer data from the sensor to the base station. Finally, I will describe the base station.

[1] See `http://www.uva-bits.nl/publications/`, consulted July 6th, 2016

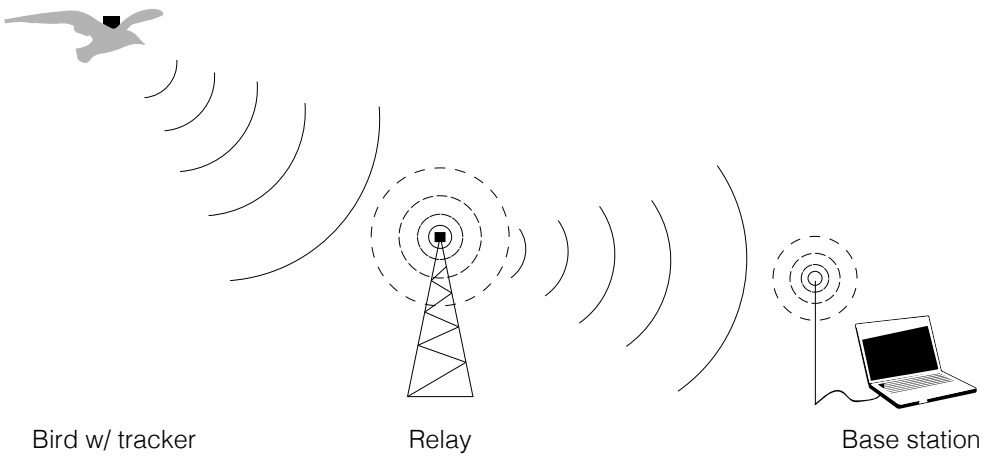


Figure 2.2: A graphical overview of a set-up of the UvA-BiTS field unit using one relay.

Sensor

The sensors come in a variety of sizes, although they roughly share the same subsystem that consists of a battery powered by a solar panel on top, 32MB of flash memory and some sensors. The different models are suited to different classes of birds (see Figure 2.3). See Table 2.1 for a summary of the tracker specifications.

Due to the battery being powered by a solar cell, it will recharge during flight or when situated in sunny conditions. By being able to recharge on-the-go the tracker can operate for very long periods away from the base station. The battery capacity is dependent on the size of the sensor.

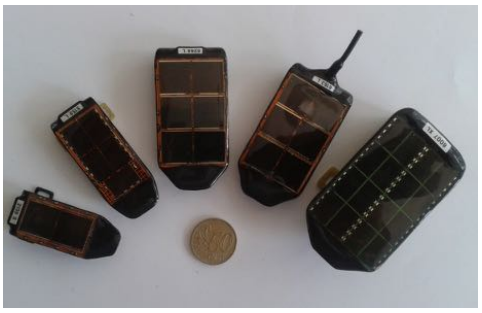
All sensor data is stored on a small 32MB flash memory chip. By doing this, the data can be gathered even when the bird is out of range of a base station. As many birds migrate south for the winter and even considering the very large transmission area that can be created using a base station and relay stations, this is out of reach. Once a bird moves into a transmission zone, its data is transferred to a base station and freed up for re-use. The flash memory is capable of storing 500,000 GPS fixes.

The tracker contains a number of sensors for gathering spatiotemporal data. Apart from the aforementioned GPS sensor, it also contains a temperature sensor, a tri-axial accelerometer and optionally a barometric pressure sensor. The accelerometer measures the acceleration in three dimensions with a frequency of 20 times per second. Noticably, flapping flight can be seen in the data, as this shows up as a periodical change in vertical speed (see Figure 2.4).

Once the bird moves into a reception area, the tracker starts to communicate with base and/or relay stations using its ZigBee[2] radio module. Being a transceiver, it can both send and receive data. The tracker can now both send the gathered spatiotemporal data to the base station and receive new instructions on measurement, e.g. sample frequency. By reducing the measuring frequency, the bird can be tracked longer without visiting a

[2]ZigBee is a wireless communication protocol. See `http://www.zigbee.org/`.

(a)



(b)

Figure 2.3: A falcon wearing a UvA-BiTS tracker (a) and various models of UvA-BiTS trackers compared to a 50 eurocent coin for scale (b). *Both images: UvA-BiTS website (`http://www.uva-bits.nl/`)*

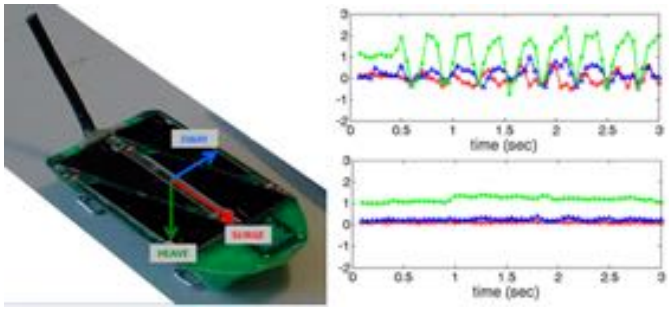


Figure 2.4: Output of the accelerometer in a UvA-BiTS tracker. The first graph shows the acceleration in terms of g during flapping flight. The second graph shows acceleration in terms of g during gliding flight. *Both images: UvA-BiTS website (`http://www.uva-bits.nl/`)*

Size	10.9 cm^2 – 87.8cm^2
Weight	7,5 – 45,0 g
Power source	Solar panel
Battery capacity	65 – 860MaH
Battery type	Lithium polymer battery
Communication	ZigBee two-way radio transceiver
Storage	32MB flash memory
Sensors	GPS, temperature sensor, 3D accelerometer (20Hz), barometric pressure sensor (optional)

Table 2.1: UvA-BiTS sensor specifications

reception area as the memory size is fixed.

Base station

The base station, a portable computer such as a laptop connected to a small antenna, is used to coordinate the gathering of data. The laptop runs software specifically designed for this task, by which received data can be stored in files and propagated through the UvA-BiTS ground station network. It also acts as an instruction interface with the tracker, by providing the user with a graphical means to alter measurement settings and send the renewed instructions back to the tracker.

The laptop connects through the internet (DropBox) with UvA-BiTS' *Virtual Lab*, in which large amounts of data can be stored and processed. The data is then stored in a central database, from which visualizations can be created. One of the options is to export the data to a KML file to be used in Google Earth. The database can also be queried by using Structured Query Language (SQL) or a graphical query builder.

Relay station

The reception area can be enlarged by using one or more relay stations. By using these signal repeaters, reception areas of up to hundreds of square kilometres can be realised. The relay stations are designed to be used extensively in outside conditions, as they are weather-proof and can operate up to three months on one battery pack.

Advantages	*Drawbacks*
Small sensors, well-developed working environment	No continuous tracking information once target leaves reception area

B) ARGOS

ARGOS is a system for retrieving positional data from trackers through the use of satellites [Cla89]. Dating back to 1989, it still in popular use and subject to active development today. The project was originally launched in a collaborative effort between the French Space Agency CNES and the American Space Agency NASA, while also involving the American National Oceanic and Atmospheric Administration (NOAA).

The ARGOS system consists of a "constellation of low Earth orbit (LEO), polar-orbiting satellites, a global network of terrestrial receiving stations, two data processing centers for continuous, round-the-clock operations"[3] (see Figure 2.6.a) and a series of trackers, in many sizes and shapes. Its main advantage is that the network can be used on every location of the Earth.

[3]See `http://www.argos-system.org/web/en/333-why-choose-argos.php`, consulted July 6th, 2016

Instruments

The current ARGOS hardware is the third iteration of the project, aptly named 'ARGOS-3'. The earliest sketches surfaced in 1997 and the system was delivered in 2002, receiving a software update in 2004. The instrument was extensively tested by Astrium and EUMETSAT, after which it has been deployed aboard the Metop-A (launched in 2006) and Metop-B (launched in 2012) satellites (see Figure 2.5). A third Metop satellite under the EUMETSAT Polar System and carrying ARGOS-3 hardware is to be launched in October 2018. Alongside these European satellites, the ARGOS-3 hardware was sent into space aboard the NOAA-19 satellite on February 6th, 2009. According to an official document introducing ARGOS-3, plans exist(ed) for one more flight plan aboard a then yet to be confirmed fifth satellite.[4]. This fifth satellite seems to be the SARAL satellite of the Indian Space Agency ISRO and CNES. SARAL stands for Satellite with ARgos and ALtiKa and was launched on February 23rd, 2013. Together with the two satellites that still carry ARGOS-2 hardware, this brings the total number of active ARGOS satellites to six (see Table 2.2).

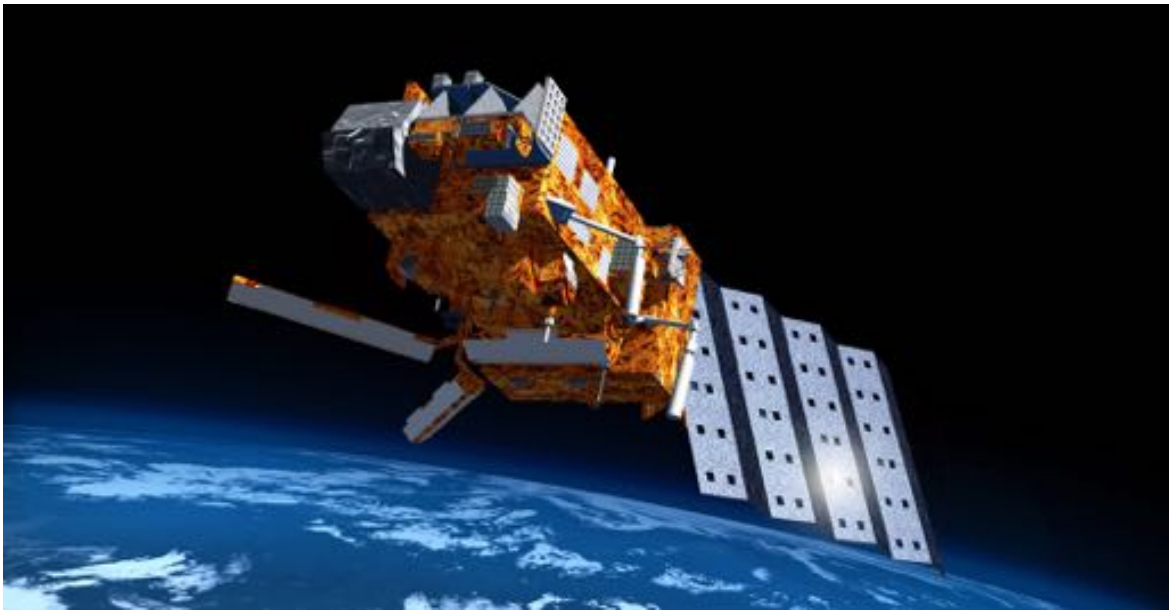


Figure 2.5: An artist's rendition of a Metop satellite, which are used to carry ARGOS hardware. *Picture: European Space Agency*

Satellite	**Launch date**	**Hardware**
SARAL	25 February 2013	Argos-3
METOP-B	17 September 2012	Argos-3
NOAA-19	6 February 2009	Argos-3
METOP-A	19 October 2006	Argos-3
NOAA-18	20 May 2005	Argos-2
NOAA-15	13 May 1998	Argos-2

Table 2.2: The six currently operational satellites that carry ARGOS hardware, in decreasing order of launch date. Note: NOAA-19 was called *NOAA-N'* (NOAA-N Prime) before launch.

[4]Source: `http://www.argos-system.org/files/pmedia/public/r286_9_argos3_metop_en.pdf`, consulted July 23rd, 2016

The data trail

The data gathered by the six satellites is sent through the air to any of the nearly 70 ground receiver stations located around the world. There are three main receiver stations: one in Svalbard (in Spitsbergen, Norway), one in Fairbanks and one in Wallops Island (both in the United States) which together receive all the messages collected by the satellites in their orbit, providing worldwide coverage (see Figure 2.6.b). The satellites send their data to all receiver stations (main or otherwise) within their reach, a circle of 5000km in diameter across the surface of the Earth.

Once a transmitter is within reach of a satellite, it stays within sight of it roughly 10 minutes. The time between two messages depends on the type of the transmitter, but ranges between 90 and 200 seconds. The satellites move across each of the poles approximately 14 times a day. The trajectory of a satellite shifts about 2800km each revolution, measured at the equator. Thus, the trajectories overlap each time. As $14 * 28 * 10^2 \approx 39 * 10^3$km and the Earth's circumference measures 40,075km [Bal09], this corresponds to just under one global sweep per satellite a day.

In 'regional mode', the network of regional stations receives the data from the satellites close to realtime: the satellites try to send their data immediately to a receiver station that is in range. However, not every place on earth is within reach of a regional station, and global coverage is thus not guaranteed. In 'global mode', the data gathered by the satellite in its orbit is sent to one of the main receiver stations once it is in range. This way, global coverage is achieved.

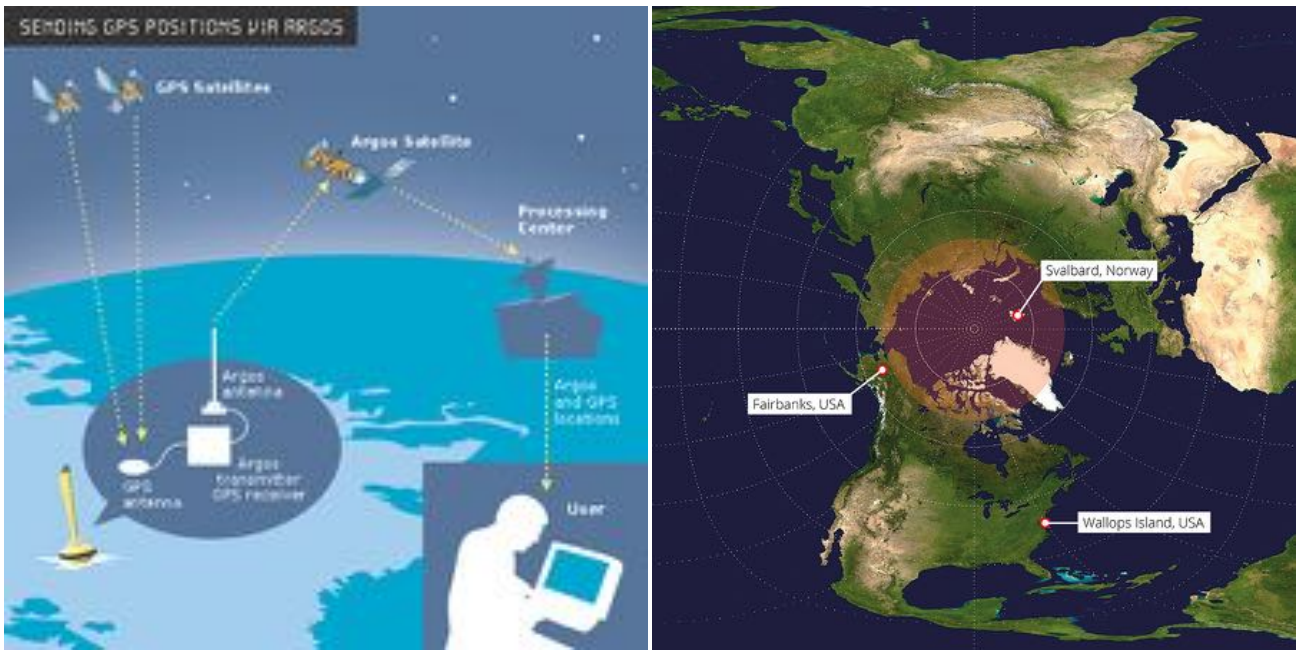
Each of the ground receiver stations sends their data further on to one of the two processing stations; one operated by CLS[5], a subsidiary of CNES, in Toulouse, France and one in Washington, USA, operated by CLS America which is itself a subsidiary of CLS. Once processed, the data can be accessed by the scientists conducting the study.

Determining the position of a tracker

The ARGOS satellites can perform a location analysis based of a transmitter by measuring the Doppler shift on its transmit frequency [VMRF02], making use of the change in distance between the satellites and transmitter during their interaction. When the distance is decreasing, the transmit frequency as received by the satellite is higher than it actually is and would be measured when the distance remains static. By measuring Doppler shift, an accuracy of up to 150 metres can be achieved. When the transmitter also has a GPS sensor, the two position measurements can be combined to achieve an accuracy of a few meters.

Advantages	*Drawbacks*
Small sensors, well-established research program, global	Real-time tracking only within certain areas

[5]CLS: Collecte Localisation Satellites



(a) (b)

Figure 2.6: The ARGOS system makes use of satellites to receive data from transmitters and then sends it via receiver stations to its processing centres (a). *Image: ARGOS website*[3]. In (b) the three main receiver stations are shown on a map (using the Cassini-Soldner projection). For reference, a red circle measuring 5000km in diameter is drawn with the North pole as its origin.

C) LoRa

Several wireless communication technologies are currently being developed for use in Internet of Things networks, network standards designed with the intention of connecting objects to the internet. These technologies include for example LoRaWAN, Bluetooth 4.0 and WiFi HaLow (IEEE 802.11ah) [APN12].

While these technologies (and many of their kind) differ in implementation, they have the same goal: providing a low energy data transmission link over a large range. Minimizing the energy cost is essential for long-term deployment of sensors: they must be able to operate for extended periods of time on a small battery charge. By providing a long range link, networks can easily be deployed on a large scale, such as the city wide coverage in Westland[6] and Amsterdam [Bla15].

LoRa or LoRaWAN is an abbreviation of *Long Range [Wide Area Network]* and is being developed and promoted by the LoRa Alliance[7], a non-profit open association. Several large companies, e.g. HP, IBM, Cisco and ARM[8] are a member of the LoRa Alliance. According to their website, the Alliance is *"collaborating together and sharing experience to drive the success of the LoRa protocol, LoRaWAN, as the open global standard for secure, carrier-grade IoT LPWA connectivity"*.

LoRaWAN makes use of transmitters that can be placed on objects (IoT-objects) and access points. The IoT-

[6]See `http://www.m2mservices.nl/nl/nieuws/m2m-services-rolt-lora-netwerk-uit-voor-het-westland`
[7]See `https://www.lora-alliance.org/`, consulted July 6th, 2016
[8]See `https://www.lora-alliance.org/The-Alliance/Member-List`, consulted July 6th, 2016

objects communicate with radio waves with the access points, while the access points are connected to the 'regular' internet using WiFi or ethernet cables (see Figure 2.7).

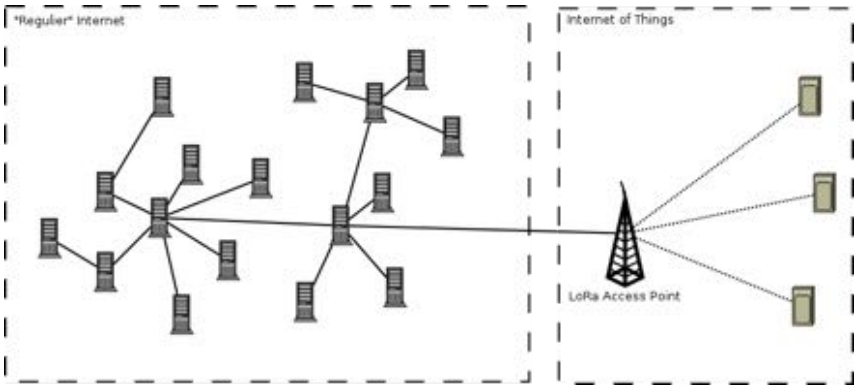


Figure 2.7: Schematic diagram of the connection of a LoRa network (box to the right) to the 'regular' internet (box to the left). Source: [vS16]

For its radio communications, LoRaWAN makes use of a section of the radio spectrum that is reserved for, as stated by the International Telecommunications Union (ITU), "industrial, scientific and medical applications"[9]. In the European Union, LoRaWAN uses the 863MHz–870MHz and 433MHz bands while in the United States the 902MHz–928MHz band is used[10].

In recent research commissioned by the Dutch Ministry of Economic Affairs, it was concluded that due to expected growth in traffic on the 863MHz–870MHz band, an increase in interference problems was to be expected and close monitoring of this growth was and is necessary. A follow-up inquiry has been started by the Dutch Telecommunications Agency in May 2016[11]

In terms of performance, LoRaWAN can provide a data rate for LoRa ranging between 0.3kbps and 22kbps and one GFSK data rate at 100kbps for Europe[12]. The minimum data rate in the United States is different at 0.9kbps to comply with FCC rules. A quick estimate based on the numbers used in [BBSBC13] for calculating the size of GPS records provides us with the data in Table 2.3.

Data rate (kb/s)	**Large record (7.7kb) (Hz)**	**Small record (0.5kb) (Hz)**
0.3	0.04	0.6
0.9	0.1	2
22	3	40
100	10	200

Table 2.3: Estimated possible frequencies of sending GPS data over LoRaWAN, assuming a) that the advertised speeds are in kilo*bits* per second and b) the size of the flash memory described in [BBSBC13] is described in mega*bytes*. *32MB (=* $256 * 10^3$ *kilobits) of flash memory can store up to* $5 * 10^5$ *GPS records, which are 15x larger if all sensor data is included. Thus, a single small record would take up approximately* $5 * 10^{-1}$ *kilobit and a large record would take up approximately 7.7 kilobit. Estimates rounded to 1 significant figure.*

[9] Article 1.15, ITU Radio Regulations

[10] See LoRaWAN Specification (¶7.1–7.4): `https://www.lora-alliance.org/portals/0/specs/LoRaWAN%20Specification%201R0.pdf`, consulted July 6th, 2016

[11] See `https://www.agentschaptelecom.nl/actueel/nieuws/2016/vervolgonderzoek-naar-internet-things` (Dutch), consulted July 6th, 2016

[12] Source: `http://www.semtech.com/wireless-rf/lora/LoRa-FAQs.pdf`, consulted July 6th, 2016. Although not specified in the document, these are probably kilo*bits* and not kilo*bytes*.

The Kerlink Wirnet 868 MHz is an example of a LoRa access point (gateway) that has a range of around 7 miles (11 kilometers)[13]. It can be obtained for around € 1500[14]. With this range, one such gateway would be enough to cover the entire Oostvaardersplassen when mounted at the Visitor's center near the entrance (see Figure 2.8).

Advantages	*Drawbacks*
Relatively cheap and readily available sensors	Low data rate

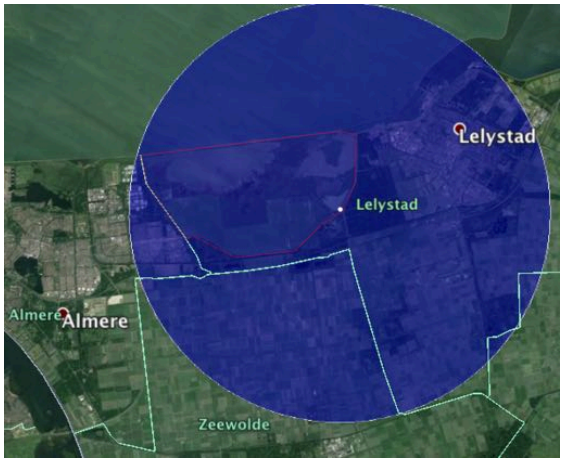


Figure 2.8: Range of 11 miles around the Visitor's center (white dot) of the Oostvaardersplassen (red-lined area)

D) Mobile communication networks

Before the existence of LoRaWAN, many portable devices were already connected to the internet through the use of GSM/GPRS networks. GPRS (sometimes dubbed 2.5G) is an extension of the original GSM protocol (dubbed 2G), allowing for (faster) data transfer. These networks were widely used for mobile phones and were followed up by newer standards such as EGPRS (dubbed 2.75G), UMTS (dubbed 3G) and LTE (said to be a 4G service, but it is under debate whether it is fully compliant with the standard).

Numerous studies have found that next to providing long-distance interpersonal communication channels, these networks can also be used to transfer data from a tracker in the field to a monitoring station. Three examples are a study on moose in Sweden in 2004 [DEE+04], a study on sable antilopes in the Kruger National Park in 2007 [ROS07] and a study on otters Portugal in 2012 [QMdJ+12]. The last study used small trackers of only 84 grams, while the first two used larger collars, the moose collar weighing circa 1.1 kilograms.

Advantages	*Drawbacks*
Relatively cheap and readily available sensors	Low data rate

[13] See `http://www.kerlink.fr/en/products/lora-iot-station-2/lora-iot-station-868-mhz`
[14] See `https://www.thethingsnetwork.org/`

2.3 Indirect tracking methods

We evaluate six indirect tracking methods: A) video feed, B) Radar, C) LiDAR, D) time-of-flight measurement, E) vibration sensors and F) human observation.

A) Video feed

A wetland usually consists of wide open areas with little vegetation obscuring the view. We could therefore conceivably use a camera system to monitor movements across the wetland. In the case of the Oostvaardersplassen, some projects have already used webcams to provide the public with an insight into the life of animals. One example of this is 'Volg de Vos' (Follow the Fox), where in 2011, 2012 and 2014 a webcam was placed in the vicinity of a fox's burrow[15].

Using a video feed might therefore be a viable solution to track animals across the fields, especially since there is little foliage obscuring the view. In early 2015 a mathematical method was designed with which positions (of porpoises) in a video can be translated to GPS locations [JH15]. To test the method, video was captured of the water between the island Texel and the town of Den Helder, i.e. Marsdiep. The cameras were placed on the island, with a view on Den Helder. The GPS coordinates of some clearly visible landmarks in Den Helder were collected, through which a line was drawn towards the GPS coordinates of each of the cameras. By taking the bearing of the porpoise location to this line and the distance of the porpoise to the camera (as seen on the video still) the real GPS coordinates of the porpoise were revealed (see Appendix A for a detailed description of the method). Estimates of the location were quite accurate, citing an error of 12 meters on a distance to the camera of 1 kilometers.

It should be possible (see Fig. 2.9) to expand on this method and use it to calculate the GPS position of animals on a wetland. In this case, a fixed camera would be pointed at the animals from outside of the protected areas.

The detection of key points in the video stills (pinpointing landmarks of which GPS coordinates have been collecting and where this line crosses the shore) are now done manually by clicking and pointing with a mouse. We believe this could in large part be done by the computer. We could use any of several feature extraction techniques, such as the Hough transform [DH72]. Should this prove viable, a system could be implemented that transforms moving objects in a live video feed to spatiotemporal data that can be visualized close to real-time on a map. This might be hard to do for the case observed in [JH15] as porpoises that move through water are hard to distinguish from the (similar coloured and waving) water, but could be easier for animals that move on (top of) a static field of grass.

[15] See `http://www.volgdevos.nl/` (Dutch), consulted July 3rd, 2016

Advantages	*Drawbacks*
Accurate	Small range (up to 1km), problem of obscurement

(a) (b)

Figure 2.9: Comparing a possible set-up on a wetland, the Oostvaardersplassen, (b) with the set-up used by Hoekendijk et al. [JH15] (a) (range in red ≈ 5km, range in green fill ≈ 1km, range in green line ≈ 2km). *Image in (a) from original paper [JH15]*

B) Radar

Radar (RAdio Detection And Ranging) and LiDAR (Light Detection And Ranging) both measure signals bouncing back from objects. Where Radar relies on the backscatter of radio waves, LiDAR uses optical waves to determine the locations of certain objects.

Radar can be targeted air-to-surface (Synthetic Aperture Radar), surface-to-air (e.g. the ROBIN bird detection system[16]), air-to-air and surface-to-surface [WP].

While the use of Radar technology to detect birds is quite established and the capabilities of Radar are clear in that respect, we could hardly find any data about using Radar for ground-based objects. We did find some related papers about the subject, discussing applications in detecting human movement in restricted areas (such as around airports) [BC00] [WP]. However, these studies proved it hard to distinguish a person from a car, let alone distinguish a cow from a horse. Additionally, for these detections it was necessary for the object to move around, so the Radar could detect the object using (Doppler) frequency shifts.

Advantages	*Drawbacks*
Large scanning distance	Expensive, inaccurate

[16]Source: `www.robinradar.com`, consulted July 6th, 2016

C) LiDAR

LiDAR is gaining popularity as a tool for scanning vegetation in ecosystems [DAXX] and height mapping [vdZ13]. Vertical resolution is typically 15 to 20 cm, horizontal resolution is $\frac{1}{3}$ to 1 m[17].

In The Netherlands, the water authority Rijkswaterstaat works together with district water boards ('Waterschappen' or 'Hoogheemraadschappen') to build a height map of the whole country. They use LiDAR technology: measuring devices carried by planes or helicopters that fire laser beams at the ground and measure the time the laser takes to reflect back to the device. In combination with differential GPS and an inertial navigation system, accurate height maps are created of small regions, which are stitched together to make a country-wide data set. After the first project, AHN-1[18], ran from 1997 to 2003 and the second project, AHN-2, ran from 2007 to 2012, the first data of AHN-3 made it to the public on September 10th, 2015. Plans are to expand the data set to have the complete country covered in 2019.

In Figure 2.10, some images can be seen from the AHN project. On these images, the resolution is sufficient to spot individual cars on the street, so it might be possible to use this technique to also spot some larger animals such as horses.

Advantages	*Drawbacks*
Large scale	Expensive, resolution might not be sufficient

(a) (b) (c)

Figure 2.10: LiDAR imagery from the AHN project in The Netherlands. See online viewer: `http://ahn.arcgisonline.nl/ahnviewer/`

D) Time-of-flight measurement

A specific class of LiDAR is the so-called 'Time-of-flight distance measurement' technique (ToF). In 2013, an experiment was conducted using photons to create a 3D distance model of a person standing in front of a sheet of wood, far away from the sensor [MKG+13]. This proved quite successful, as a resolution of one

[17]Source: `http://web.pdx.edu/~jduh/courses/geog493f12/Week04.pdf`, consulted July 6th, 2016
[18]AHN: Algemeen Hoogtebestand Nederland

square centimetre was achieved with a measuring distance of up to one kilometre. The installation used in this experiment is however quite cost-intensive and too slow to be used to track moving objects.

There are commercial solutions available that speed up the process. These cameras are able to capture a large number of frames per second, but only relatively small frames of a few hundred by a few hundred pixels. Furthermore, the distance they can measure is not very large at only a few meters.

Disregarding the rather slow speed of scanning at a long range, there is the problem of occlusion: when two objects stand on the same line of sight, only the object in the foreground can be sensed. This problem might be solved by keeping track of objects over time, although it will be hard to distinguish objects from one another if they move close.

Advantages	*Disadvantages*
Very high resolution	Expensive and slow (quicker solutions do not have a sufficient range)

E) Vibration sensors

Measuring vibrations on the ground can indicate whether animals are close by. By using more than one in conjunction, the direction of a vibration travelling through the ground can be determined. An example of one such sensor is the 'Trespasser', developed by the Dutch company SensingClues[19]. It can be easily deployed, but does not give a detailed picture of the situation around it: it only records the presence of *something* nearby, not what that vibration-causing object is.

Advantages	*Drawbacks*
Easily deployed	Needs power (electricity), doesn't give a detailed picture of the situation

F) Human observation

Traditionally, animals are tracked by observing them in person during field research. As this is not in the scope of this thesis, we will not delve further into this. It is a tried and true practice, but very costly in personnel time. One of the goals of this thesis is to find a way of minimizing this time.

Advantages	*Drawbacks*
Detailed information	Relatively expensive in human time, hard(er) to interface with a database

[19] See `https://sensingclues.com/trespasser/`, consulted July 6th, 2016

2.4 Chapter summary

Section	Method	Advantages	Drawbacks
2.2.A	UvA-BiTS	Small sensors, well-developed working environment	No continuous tracking information once target leaves reception area
2.2.B	ARGOS	Small sensors, well-established research program, global	Real-time tracking only within certain areas
2.2.C	LoRa	Relatively cheap and readily available sensors	Low data rate
2.2.D	Mobile communications networks	Relatively cheap and readily available sensors	Low data rate
2.3.A	Video feed	Accurate	Small range (up to 1km), problem of obscurement
2.3.B	Radar	Easy deployment	Expensive, inaccurate
2.3.C	LiDAR	Large scale	Expensive, resolution *might* be insufficient
2.3.D	Time-of-flight	Very high resolution	Expensive and slow (quicker solutions do not have a sufficient range
2.3.E	Vibration sensing	Easily deployed	Needs power, doesn't give a detailed description of the surroundings
2.3.F	Human observation	Detailed information	Labor intensive, relatively hard to interface with database

Table 2.4: A summary of the methods covered in this chapter.

In this chapter, we described ten techniques for gathering spatio-temporal data about animals. In each section, we included a small table displaying the advantages and drawbacks of the method. In Table 2.4 these are aggregated.

UvA-BiTS is interesting because it consists of a complete system, like ARGOS, but instead of using satellites it uses receptors on the ground or on a tower. The trackers use a memory unit to save data while outside of a reception zone, which would go against the wish to create a real-time monitoring system. However, since the Oostvaardersplassen is an area small enough to be entirely covered, this should pose no problem. The question remains whether the system for processing the data, being designed with this caching of data in mind, is suitable for real-time observation.

The go-to solutions most used seem to be either GPS trackers using **mobile communication networks** or GPS trackers using **ARGOS**. The latter is far better suited to keep track of animals across the world and in situations where it can be hard to physically come close to the animals and seems to be a bit excessive seing as

the area in the case only consists of about 50 km^2 and lies in between two cities (Almere and Lelystad). The solutions based on mobile communication networks do seem viable, especially considering cellphone coverage in the Netherlands is excellent.

Lora(WAN) is equally viable, for its relatively low cost and optimization towards power efficiency. This makes it possible to have trackers that continue to operate for long periods of time without having to recharge, even operating on solar power. It does need a transceiver station to be installed to connect the network, but the periodical costs are less compared to mobile communication networks, where the services and networks of access providers are used. An advantage of Lora is the intrinsic connection to the internet, which makes it easy to set up a system (a server) with a database to process the data.

Using a **video feed** is also a viable method to track animals. The method described in that section to extract GPS coordinates from the video feed could, with some adaptations and more research into the application of computer vision to automate the now tedious labeling of objectcs, be deployed in the Oostvaardersplassen.

We were not able to acquire details about every method. In particular, while the use of **Radar** technology to detect birds is quite established and the capabilities of Radar are clear in that respect, we could hardly find any data about using Radar for ground-based objects. **LiDAR**, using light instead of radio waves, could be employed, but the only solutions currently used are based on scanning from the air and that is impractical for continued observation. **Time-of-flight** measurements seem to suffer from a trade-off between being very fast but at a small distance with a low resolution and being much slower at a larger distance and higher resolution. On top of that, it shares with more methods the problem of obscurement: once one object moves behind another, it is invisible to the observer. This could only partially be solved by keeping a memory of where objects should be.

Two methods we discussed only briefly are vibration sensing and human observation. Using **vibrations** to sense locations seems viable, but it's restrictions make it suboptimal for our use case. It might be possible to lay out a network of vibration sensors, thus providing insight in where the most movement happens across the area. A visualization for this that springs to mind is the heatmap. **Human observation** has been the method of choice for tracking animals for ages and is, thanks to the ease of which it can be done, still widely used. As examples like the Dutch 'National Garden Bird Count' (Nationale Tuinvogeltelling) show, it can even involve the public. Although not impossible, it is undesirable to have people watching the areas every day and keeping notes of the locations of the animals; an automated solution is preferred.

Chapter 3

Data visualisation

***In this chapter**, we use four methods to visualise spatio-temporal data of animals. First, we discuss spatio-temporal data visualisations in general. Then, we implement four visualisation methods and test them using two sample data sets of which one consists of real data about horses in Mongolia and the other consists of fictional data, generated to mimic animal movements on the Oostvaardersplassen. After testing the visualisations with a small data set, we increase the sample size of the fictional data set to assess whether the visualisations hold up when used on larger amounts of moving agents. We describe Cesium, a Javascript library in which interactive visualisations can be plotted onto a map. Finally, we present our conclusions.*

3.1 From graphs to spatio-temporal visualisations

This chapter concerns visualisation. We explain what they are (in the context of computer science), in what ways visualisations can be classified and how we can use them to gain more insight into our data. First, in this section, we will give an introduction into the field of visualisations[1]. In Section 3.2 we will describe the data sets we used to test visualisations. In Section 3.3, we will move on to the visualisations themselves. We began with a traditional animated 2D map (Section 3.3.1). After that, we used Google Earth to plot the data on a map and be able to use the functions of Google Earth to move around the graph (Section 3.3.2). Third, we implemented functionality to plot the data using Voronoi diagrams, which we built into an animation showing the changes over time (Section 3.3.3). Finally, we used heatmaps to display the most travelled locations on the map (Section 3.3.4). In the last section, we use the visualisations on a larger data set to see whether they would hold up when we increased the number of individuals tracked.

First, we should explain what exactly we mean by visualisations. By visualisations, in the scope of this thesis,

[1]This section is for a large part based on the PhD-thesis of Ayush Shrestha [Shr14].

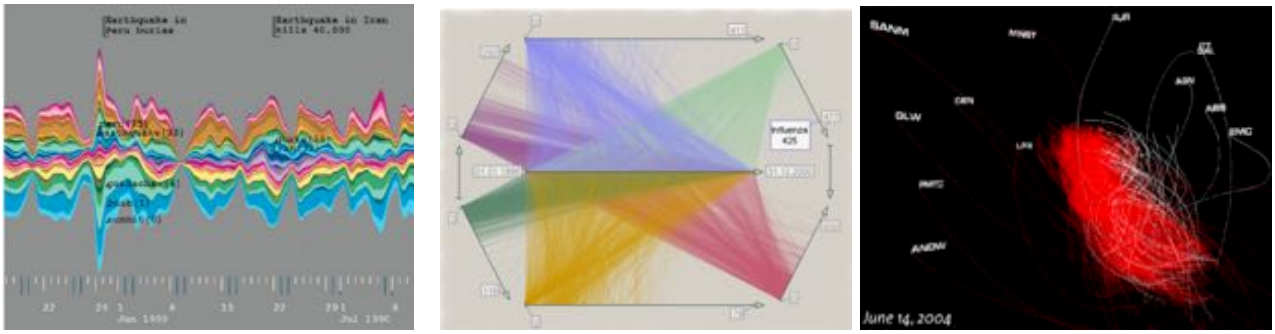
we mean methods of presenting data by the use of a certain algorithm. The use of an algorithm makes it possible to recreate the visualisation at a later time, or with other data. This is what sets visualisations in computer science apart from those in other research areas.

Data visualisations are used every day in office software, where people will encounter them in the form of 2D or 3D graphs. Examples of these graphs include the pie chart, the scatter plot and the histogram. 2D graphs, by definition, show data on a flat, two-dimensional surface (a plane). The inclusion of a third axis in a 3D graph opens up more possibilities, most notably including a third variable to plot when the dimension of the data exceeds two dimensions. However, the dependencies on more than two variables can also be shown in a 2D graph, for example by a change in color, shape or size.

3D visualisations might also be used when the data is geospatial in nature, as the data can be plotted on a map relating to the original location. However, these visualisations suffer from inherent problems such as occlusion, glyph cluttering and overplotting. These might be solved or reduced by adding functionality to zoom, rotate or move the view of the graph. This is an advantage present in software like Google Earth. The Google Earth API, with which extensive animations could be written for the program, has been discontinued. An alternative is the Cesium platform, which uses a web-based Javascript engine to render (3D) maps[2].

Time can be inspected by means of one of two primitives, as described by Aigner in [AMM+07]. The first is by looking at the values in the data at one point in time, while the second is by looking not at one point in time, but rather at a certain time interval[3].

To visualise the change in a variable in *time*, numerous inventive approaches have been proposed. Among them are ThemeRiver (changes in document collections over time) [HHWN02], TimeWheel (axes-based approach for multivariate data) [TAS04] and flocking boids (stock market visualisation based on simulation and animation of flocking behaviour) [Moe04], as noted by Aigner [AMM+07] (see Figure 3.1). On the website of his book, a collection of visualisation techniques for time-oriented data is maintained[4].



(a) ThemeRiver [HHWN02] (b) TimeWheel [TAS04] (c) Flocking boids [Moe04]

Figure 3.1: Three examples of time-based visualizations. *Adapted from their respective papers.*

[2]See `https://cesiumjs.org/`, consulted August 4th, 2016
[3]One could argue, though, that the first is an instance of that latter where the interval is just very (or rather *infinitely*) small.
[4]See `http://survey.timeviz.net/`, consulted August 4th, 2016

Man has long been busy finding ways to visualize *space*, as evidenced by maps found as early as 2300 B.C., in the form of Babylonian clay tablets. Over the years, cartography has continued to improve. The advent of satellites which can make accurate photographs of the earth has proved a huge step. Since the 1960s, people have worked on creating Geographic Information Systems (GISs), which are data structures or standards for describing geographic objects such as houses, bridges and rivers.

Spatio-temporal data binds these two (time and space) together. It consists of data describing objects in space over a certain time interval. Shrestha [Shr14] referenced three main types of visualizations for spatio-temporal data: 1) *multi-views*, which monitor change over time by use of multiple graphs next to each other, each displaying one situation at a certain time; 2) *animations*, which show the change over time by showing the situations after each other through which the viewer can scroll by manipulating the time value and 3) *isosurfaces*, which shows longitude and latitude along axes X and Y and time in the Z axis (a space-time cube).

We have chosen to implement four graphing methods: 1) regular 2D moving points, 2) isosurfaces (3D Time-graph), 3) Voronoi diagrams and 4) heatmaps. We used two sample sets to test these methods: one real data set concerning Przewalski's horses and one data set that we generated to fit the Oostvaardersplassen.

3.2 Sample data

To test and evaluate different approaches to visualizing spatiotemporal data concerning moving animals, we needed some sample data. Due to the time scope of this thesis, we were unable to gather data of animals moving around the Oostvaardersplassen, so we went looking for data that comes close to it and is in nature, scope and taxonomical sense optimally equivalent.

Movebank

The Max Planck Institute for Ornithology hosts a freely online accessible database: Movebank [WK14]. It is accessible through the website `www.movebank.org`. By registering an account, one can access the rich collection of tracking data uploaded and archived by researchers around the world. The website features a search functionality, the 'Tracking Data Map', by which the collection can be filtered to find studies of certain species (see Figure 3.2). We used it to find the study featured below about Przewalski's horses in Mongolia.

Sample data set 1 - Przewalski's horses in Mongolia

The first data set used concerned data about the reintegration of Przewalski's horses in Mongolia. During the 18th century, these horses were found on steppes ranging from the eastern regions of Kazakhstan to the

Figure 3.2: Movebank's Tracking Data Map, with which data of tracking studies can be browsed and, in some cases, downloaded.

Mongolia and northern China. However, in the 20th century, the number of horses rapidly decreased. The last sighting of Przewalski's horses in the wild is said to have been around the late 1960s [Col10]. The species were kept in existence through continued breeding in captivity. In 1997, a mission was started to reintroduce the horses into the wild, in a protected area in the Gobi Desert.

During the years following the reintroduction, the horses were tracked across this area, the Gobi B Special Protected Area. This data set is the one we use here. A total number of 10 horses were tracked over the years of 2001 through to 2007, although not all horses were tracked at the same time. The trackers record a location every 12 hours.

For detailed information about this data set we refer to Appendix E.

Sample data set 2 - Artificial agents roaming the Oostvaardersplassen

The data set about Przewalski's horses is useful for longer term analysis, but not entirely suitable to test visualization methods for visualizing high frequency or realtime spatiotemporal data. As real data about the focus area is out of reach, we created a Python script that produces artificial agents ('horses') moving around on a map of the Oostvaardersplassen.

The agents move in a semi-random fashion, travelling in a certain direction until they encounter a body of water. Once that happens, they turn around and try a different (random) direction until one is found that satisfies an evalution function. This is done by using the evaluation function for each step of each agent, where

it checks each of the next n steps forward for evidence of water[5]. The existence of water on a certain position is determined by loading a black-and-white bitmap image of the map of the area, where water bodies are black and land is white. The speed of the moving agents is determined by the following formula:

$$\Delta s_{pixels} = \frac{v}{\gamma} \cdot \Delta t \tag{3.1}$$

Where Δs_{pixels} denotes the distance to be moved in pixels, v the speed in meters per second, γ the meters per pixel ratio of the used map image and Δt the size of the timestep that is used. A smaller timestep improves fluidity of the animation, but increases computing load.

In this simple version, we do not take into account any other factors such as attraction to feeding grounds and flocking behaviour. Especially the last factor might be interesting to implement to accurately model animal populations that move in herds.

The Python scripts creates a file for each of the horses, that contains a continually updated location. These locations can be visualized on a map. To test whether the script creates convincingly moving agents on the map of the Oostvaardersplassen, the locational data of the artificial agents was first visualized simply as named points on top of the map original that was flattened and used to find the bodies of water that the horses avoid (see Figure 3.3).

Figure 3.3: Location of sample agents on the map. The white dotted circle denotes the area where the agents were spawned (circle added after taking screenshot).

[5]It checks *each* of the steps, to avoid 'jumping' over a body of water smaller than n resolution points.

3.3 Visualizations

In this section, we will use the data sets described in the Section 3.2 to test four visualization methods. We will implement a 2D map, a 3D graph on a map, a Voronoi diagram and a heatmap.

3.3.1 Moving points on a 2D map

One way of visualizing location data is by showing its position on a map. This way, the viewer can see the surroundings of the positional data, and view the data in its context.

Using the Python package `Basemap` [H+07], we can use the capabilities of the Matplotlib library to plot data on any section of the world. It even provides some standard world maps to be used as a background, including NASA's Blue Marble image and NOAA's ETOPO1 Global Relief Model[6].

For our use however, these standard background maps do not have a sufficient resolution: once zoomed in to the Oostvaardersplassen, the readability of the map drops drastically. We therefore use the Basemap package's functionality to connect to the servers of ESRI ArcGIS to download a high resolution map image of the area we need:

```
# Initialize a Basemap object, defining the corners of our map
map = Basemap(llcrnrlon=5.244927,llcrnrlat=52.383565,urcrnrlon=5.445427,urcrnrlat=52.511627,
epsg=5520)
# Download the map image from http://server.arcgisonline.com/
map.arcgisimage(service='ESRI_Imagery_World_2D', xpixels = 1500, verbose = True)
```

This does take a few seconds, depending speed of the local internet connection.

We use the second data set described in the section above: the artificial agents. We create a data set of the latitude and longitude of 23 agents during 500 timesteps. Plotting this data set in an animated fashion resulted in the frames displayed in Figure 3.4. As with the videos displaying the animated Voronoi diagrams described in the corresponding section below, we uploaded the video of this animation to YouTube[7].

An improvement on this would be to show the previous positions of the agents as a line, drawn from its origin along each data point to its current position. To avoid a cluttered image, this line could fade out after a certain amount of data points, so only the most recent data points remain visible. We were as of yet unable to produce this with this package and we are not aware of whether this is possible. We did however manage to create this effect in Cesium (see Section 3.5).

[6] See `http://matplotlib.org/basemap/`
[7] `https://youtu.be/VdRip2DY_ak` (15 frames per second)

(a) $t = 4$ (b) $t = 496$

Figure 3.4: The AI agents roaming the Oostvaardersplassen (blue dots), at $t = 4$ (a) and $t = 496$ (b). Plotted using Python's Matplotlib and Basemap packages.

3.3.2 3D Timegraph

In this section, we will implement the 3D Timegraph and use it to visualize each of the two data sets mentioned in Section 3.2.

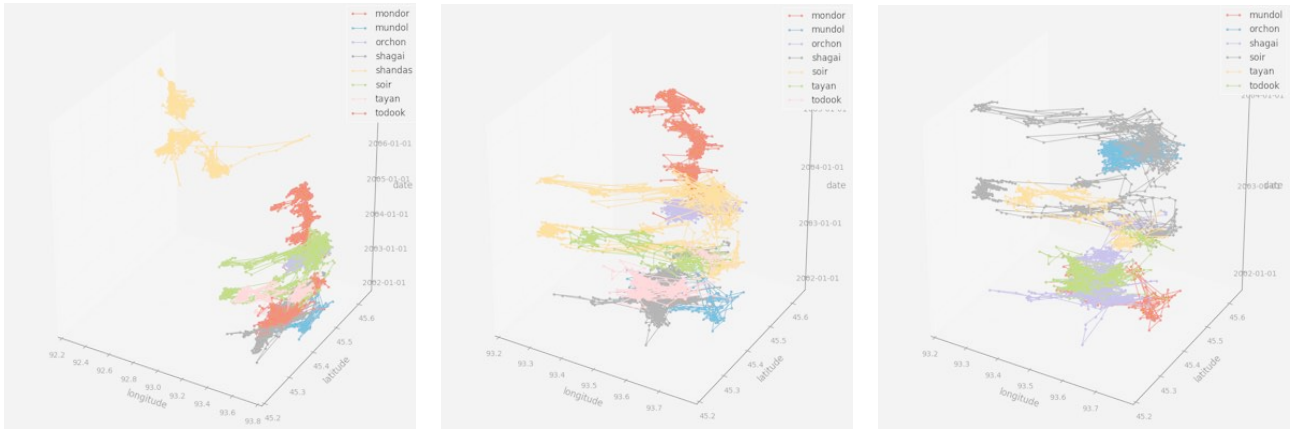
Data set: 1 - Przewalski's horses in Mongolia

We started with creating a 3D graph using Matplotlib (see Appendix C) which shows on the X and Y axis the location as latitude and longitude and on the Y axis the time. This already cleared some things up about the data set, namely the distribution of the trackers over time. In particular, one horse was tagged at a later time when most of the other horses were no longer tracked. Removing this horse from the graph made it possible to zoom in to the details of the other horses (see Figure 3.5).

These graphs are static. It means, as illustrated above, that looking for interesting details entails rerendering the graph. It would be better if we could freely zoom into and look at the graph from different angles.

One of the possibilities is loading a KML-file (see Box 2) into Google Earth. Therefore, we wrote a script that converted the data points of the data set, which were provided in Keyhole Markup Language (KML, see Box 3.1), to lines. The longitude and latitude coordinates were kept the same, but a height attribute was added that depends on the time stamp accompanying the location.

After loading the resulting KML-files into Google Earth Pro, we saw the points were successfully attributed with a height (see Figure 3.6.a). However, it was quite hard to connect the location of a 'floating' line with the location on the ground, so we opted to turn on the 'extrude' option available in the KML language. This draws a plane from the line to the ground (see Figure 3.6.b).



(a) (b) (c)

Figure 3.5: A 3D graph produced with the Python package Matplotlib (see Appendix C) of the horses in the study about Przewalski's horses in Mongolia [WK14]. In (a), all of the horses are shown, but very visibly one horse was followed in a later time frame than the rest of the horses. This horse (codenamed 'shandas') was removed in the second plot (b). In the third plot (c), the horse codenamed 'mondor' was also removed.

We did notice that although we made the extruding planes semitransparent by lowering the alpha-value in its styling definition, the program only shows the underlying terrain, not other lines that run behind it.

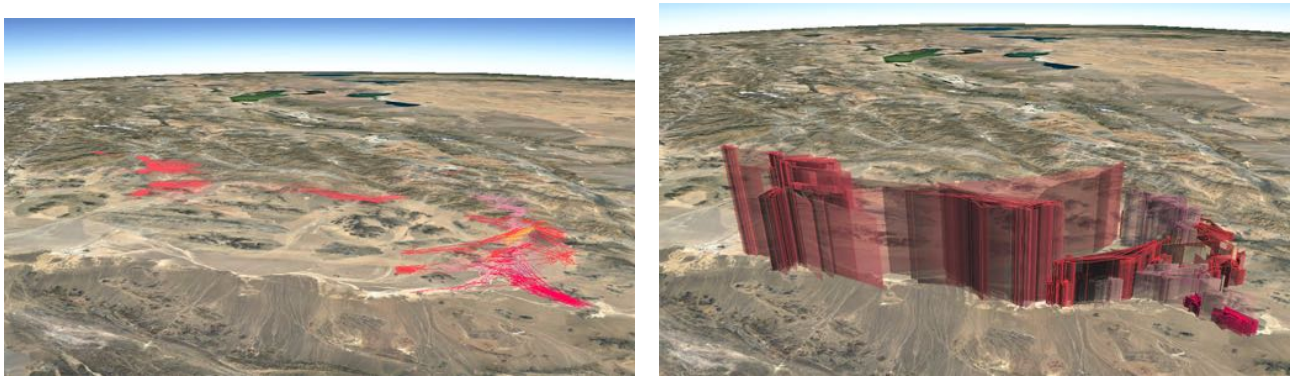
The KML file format

This format, its name an abbreviation of 'Keyhole Markup Language', is an XML-based data container format. Just as XML, it describes objects with certain properties. These objects might have objects embedded in them. In the case of KML, the objects describe geographical data entities such as buildings, borders or shapes and their geographical location. Geographical objects are of the type `Placemark`.

A basic XML file containing one Placemark:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Placemark>
    <name>Simple placemark</name>
    <description>Attached to the ground. Intelligently places itself
       at the height of the underlying terrain.</description>
    <Point>
      <coordinates>-122.0822035425683,37.42228990140251,0</coordinates>
    </Point>
  </Placemark>
</kml>
```

Box 3.1: The KML file format



(a) (b)

Figure 3.6: A 3D graph produced with the converter script in Appendix D, which converts KML point sets into KML lines without (a) or with (b) the option 'extrude' turned on (which extends the line to the ground). *Viewed in Google Earth Pro, obtainable at* `https://www.google.nl/earth/download/gep/agree.html`

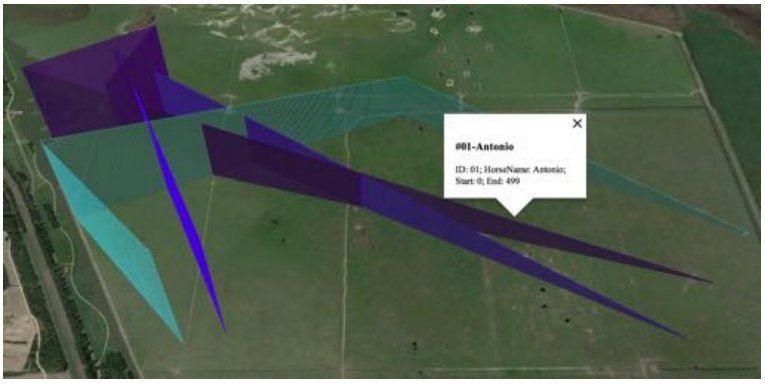


Figure 3.7: Each KML file can be given a name (displayed bold) and description, which are displayed when the line is clicked on.

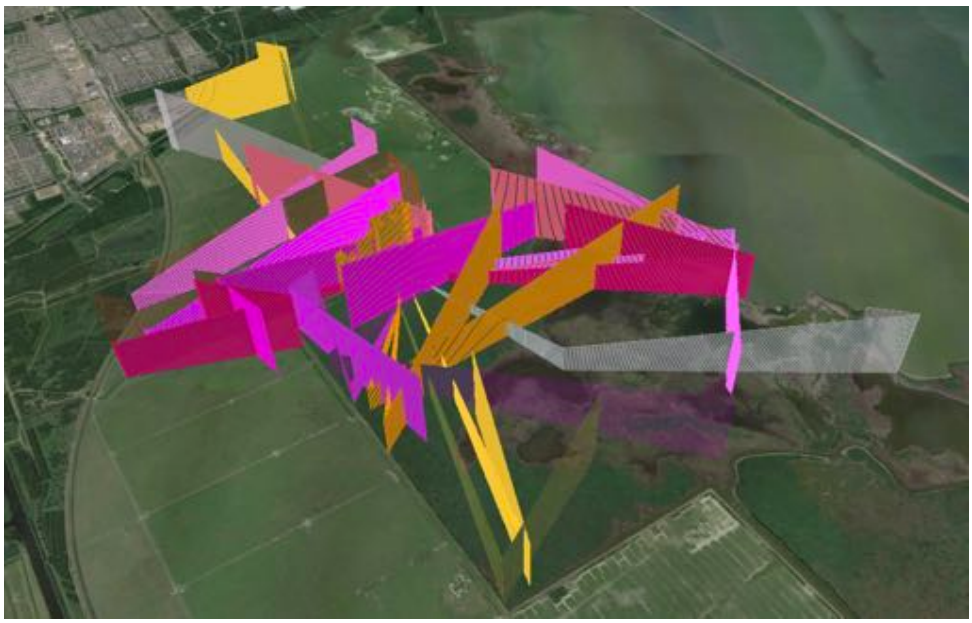
Data set: 2 - Artificial agents

After generating and converting a data set of 500 steps for 23 horses (simultaneous steps; for each step a location is recorded for every horse), we again opened the KML files with Google Earth.

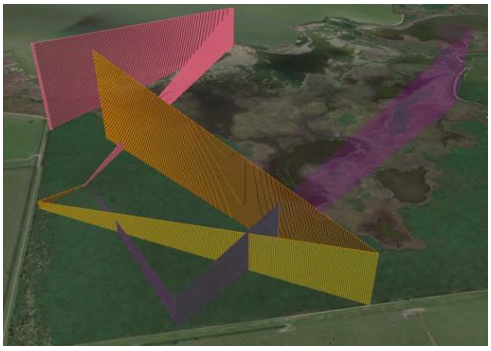
This turned out to make for quite a chaotic image (see Figure 3.8.a). However, we could zoom in and look at each of the lines in detail. Google Earth also allows for toggling the visibility of each of the lines, so individual 'horses', or a small number of them, can be inspected more easily (see Figure 3.8.b and .c). It would be interesting to see whether inspecting the myriad of tracks is easier when viewed on a (much) larger screen[8].

One additional useful built-in feature of KML files is that they have fields available for storing a name and a description of the data. The information becomes visible as a tag when one clicks on the line in Google Earth (see Figure 3.7).

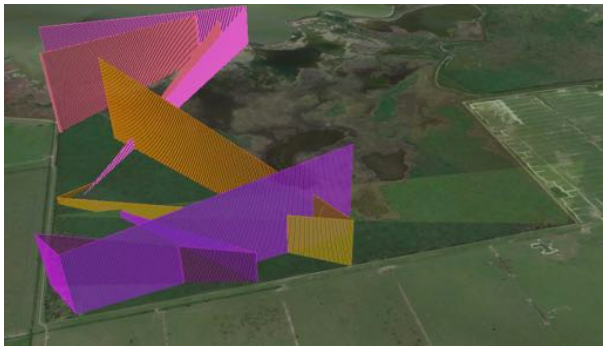
[8]Current device: 12-inch MacBook Pro (2015)



(a) $n = 23$



(b) $n = 3$



(c) $n = 5$

Figure 3.8: A 3D graph of the movement of the artificial agents, first plotting all of them (a) ($n = 23$), then only a few (b) ($n = 3$) and then a few more (c) ($n = 5$)

3.3.3 Voronoi overlay

A Voronoi partitioning, named after the Russian mathematician Georgy Voronoi, is a method of dividing an n-dimensional space into subspaces. In this section, we will look at building a 2-dimensional Voronoi diagram based on the position of our moving objects.

To divide a (2-dimensional) plane into Voronoi regions, called "cells", we draw a cell around each data point. The size of the cell depends on the distance of the data point to other data points. The border of a cell is defined as the collection of points where the distance to the cell's data point is equal to the distance to the nearest other data point. The Voronoi diagram is then the tuple (collection) of all Voronoi cells. It can be visualised by drawing the cells in a 2D graph (see Figure 3.9). The cells can be drawn with background colours or border colours to improve readability.

In mathematical terms: a Voronoi cell C containing data point P_i is defined as the set of all points in the

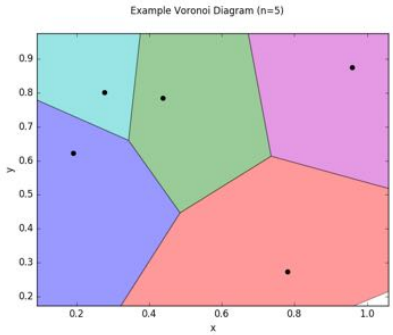


Figure 3.9: Example of a Voronoi Diagram, with 5 randomly generated data points.

2-dimensional space X for which the distance to P_i is smaller than or equal to its distance to any of the other data points in the set of data points P:

$$C_{P_i} = \{x \in X \mid \forall p \in P : d(x, P_i) \leq d(x, p)\} \tag{3.2}$$

In Python, Voronoi diagrams can be created using the *Voronoi* module of the *spatial* package of *scipy*.

We used the script for creating a data set of artificial 'horse' agents moving about the Oostvaardersplassen, described in the previous section, to create a data set of 23 moving agents that move for 500 steps.

With this data set, we ran our new Matplotlib script enriched with the Voronoi module for each of the 500 steps, resulting in 500 Voronoi diagrams. To each diagram, colour was added to differentiate the cells. The locations of the horses were added as blue dots. To make the graph more recognisable, we scaled the original 'water location image' used to intelligently move the agents and set it as the background to the plot. By creating a sequence of all plots (steps 1 to 500 denoted as $t = 0$ to $t = 499$), we were able to play it as an animation. Some of these frames are pictured in Figure 3.10, the last frame is displayed enlarged in 3.11.

We uploaded the animations to YouTube: the first version, only drawing a partial Voronoi diagram[9], the second version, incorporating colors[10] and the final version, the diagram displayed over a map (the map is scaled so the points move over their original coordinates)[11].

[9] https://youtu.be/jL-e4e1Eyo8
[10] https://youtu.be/SCGiOvUuoC8
[11] https://youtu.be/jL-e4e1Eyo8

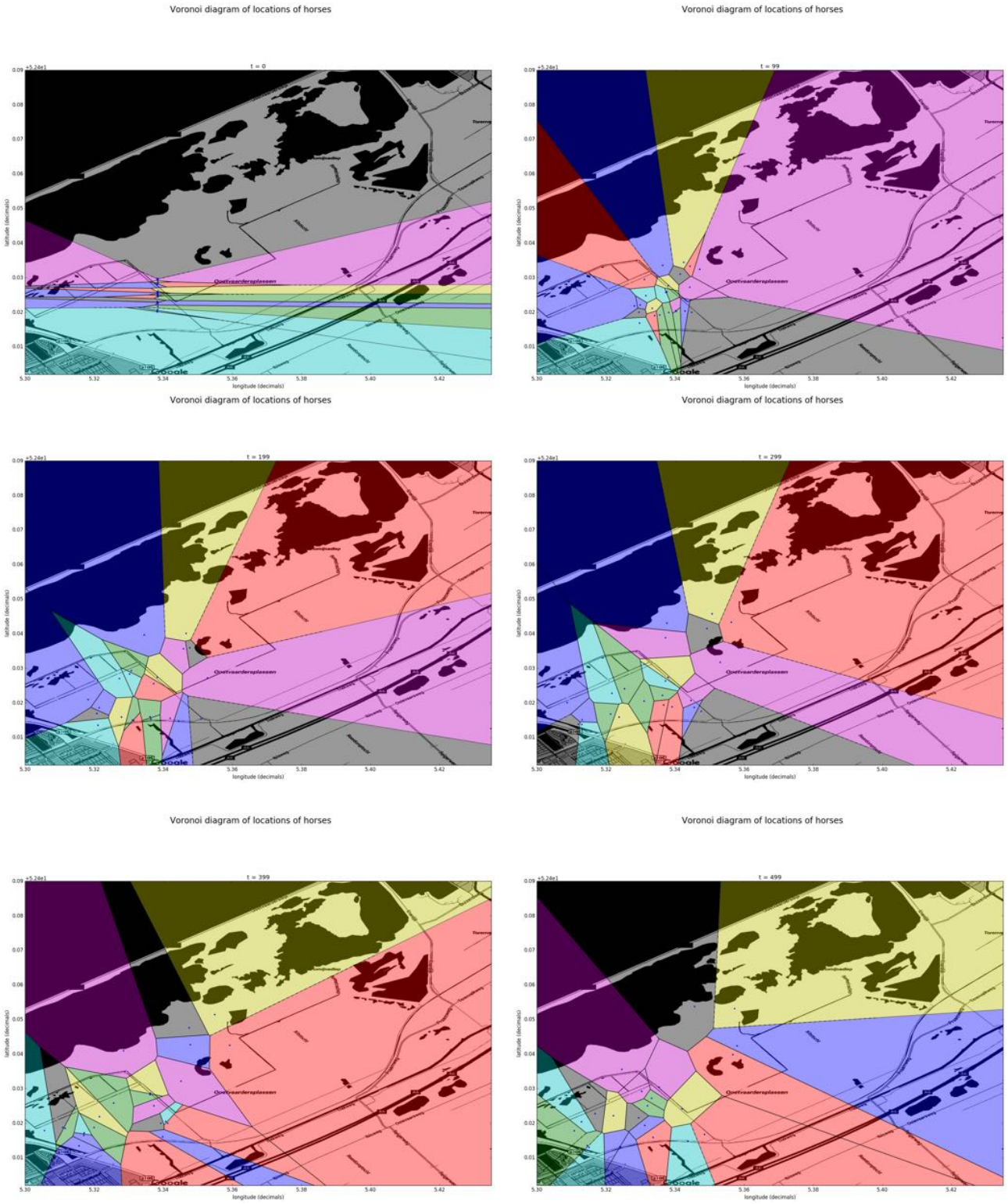


Figure 3.10: Voronoi diagrams: various steps in the process.

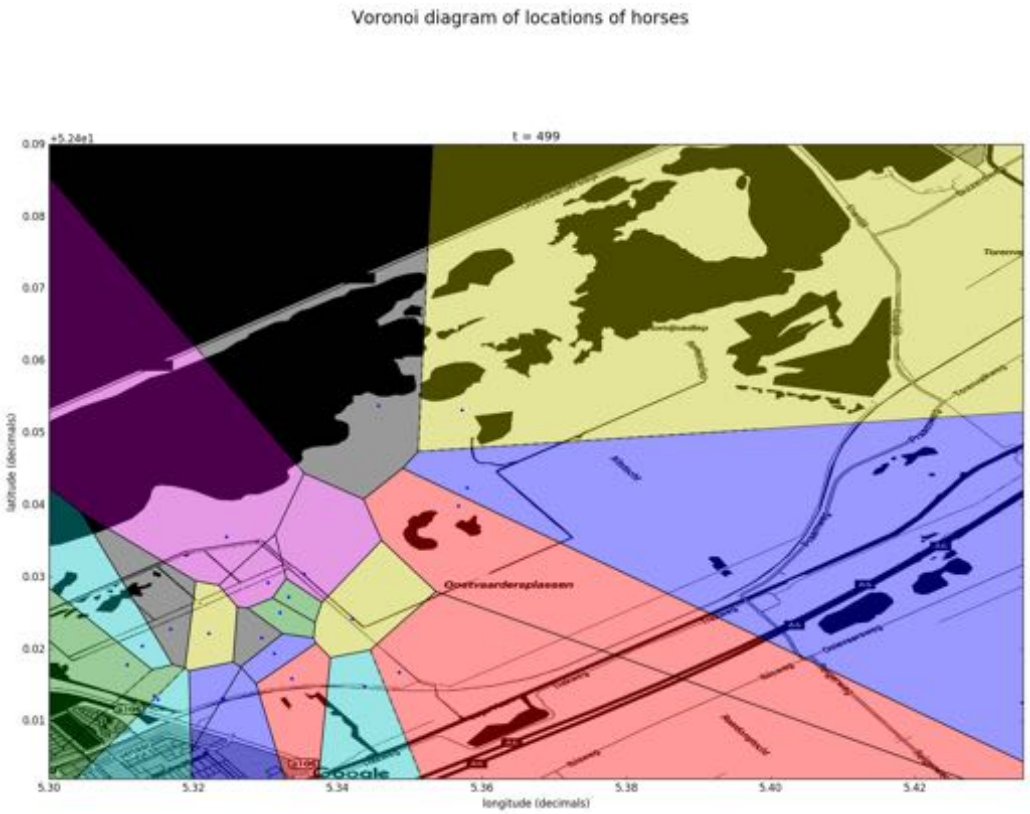


Figure 3.11: Voronoi diagram, final step.

3.3.4 Heatmap

A heatmap is a method of displaying the dispersion of data points. In this section, we will look into the possibility of using the data set we also used in the previous section (Voronoi), following 23 'horses' during 500 time steps.

Heatmaps originate from temperature imaging, where the temperature of certain parts of an object are denoted by a colour in a colour range. By using the same principle not for the temperature but for the density of data points, we can provide insight into a (large) data at a glance.

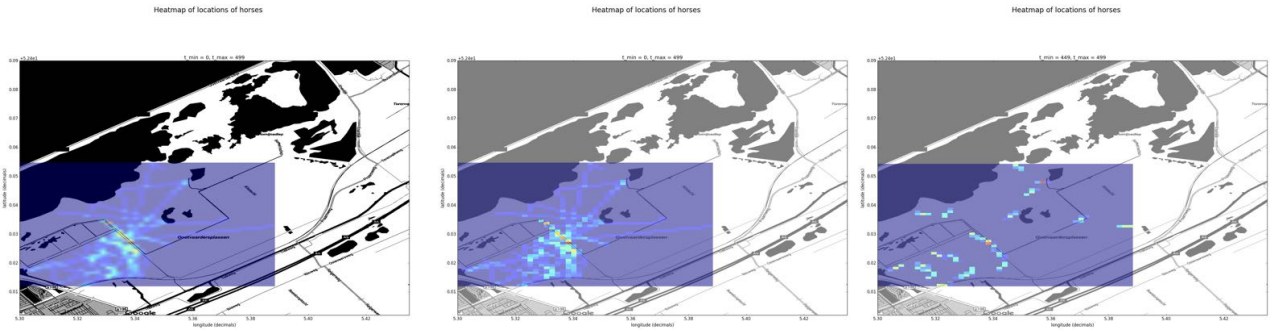
Heatmaps can be built in two or more dimensions, but they are based on the same principle. The graph is divided into subspaces (in 2D: small planes, in 3D: in cubes) of a certain size. Then, each subspace is given a value according to how many data points in the data set would be plotted into that space. Usually, this value is then visualized using a colour range. This range could for example be a gradient of blue to red, where the more red a cell is, the more data points are associated with it.

The heatmap(s) we produced used two dimensions and a colour to represent data density. By using two dimensions, we could overlay the heatmap on the map of the Oostvaardersplassen, just as we did with the Voronoi diagrams, so the data can be viewed and inspected in context of the surroundings. We wanted to see the change in the heatmap over time, so we plotted the heatmap at each time frame in the step range. We then

created an animation of these heatmaps. After this, we used the same process to create an animation of the heatmap, but for each map considering only the last 50 steps. This made the heatmap more flexible and closer resembling the most recent state of affairs. We then inspected the animations to see how much they differed.

We used the `histogram2d()` function out of the `numpy` package for Python to generate the heatmaps and `matplotlib` to plot them over the map image. First, we rendered two animations building up to a heatmap of the full data set, one with interpolation set to "nearest", resulting in sharp-edged boxes[12], and one with interpolation set to "None", resulting in a blurred but smoother view[13]. Second and last, we rendered two animations, one smooth and one sharp, of the heatmap as it is built stepping through the data set with a window of only the last 50 steps.

Once we created these plots, it was clearly visible that once we narrowed the scope of the heatmap to only the last 50 steps, the image became more like a plot of the trails of the agents. In essence, this is of course what happens: a trail of the last 50 positions is drawn on the map. This can be useful to visualise the direction the individual horses are travelling. Furthermore, due to the nature of the heatmap, locations where trajectories intersect turn up with a brighter (more red) colour due to the location being mentioned twice in the data set: once for each agent.



(a) Smoothed, full scope (b) Unsmoothed, full scope (c) Unsmoothed, scope = last 50 steps

Figure 3.12: A comparison of three approaches to plotting the heatmap. First, a smoothed view of the heatmap's 'boxes' (a). Second, the same view with the smoothing turned off (b). Third, plotting with smoothing turned off and only capturing the last 50 steps in the data set (c). This results in more clearly visible trail-like lines. *Note: in (b) and (c), the opacity of the background image is turned down a bit.*

3.4 Plotting a larger data set

Using the plotting methods with the data sets as described in the previous section proved reasonably useful for visualizing the position of objects on the map. We increased the size of the fictional data and rerendered the 3D graph, Voronoi diagram and heatmaps to see whether they held up when using a larger amount of agents.

In the upcoming figures, we used the same script as before (see Appendix B), generating data of n moving

[12] See `https://youtu.be/-itUQNJmxVE` (without smoothing)
[13] See `https://youtu.be/s04TvZJW8j8` (with smoothing)

agents on the Oostvaardersplassen. This time, instead of generating data for $n = 23$, we generated two data sets: one with $n = 1024$ and one with $n = 4096$. The amount of steps for both data sets was 1000, so $t_{min} = 0$ and $t_{max} = 499$. When generated, the first data set amounted to 32.5MB and the second data set amounted to 129.8MB. The times it took to plot each frame can be found in Table 3.1. In the following three paragraphs we describe our findings regarding the effectiveness of the graphs with the new data set(s).

3D graph The 3D graph becomes very cluttered under the increased data set. This makes it almost impossible to draw conclusions from the graph, even considering the amount of agents in this image was limited to $n = 512$, instead of the much larger numbers in the other graphs. One upside is that it is still possible to toggle visibility of individual paths, but this is quite the hassle. This view could be improved by taking a smarter approach to plotting the paths, as is described in the next section about Cesium.

Voronoi We see that while the heatmap is improved by the larger data set, the Voronoi graph becomes quite cluttered by all the small cells. This is especially true for the field in the bottom left. We viewed the same plot on a video wall to see whether this problem can (at least partially) be solved by inspecting it on a much larger screen. This was the case: on this large screen it was much easier to follow and watch the small cells. For pictures of this, see Appendix H.

Heatmap The heatmap copes well under the increased number of moving agents. It even seems to be the case that the heatmap profits from a larger data set. We see a clear distinction between the field to the bottom left and the field more to the top right, where less activity occurs. This is exactly what the heatmap is meant to show.

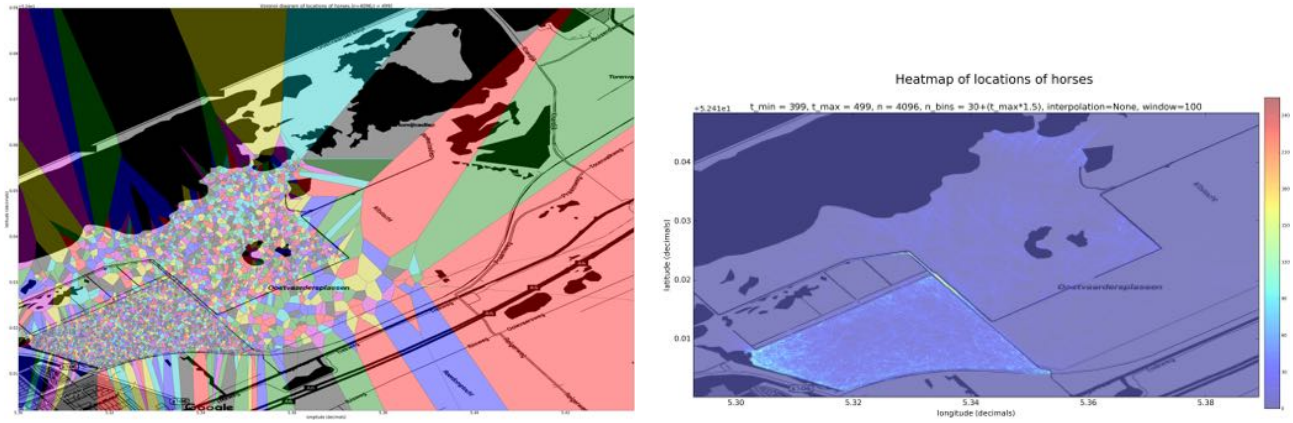
	$n = 23$	$n = 1024$	$n = 4096$
Voronoi	11s	14s	31s
Heatmap	1s	3s	7s

Table 3.1: Indication of time spent plotting one frame ($t = 499$). Timed using the function `time` in Bash. Voronoi script: `fastervoronoi.py` (see Appendix F). CPU: Intel(R) Core(TM) i5-5257U @ 2.70GHz.

3.5 Cesium

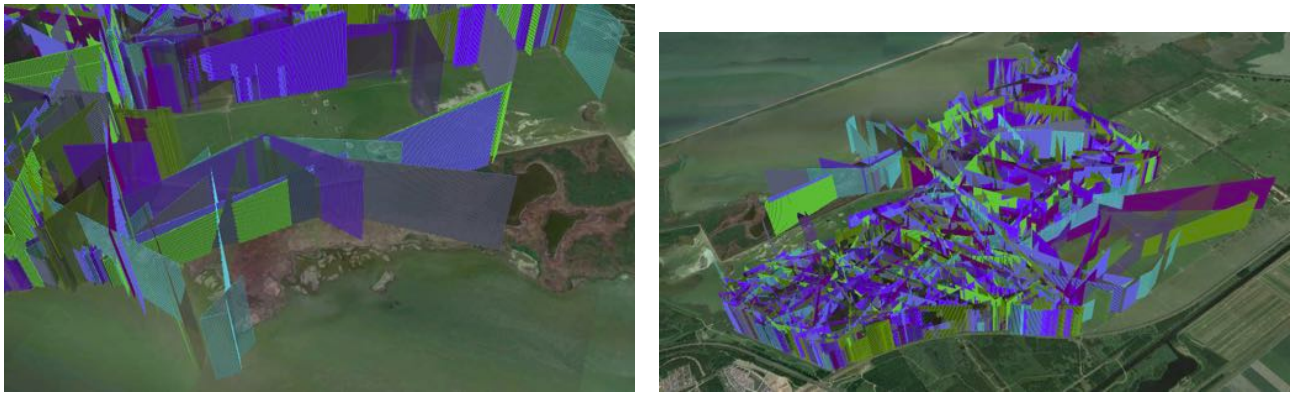
When using Google Earth, we liked the ability to move around and zoom, but the plotting of the data was quite static. There used to be an API by which we could write programs to run in Google Earth, the Google Earth API, but it is deprecated as of December 12th, 2014[14] and will be shut down by the end of 2016. An alternative for this is Cesium: an open-source Javascript library for developing Google Earth-like visualizations based on WebGL. It can project shapes and data on globes and maps. The project is based at `https://cesiumjs.org/`.

[14]Source: `https://developers.google.com/earth/`, consulted August 9th, 2016



(a) Voronoi diagram (b) Heatmap

Figure 3.13: The data set with $n = 4096, t_{max} = 499$ plotted as a Voronoi diagram (a) ($t = 499$), a heatmap (b) ($t_{min} = 399, t_{max} = 499$) and as seen in Google Earth (c) ($t_{min} = 0, t_{max} = 499$).



(a) (b)

Figure 3.14: The data set with $n = 512, t_{max} = 499$ plotted as a 3D graph, viewed as a KML file in Google Earth (Pro).

We installed the Cesium by first installing Node.js, then downloading the latest version of Cesium as a zipfile from `https://cesiumjs.org/downloads.html`, unzipping it and running `npm install` in a terminal in the unzipped folder. Finally, we started a local web server by running `node server.js`, which starts the server with the settings defined in the file 'server.js' which comes with the Cesium zipfile.

Starting a Cesium viewer is done by creating a HTML-document which includes a Cesium widget and opening it in a browser (in our case by browsing to `localhost:8080/[file]`). This widget is expanded to fill the browser window and optionally the entire screen. It can be manipulated with Javascript code. The user can use the mouse to view different parts of the Earth, just as in Google Earth.

First, we wanted to get as far as we got with Google Earth: importing KML files and showing them in the viewer, using Cesium's `KmlDataSource` class [15]. This was successful, as can be seen in Figure 3.15. The cluttering that appears in Google Earth is still an issue that we tried to solve by animating the lines.

The second step was thus to move beyond what we could accomplish in Google Earth: change the behaviour of the data lines as they are drawn in the viewer. In particular, we wanted to be able to animate the lines beyond the capabilities of standard KML files. This was accomplished by not loading the data as a KML file, but loading the original CSV data file containing rows of GPS locations and drawing each 'step' between locations as a polygon onto the screen. As Cesium supports toggling visibility of an object based on the currently viewed date, we implemented this (the code is inluded in Appendix G).

The standard Cesium user interface includes a time slider with which the user can scroll at various speeds through time (see Figure 3.16). The dates in the objects are in Julian Day Number (JDN) format: the integer number of days since January 1, 4713 BC, proleptic Julian calendar[16]. We placed the date of testing, August 9th 2016, in the code[17] to set this as the starting date for the fictional data we plotted. By scrolling through the time line, the new data points appear on the map. By default, the time moves in real-time at one second per second. Time can also be paused or reversed.

The visuals created in Cesium can be seen in Figure 3.17.

[15] See `https://cesiumjs.org/Cesium/Build/Documentation/KmlDataSource.html`, consulted August 9, 2016
[16] Starting to count at noon, Greenwich Mean Time.
[17] August 9th 2016 (00:00) corresponds to 2457610.5 in JDN.

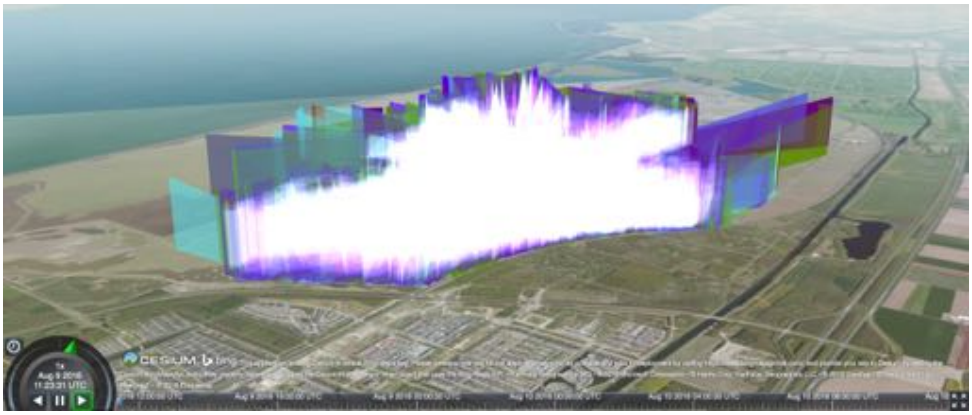


Figure 3.15: KML imported as KmlDataSource in Cesium. Note the white glow, probably caused by the partial transparency of the line stretches that is handled peculiarly by Cesium (i.e. the colours of the semi-transparent layers are added to form a white colour, instead of the semi-transparent layers only letting part of the 'light' pass through as would be natural).

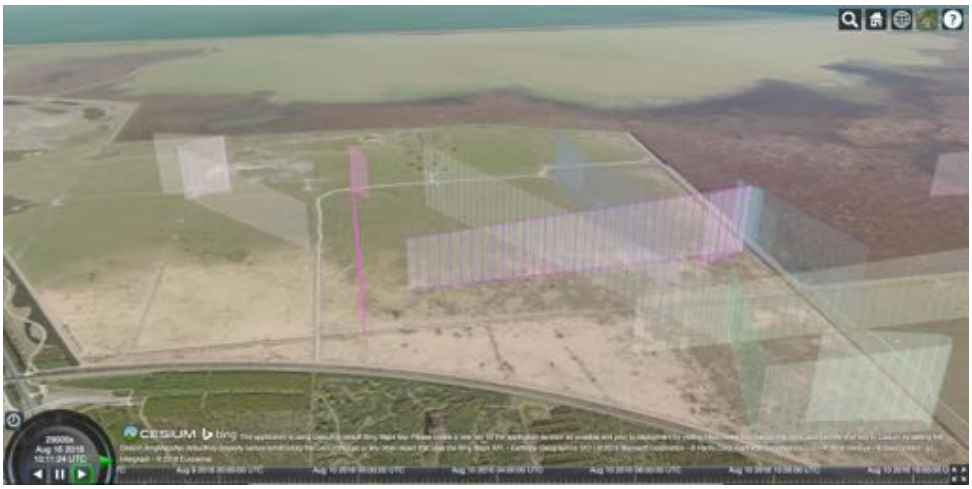
Figure 3.16: A detailed look at the Cesium time controller. Note the green arrow that designates the speed at which time moves (it can be dragged around the circle to increase or decrease the speed of time) and the blue marker at the start of the time line denoting the current time frame. The date and time of the current time frame are also displayed in the centre of the circle. The button at the top left, displaying a clock icon, resets the time frame to the current, real, time.

3.6 Chapter summary

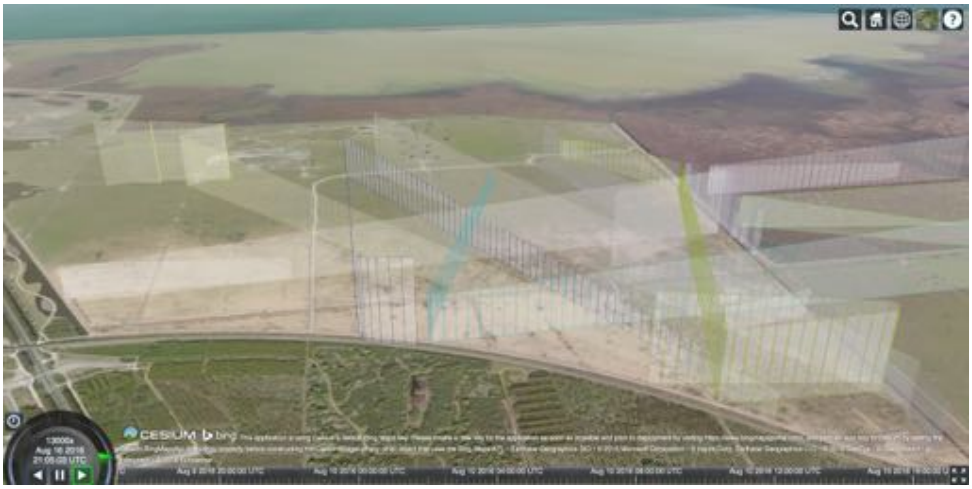
In this chapter, we reviewed four ways of visualizing spatio-temporal data. We began with the 'traditional' 2-dimensional map (1) onto which data points were placed. This was then expanded to a 3-dimensional graph (2) showing the previous locations of the animals. It proved more useful when the map of the area was projected behind the data, so context was created for the viewer. We then experimented with Voronoi plots and heatmaps, while keeping the map image as a background. The Voronoi plots (3) could be used to visualize distances between animals, and differentiating groups using the human ability for pattern recognition. When dealing with larger groups, the Voronoi plot became quite cluttered, but this could partially be solved by viewing it on a larger screen (see Appendix H). The heatmap (4) proved useful for determining attracting locations for animals to visit, and improved in usability when plotting the larger data set. It was however interesting to see that by plotting the small dataset filtered by only showing data within a time window, the heatmap turned into a path visualization. Places where paths crossed were automatically highlighted.

A distinction can be made between two visualization approaches in temporal sense: 1) visualizing real-time data and 2) visualizing collected data after the fact. Voronoi diagrams and 'points on a 2D map' can without any modification be used for both. For heatmaps and 3D graphs, one must make a decision about the time window

(a) $n = 25, ret = \infty, \Delta h_{\Delta t=1} = 0.5$



(b) $n = 10, ret = 5, \Delta h_{\Delta t=1} = 2$



(c) $n = 10, ret = 2, \Delta h_{\Delta t=1} = 2$

Figure 3.17: 3D Graphs in Cesium. The first image displays a KML file loaded into Cesium (a). The second image displays 25 agents moving without fading (b). The third (c) and fourth (d) images show agents moving with fading, where the retention time *ret* determines how long the data points stay on screen. The value of $\Delta h_{\Delta t=1}$ describes the height (in meters) each data point rises relative to the previous data point in that series.

the visualization method is to display. The heatmap must continually adapt the colours on the map to avoid raising every pixel to its highest intensity over time, while 3D graphs (as discussed here) would eventually rise to enormous heights as time progresses. A solution for the latter would be to let the lowest timestamp in the current timeframe be equal to a zero or minimum height and re-evaluate this in each frame.

It was interesting to use Cesium to graph the data as a 3D graph on a map. We would like to see this expanded upon: 1) improving interaction with the data, e.g. by toggling visibility of data points and 2) by loading data into the program in real-time.

Chapter 4

Data analysis

***In this chapter**, we will apply two methods for analyzing the data we generated in the previous chapter. For the first method, we use the distance between the animals as a means for finding unusual behaviour. The distance we use is the distance to each animal's closest neighbour. We start by inspecting the distance per animal, then we establish the average minimal distance in the population over time. For the second method, we use the k-means clustering algorithm to try to find subgroups of animals in the population. In Section 3.6, we summarize this chapter and present our conclusions.*

4.1 Obtaining knowledge from data

In this section, we want to use the data to improve our knowledge of the environment we are modelling. Visualization helps in this regard, by using the human's quite extraordinary capability for recognizing patterns in images [Mat14]. However, some patterns are hard to extract from images alone. In this section, we use the computer's *forte*: crunching numbers. We try out a few techniques to analyse the data generated by our horse-moving script.

First, in Section 4.2, we look at the distances between the horses. Due to the nature of the script, placing the horses near each other at $t = 0$ and then letting them wander until they encounter water, we expected that the distances would grow over time and reach an equilibrium once the average distances could not grow any further.

Second, as described in Section 4.3, we tried to distinguish different clusters (groups) of horses. We used the k-means clustering algorithm. This algorithm partitions the population into k clusters, which can then be either visualized or further processed. We tried out a number of values for k and describe which seemed to be the most accurate.

4.2 Distance over time

To analyse the distance between horses over time, we chose to look at, in a certain timeframe t, the nearest neighbour of each of the data points on the map.

Since the number of points generated is not extremely large, we applied a 'naive' search approach: for each point, we calculate the distance to each of the other points and save this distance in a proximity matrix D. In this matrix, the value of $D[i][j]$ refers to the distance between points i and j. We do not need to fill the entire matrix; the value of $D[i][j]$ is by nature the same as the value of $D[j][i]$. Furthermore, the value of $D[i][i]$, the distance of a point to itself, is of no use.

In applying it to the case, we are more interested in the relative changes[1] in the distance of the horses and not in the *actual* distance. Therefore, we chose to use the Manhattan distance metric. This means that instead of calculating the shortest path ('as the crow flies') between two points, which would be calculated by using Pythagoras's theorem, we simply add the difference over one axis (e.g., 'x') to the difference over the other (perpendicular) axis (e.g., 'y'). Addition is easier and faster to calculate than squaring and taking the square root of the differences in the two axes.

If we wish to calculate the actual distance between horses (perhaps for other analyses), we would have to use a conversion function for the distances between horses, as the positions are in longitudinal and latitudinal coordinates. To convert the latitudinal difference to meters, we can use a relatively simple multiplication with the amount of meters that constitutes one degree of latitude[2]. However, the longitudinal factor varies very much from pole to pole, as one degree of longitude is the largest in meters at the equator, and zero at the North or South pole. The amount of meters per degree of longitude therefore depends on the latitude. To find this value, we can use the following formula:

$$\Delta^1_{longitude} = \frac{\pi}{180} \cdot a \cdot cos(\phi) \tag{4.1}$$

Where a is the radius of the Earth and ϕ the latitude of our position. The latitude ranges from 0° at the equator to 90° at the poles.

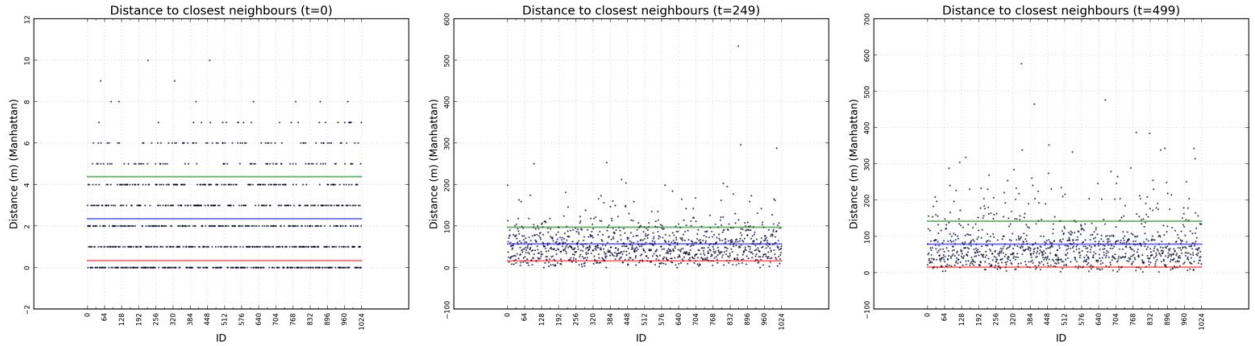
In the end, we did use the conversion function, even though we calculated the distance using the Manhattan metric, to give at least a rough idea of how large the distances would be on the field.

For each timeframe, we calculated for each horse the distance to its nearest neighbour. The minimal distance was then averaged with the minimal distances of the remaining horses in that timeframe to produce the average minimal distance and the standard deviation to that average. The results of this are shown in Figure 4.1.

[1] That is, once we know the value at a certain point in time, we want to compare it with another point in time to see whether change has occurred.

[2] This value does vary slightly due to the Earth not being a perfect sphere, but we're only analyzing a small area.

Here, we see that at the start, the distances are small and do not vary very much. This is true to the way the generating script worked: the horses spawn in a region close to each other. In the next frames, we see that the distances increase, with a few being much larger than the rest, but most of them residing in the range of 0 to 150 'Manhattan' meters.



(a) $t = 000$
$avg = 0.02$
$stdev = 2.36$

(b) $t = 249$
$avg = 40.56$
$stdev = 56.92$

(c) $t = 449$
$avg = 63.10$
$stdev = 78.80$

Figure 4.1: The minimal distances of all the horses at $t = 0$ (a), $t = 249$ (b) and $t = 499$. Also included are the average value (avg, blue line) and standard deviation ($stdev$). The green lines show $avg + stdev$, the red lines show $avg - stdev$.

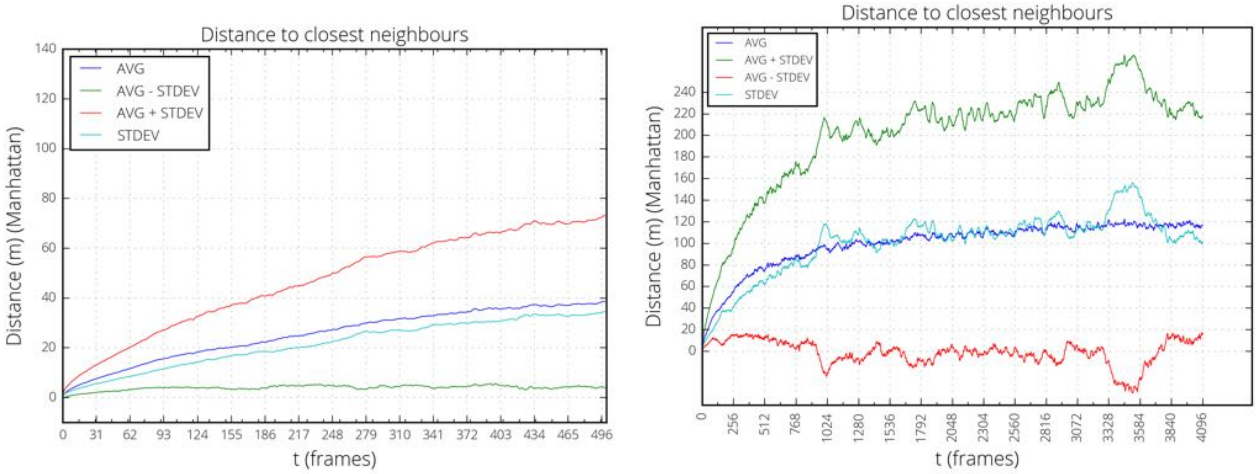
We then wondered to what extend this spreading behaviour had its influence over time. As shown in Figure 4.1, we noticed that the difference between $t = 0$ and $t = 249$ is much larger than the difference between $t = 249$ and $t = 499$. This would make sense, as the horses cannot always move further from each other; the space they can move in is limited by the size of the frame (boundaries of the field). To find out how the average minimal distance behaved over time, we calculated the value for a larger amount of frames, resulting in 4.2.a. It looked like the average was indeed flattening out, so we did the same with another data set that had fewer horses, but more frames. This resulted in 4.2.b, which seems to show the average minimal distance leveling off at around 100 to 120 meters.

4.3 Clustering

A large number of animal species display flocking behaviour and move in (social sub-)groups. By dividing the population into clusters, we can get insight into the different subgroups that might exist in the population. We used the k-means clustering algorithm. We start this section by briefly describing this algorithm, after which we show and discuss the results.

The k-means algorithm works by starting with an initialisation step, after which two steps are repeated until the result is satisfactory. The result is a certain number (k) of groups, in which the data points are classified. Each group has a center point, the average value of the data points it contains.

The first step of the algorithm is in some way (randomly or with a certain strategy) placing the k center points.



(a) $t_{max} = 500$
$n = 4096$

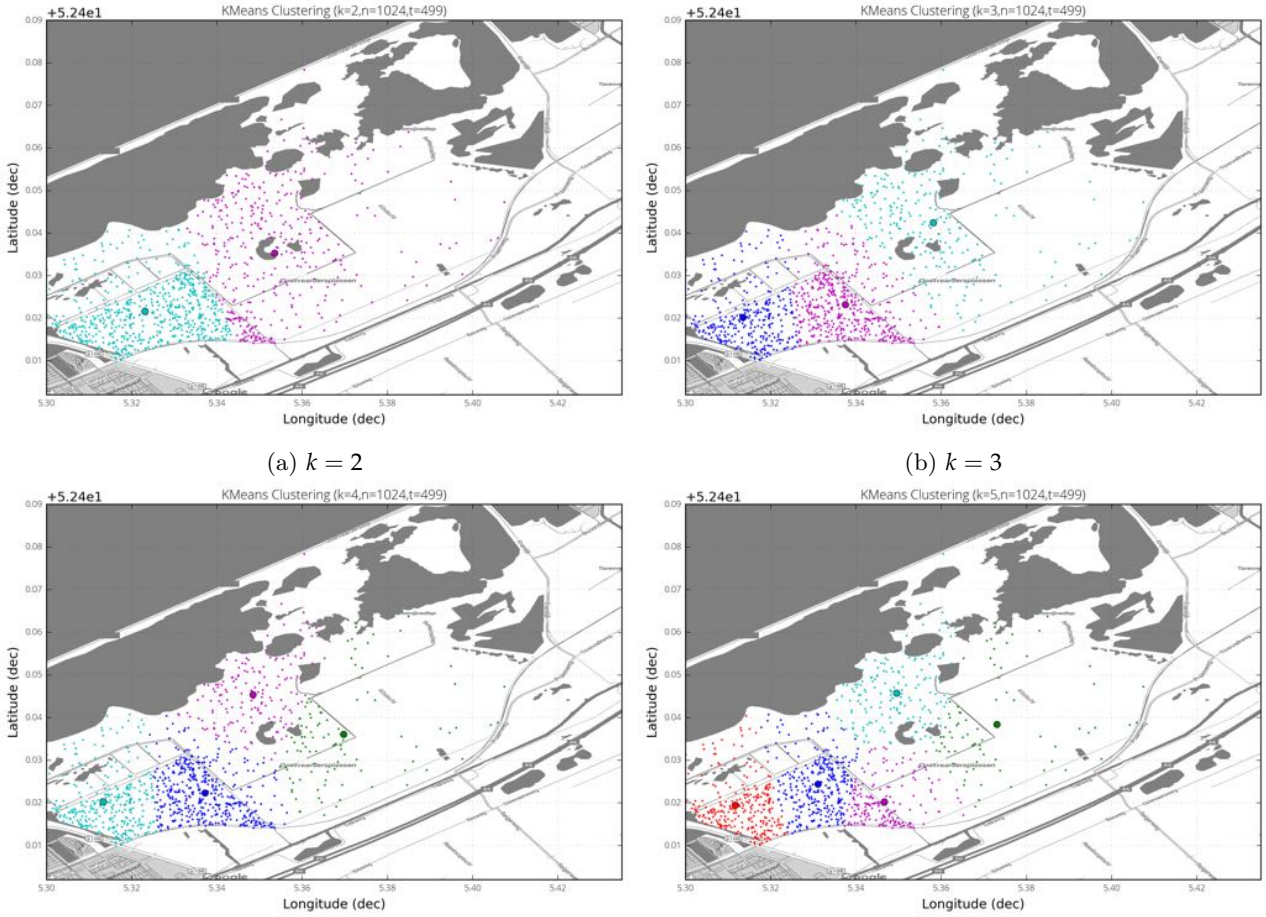
(b) $t_{max} = 4096$
$n = 1024$

Figure 4.2: The average minimal distances of each frame (AVG) for two data sets, containing positions for n horses in t_{max} timeframes. Also shown are the standard deviation (STDEV) and the average plus and minus the standard deviation (ARG±STDEV).

The second step is assigning all data points to the center point they are closest to, thus classifying them. The third step is to reposition the center point to perfectly average its group. The second and third step are repeated until the center point does not move (more than a certain threshold) anymore.

We compared the resulting graphs for $k = 2$ to $k = 5$, which are displayed in Figure 4.3. In the Figure 4.3.a, we see that for $k = 2$ the algorithm nicely almost perfectly divides the population into a group that is contained within the bottom-left (almost trapezoid) field and one that roams the larger fields. This is also what we saw in the Voronoi diagrams in Section 3.3.3. If we increase k, more subgroups are formed, but not much is gained. In the future, we might examine the minimum or average distance between horses for each subgroup to determine the coherence of that subgroup.

In the end, it seems not much is gained in this case by clustering the population. Depending on the animal species, real animals are more likely to display flocking behaviour. We believe therefore that the applicability and usefulness of clustering improve when using real-world data.



(a) $k = 2$ (b) $k = 3$

(c) $k = 4$ (d) $k = 5$

Figure 4.3: The k-means clustering algorithm applied to our data set ($n = 1024, t = 499$).

4.4 Chapter summary

In this chapter, we looked at two methods for analyzing data.

First, we looked at the distance between animals over time and used for each horse the distance to its nearest neighbour, which could be used to monitor the coherence in a herd of animals over time. To calculate the distance to each horse's nearest neighbour, we simply considered the distance between each horse and every other horse, resulting in $n(n-1)$ computations. We did bring this number down by only considering pairs we did not treat yet, to $\frac{1}{2}n(n-1)$, but should time become scarce when dealing with larger data sets, we suggest using a k-d tree to find each horses nearest neighbour. This should bring the number of computations down from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$.

Second, we looked at clustering of the animals. Again, this could be used to monitor herd behaviour in animals over time or at one point in time. Due to the semi-random behaviour of the animals in our fictional data set, we found two subgroups: one group of animals seemed to stay in the smaller area to the bottom left of the map (in a north-up orientation) and one group of animals scattered over the rest of the map. As implemented, this method required a human to choose a k-value (the number of subgroups to detect). This could be automated by evaluating the coherence of the group that is detected (e.g., by considering the distances in the group as discussed in the previous paragraph).

Chapter 5

Conclusions

***In this chapter**, we summarise the chapters and distill conclusions on the discussed matters by means of the problem statement and research questions.*

5.1 Problem statement and Research questions

Our problem statement was stated as follows:

PS: *How can we use spatio-temporal data of animals in wetlands to aid management and interest the public?*

The answer to this question lies in the conclusions we describe below, for each of the research questions. We summarise and distill conclusions from Chapters 2 to 4 by considering each of the three research questions. We can use the spatio-temporal data of animals by gathering them as described at RQ 1, visualise them as described at RQ 2 and analyse them as described at RQ 3.

RQ 1: *How can we gather spatio-temporal data of animals in wetlands?*

In Chapter 2, we discussed ten methods by which animals can be tracked, describing two categories. First, we looked at four 'direct' tracking methods, which involve the use of small tracking devices, sensors, that are placed upon an animal and send their data to a base station. With these methods, we focused on the technology the trackers use to communicate with the base station. ARGOS is an interesting project, but not suited to our case as the area we would like to monitor is rather small. More interesting are the 'traditional' option of mobile communication networks and new solutions in the form of LoRa and the UvA-BiTS system. Second, we looked at six 'indirect' tracking methods, which do not involve any interference in the habitat of the animal. Of these,

most were found not fit to use for detailed observation in the case of a wetland like the Oostvaardersplassen. However, the use of a video feed remains an interesting option that deserves more research.

RQ 2: *Which methods can be used to visualize the spatio-temporal data?*

In Chapter 3, we reviewed four data visualizations possibly applicable in our case. For this, we used two data sets: one real data set about Przewalski's horses in Mongolia, and one fictional data set, generated to behave like animals moving about the Oostvaardersplassen. The first option we implemented was plotting the locations of our targets on a 2D map. We then went on to examine three other ways of visualizing the spatio-temporal data: a 3D timegraph or 'isosurface', a Voronoi graph placed on top of a map image and a heatmap plotted over the same map image. The addition of a map image proved valuable in producing context for the moving points[1]. Initially, the heatmap did not look very interesting, but this changed when we increased the number of agents (n) in the data set. In the Voronoi plot however, this modification made it hard to distinguish the individual cells, which of course became a lot smaller. We tested this on a large screen[2] which did improve visibility and readability of the visualization (see Appendix H). From this we conclude that the Voronoi diagram is still useful for larger amounts of animals, but better when reviewing smaller groups. The heatmaps can be used to monitor two aspects of the nature reserve: 1) to find much used routes the animals take and 2) to reveal areas that are veryattractive to the animals.

RQ 3: *Which methods can be used to analyse the spatio-temporal data?*

In Chapter 4, we used the same data sets as in Chapter 3 to perform two analyses. First, we looked at the possibility of plotting and inspecting the average closest distance to a neighbour. Second, we used the k-means clustering algorithm to find clusters of animals. Both were reasonably interesting, but the data set used (the fictional set) was, as was to be expected by its nature, quite regular in its distributions. It would be interesting to see the same graphs when a data set is used with individuals that move more in a flock, as some species do. This behaviour could in the future be implemented in the script that generates the fictional locations.

5.2 Concluding remarks

It was interesting to take this first step into this field of biology with a focus on data gathering and visualization. We look forward to the possibilities of further gathering animal data as described in Chapter 2 and both improving the visualization and analyzation methods described in Chapter 3 and 4 and exploring more methods

[1] See especially the Voronoi animation videos `https://youtu.be/yjOgWMYbCUU` (without map) and `https://youtu.be/jL-e4e1Eyo8` (with map).

[2] Twelve 32" LCD televisions arranged in a 3x3 matrix, mounted to the wall of a room.

to get the most out of the data.

The model known as the 'DIKW Pyramid' describes four steps in working with data: **D**ata, **I**nformation, **K**nowledge and **W**isdom. Data can be transformed into information, which can lead to knowledge about a system and finally the wisdom to make decisions regarding the system. In this thesis, we formed the base of this pyramid: we described methods suitable to gather data and how to transform this data into information and knowledge. The final step lies, in our case, with providing the knowledge to the management of the nature reserve and working with them to improve the policies. By perfecting the methods described in this thesis, implementing them and deploying them, we can do just that.

Appendices

This section contains appendices to the main text:

A Hoekendijk's method

This method was used in [JH15] to distill GPS coordinates of a porpoise from a video still image.

B Horses in Python

The Python script used to generate spatio-temporal data for fictional horses moving around the Oostvaardersplassen.

C Matplotlib script

The Python script used to generate 3D plots without a map, as seen in Figure 3.8.

D KML converter script

The Python script used to convert KML files without a height attribute to a KML file with graph lines with the timestamp as height value.

E Przewalski's horse study

Detailed information about the study on Przewalski's horses conducted by Petra Kaczensky. This data set was used in Chapter 2: Data visualization.

F Voronoi plot script

The Python script used to generate Voronoi diagrams out of CSV files containing spatiotemporal data.

G Cesium example

The JavaScript script used in the example Cesium visualization in Chapter 2: Data visualization. The surrounding HTML code has been left out.

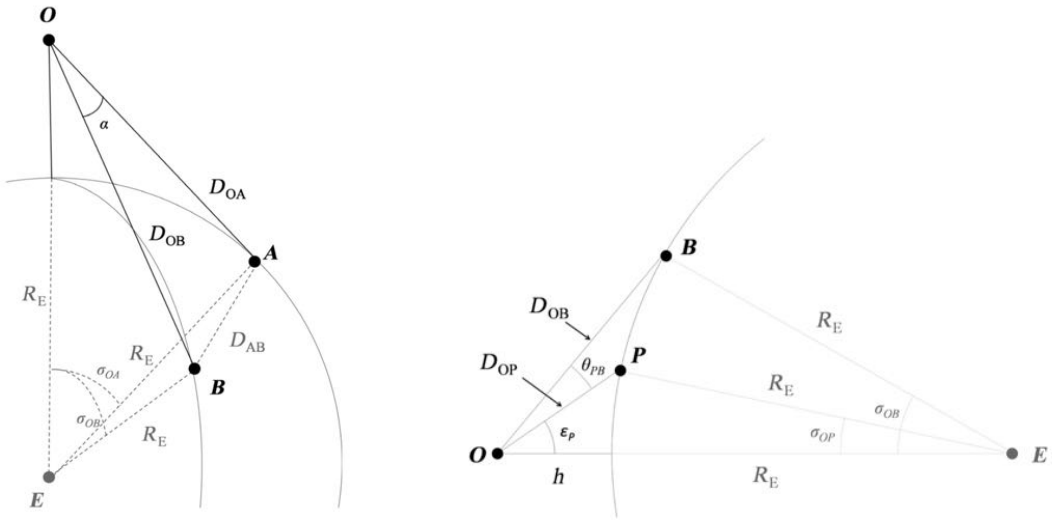
H Viewing on a large screen

Following the generation and evaluation of the several visualizations that we discussed, we tested whether some of the more cluttered visualizations would be improved when viewed on a larger screen.

A Hoekendijk's method

To determine the geographic location of a porpoise, we need a) the geographic position and height of the camera, b) the distance between the porpoise and the camera (D_{OP}) and c) the bearing of the sighting (the horizontal deviation from a reference point). To find b) and c), we follow the following steps. Once the distance of the observer to the porpoise and the bearing are known, we can use adapted versions of two functions R has built in, to get the geographic coordinates (see last step).

First two images: original paper [JH15]



I) Finding the distance between the observer and the porpoise (D_{OP})

1. Find σ_{OA}, σ_{OB} and σ_{AB}:

$$\sigma_{[X][Y]} = 2\sin^{-1}\left(\sqrt{\left(\sin\frac{\Delta\varphi_{[X][Y]}}{2}\right)^2 + \cos\varphi_{[X]}\cos\varphi_{[Y]}\left(\sin\frac{\Delta\lambda_{[X][Y]}}{2}\right)^2}\right) \tag{1}$$

2. Find D_{AB} (using the law of cosines):

$$D_{AB} = \sqrt{2R_E^2 - 2R_E^2\cos(\sigma_{AB})} \tag{2}$$

3. Find D_{OA} and D_{OB} (similar to the above):

$$D_{O[X]} = \sqrt{2R_E^2 + (R_E+h)^2 - 2R_E(R_E+h)\cos(\sigma_{O[X]})} \tag{3}$$

4. Find α (again using the law of cosines):

$$\alpha = cos^{-1}\left(\frac{D_{AB}^2 - D_{OA}^2 - D_{OB}^2}{-2D_{OA}D_{OB}}\right) \tag{4}$$

5. Find pixel size q (in radians) (using the Pythagorean theorem):

$$q = \frac{\alpha}{\sqrt{(A_y - B_y)^2 + (A_x - B_x)^2}} \tag{5}$$

6. Find ε_B (using the law of cosines):

$$\varepsilon_B = \cos^{-1}\left(\frac{(R_E + h)^2 + D_{OB}^2 - R_E^2}{2(R_E + h)D_{OB}}\right) = \cos^{-1}\left(\frac{2R_E h + h^2 + D_{OB}^2}{2(R_E + h)D_{OB}}\right) \tag{6}$$

7. Find θ_{PB} and, subsequently, ε_P:

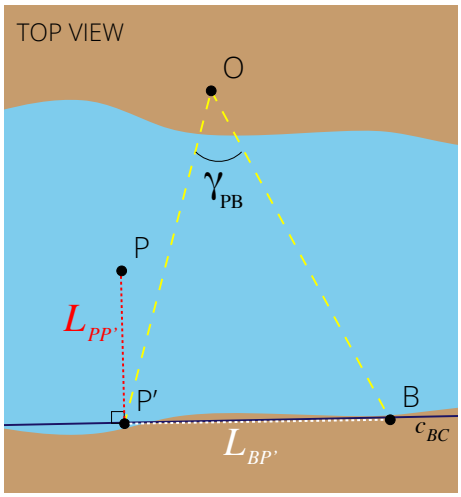
   See the original paper for the calculations to find θ_{PB}. In short, θ_{PB} is found by calculating $L_{PP'}$ (see step II) and multiplying it by q.

$$\varepsilon_P = \varepsilon_B - \theta_{PB} \tag{7}$$

8. Find D_{OP}:

$$D_{OP} = (R_E + h)\cos\varepsilon_P - \sqrt{(R_E + h)^2(\cos\varepsilon_P)^2 - (2hR_E + h^2)} \tag{8}$$

II) Finding the bearing relative to reference point B (γ_{PB})



To find the horizontal deviation from reference point B, we project P onto the artificial horizon line through B (c_{BC}) to get P'.

See the original paper for the calculations to find the coordinates of P' and the artificial horizon line through B, c_{BC}.

1. Find the length of line segment $L_{PP'}$:

$$L_{PP'} = \sqrt{(P_y - P'_y)^2 + (P_x - P'_x)^2} \tag{9}$$

2. Find the bearing:

$$\gamma_{PB} = q \cdot L_{BP'} \tag{10}$$

Finally, we use the distance between the observer and the porpoise and the bearing in the (adapted) functions `bearing` and `destPoint` of the R geosphere-package [RHV], which give us the latitude and longitude coordinates of the porpoise.

List of variables

$\boldsymbol{O}$	Position of observer
$\boldsymbol{A}$	Reference point A
$\boldsymbol{B}$	Reference point B
$\boldsymbol{P}$	Position of porpoise
h	Height of observer
α	Angle between A,O and B
$\boldsymbol{E}$	Center of the Earth
R_E	Radius of the Earth
$\sigma_{[X][Y]}$	Interior spherical angle between $[X]$ and $[Y]$
$D_{[X][Y]}$	Distance between $[X]$ and $[Y]$
$L_{[X][Y]}$	Distance in pixels between points $[X]$ and $[Y]$
q	Size of an individual pixel (in radians)
$\varepsilon_{[X]}$	Vertical angle between $[X]$, O and E
$\varphi_{[X]}$	Latitude of $[X]$
$\lambda_{[X]}$	Longitude of $[X]$
$\Delta\varphi_{[X][Y]}$	Difference in latitude between $[X]$ and $[Y]$
$\Delta\lambda_{[X][Y]}$	Difference in longitude between $[X]$ and $[Y]$
$[X]_x, [X]_y$	Pixel coordinates (x,y) of reference point $[X]$
$\gamma_{P[X]}$	Bearing of P relative to point $[X]$

B Horses in Python

This Python script was used to generate spatio-temporal data for fictional horses moving around the Oostvaardersplassen. It outputs a plaintext file containing rows of locations (as pixel coordinates x and y on the provided image). By way of commenting, it can be used to either output a KML or a CSV file.

```
from PIL import Image
import time
import sys
import random
import os
import math

def writeCSVHeader(id,rgb):
  #os.system("echo 'latitude,longitude,height' >> horses-csv/"+id+".csv")
  return

def writeKMLHeader(id,rgb):
  os.system("echo '<?xml version=\"1.0\" encoding=\"UTF-8\"?><kml xmlns=\"http://earth.google.com/
      kml/2.2\"><Document><Style id=\"style-1\"><LineStyle>        <color>"+rgb+"66</color>        <
      width>1</width>      </LineStyle>      <PolyStyle>        <color>"+rgb+"66</color>      </
      PolyStyle><IconStyle><color>"+rgb+"66</color><scale>1</scale><Icon><href>http://maps.google.
      com/mapfiles/kml/pal4/icon57.png</href></Icon></IconStyle></Style>        <Placemark>          
      <name>HorseID:"+id+"</name>          <visibility>1</visibility>          <description>HorseID:"
      +id+", color:"+rgb+"</description>          <styleUrl>#style-1</styleUrl>          <LineString>
                          <tessellate>1</tessellate>          <extrude>1</extrude>          <
      altitudeMode>absolute</altitudeMode>          <coordinates>' >> horses-kml/"+id+".line.kml")

def writeKMLCoords(x,y,z,id):
  os.system("echo "+str(pixToLongitude(x))+","+str(pixToLatitude(y))+","+str(z)+" >> horses-kml/"+id
      +".line.kml")

def writeCSVCoords(x,y,z,id):
  csvstrings[str(id)].append(str(pixToLongitude(x))+","+str(pixToLatitude(y))+","+str(z))
  #os.system("echo "+str(pixToLongitude(x))+","+str(pixToLatitude(y))+","+str(z)+" >> horses-csv/"+
      id+".csv")


def evaluatePosition(x,y,image):
  e = 0
  for n in range(1,len(x)):
    e_current = sum(image[int(x[n]),int(y[n])])
    e += e_current
  return e

```

```
30 def moveHorse(horse,image,i):
31   v = horse['speed']
32   r = horse['rotation']
33   x = horse['x'] + math.cos(math.radians(r))
34   y = horse['y'] + math.sin(math.radians(r))
35   temp_x = []
36   temp_y = []
37   for n in range(1,4):
38     temp_x.append(x + n*math.cos(math.radians(r)))
39     temp_y.append(y + n*math.sin(math.radians(r)))
40
41   if ((x <= 2*v or x >= (im.size[0]-2*v)) or (y <= 2*v or y >= (im.size[1]-2*v))):
42     return
43
44   while (evaluatePosition(temp_x,temp_y,image) < 700*(len(temp_x)-1) and True):
45     temp_x = []
46     temp_y = []
47     r = random.randint(0,360)
48     for n in range(1,4):
49       temp_x.append(x + n*math.cos(math.radians(r)))
50       temp_y.append(y + n*math.sin(math.radians(r)))
51
52   horse['x'] = x + v*math.cos(math.radians(r))
53   horse['y'] = y + v*math.sin(math.radians(r))
54   horse['rotation'] = r
55   if (mode == "kml"):
56     writeKMLCoords(horse['x'],horse['y'],i,horse['id'])
57   elif (mode == "csv"):
58     writeCSVCoords(horse['x'],horse['y'],i,horse['id'])
59
60
61 def printToTerminal(horse,image):
62   print(chr(27) + "[2J")
63   for i in range(0,im.size[1],50):
64     sys.stdout.write(str(i)+"␣|")
65     for j in range(0,im.size[0],25):
66       if horse['x'] == j and henk['y'] == i:
67         sys.stdout.write('X')
68       elif sum(pix[j,i]) > 100:
69         sys.stdout.write('.')
70       else:
71         sys.stdout.write('␣')
72     print ""
73
```

```
74 def newHorse(horse_name):
75   horse = {'id':horse_name,'x':1400, 'y':random.randint(1100,1300), 'speed':speed_horse, 'rotation':
         random.randint(0,360)}
76   horses.append(horse)
77   os.system("echo␣"+horse_name+".txt␣>>␣horsenames.txt")
78   os.system("echo␣"+str(len(horses))+"␣>␣no_of_horses.txt")
79
80
81 def horseDistance(horse1,horse2):
82   return abs(horse1['x'] - horse2['x']) + abs(horse1['y'] - horse2['y'])
83
84 def calculateDistances(horses):
85   mindistance = 1000000
86   for i,horse in enumerate(horses):
87     for j,horse2 in enumerate(horses):
88       if (j>i):
89         d = horseDistance(horse,horse2)
90         if (d<mindistance):
91           mindistance = d
92         #if (horseDistance(horse,horse2) < 100 and False):
93           #newHorse(horsenames.pop())
94   print mindistance
95
96 def printCoords(horse):
97   return str(horse['x'])+","+str(horse['y'])
98
99 def pixToLongitude(x):
100     return 5.478612 - (2876-x)*(0.241279/2538)
101
102 def pixToLatitude(y):
103     return 52.494943 - (y) * (0.093713/1614)
104
105
106 ##########################
107 # CREATE THE HORSES
108 ##########################
109
110 im = Image.open("images/map-flattened.bmp") #Can be many different formats.
111 pix = im.load()
112 print im.size[0] #Get the width and hight of the image for iterating over
113 print im.size[1]
114 size = im.size
115 horses =[]
116 timestep = 0.1
```

```
117 meter_per_pixel = 6.0
118 speed_kmph = 15.0
119 speed_mps = speed_kmph/3.6
120 speed_horse = timestep * speed_mps / meter_per_pixel
121 csvstrings = {}
122
123 #mode = 'kml'
124 mode = 'csv'
125
126 colors = ['194fdd','e442cb','bb896d','b2de11','b67405','d2df6a','5df5ea','252140','a21793','f0fe2b',
        '22e400','8a1f92','85fd28','d3ff6b']
127
128 if (mode == "csv"):
129   os.system("rm␣horses-csv/*.csv")
130   os.system("rm␣horses-csv.n1024.t4096/*.csv")
131 elif (mode == "kml"):
132   os.system("rm␣horses-kml/*.kml")
133
134 i_min = 0
135 i_max = 1024
136
137
138 print "Writing␣headers␣in␣"+mode+"␣mode..."
139 for i in range(i_min,i_max):
140   if (mode == "kml"):
141     writeKMLHeader(str(i),colors[i%len(colors)])
142   elif (mode == "csv"):
143     writeCSVHeader(str(i),colors[i%len(colors)])
144   csvstrings[str(i)] = []
145   newHorse(str(i))
146 print "Headers␣written."
147
148 j=0
149 j_max=4096.0
150
151 print "Starting␣movement␣generator..."
152 while (j<j_max):
153   for horse in horses:
154     moveHorse(horse,pix,j)
155     #os.system("echo "+printCoords(horse)+" > horses/"+horse['id']+".txt")
156
157   time.sleep(timestep)
158   j += 1
159   print str(round(float(j)/j_max*100,1))+ "%␣|␣" + str(j) + "/" + str(j_max) + "␣steps"
```

```
160 print "All␣movements␣have␣been␣generated."
161
162 print "Starting␣to␣write␣to␣output␣in␣mode␣"+mode+"..."
163
164
165 for i in range(i_min,i_max):
166   outputfile = open("horses-csv.n1024.t4096/"+str(i)+'.csv','w')
167   outputfile.write('longitude,latitude,height\n')
168   for line in csvstrings[str(i)]:
169     outputfile.write(line+'\n')
170   outputfile.close()
171   print "Written␣csv␣file␣for␣ID="+str(i)
172 print "Done␣writing␣to␣output."
173
174 '''
175 for i in range(i_min,i_max):
176   if (mode == "kml"):
177     print "Writing footer..."
178     os.system("echo '</coordinates></LineString></Placemark></Document></kml>' >> horses-kml/"+str(i
          )+".line.kml")
179     print "Footer written."
180   #elif (mode == "csv"):
181 '''
182
183 print "Finished."
```

C Matplotlib script

This Python script was used to generate 3D plots without a map, as seen in Figure 3.8.

```
import matplotlib as mpl
import numpy as np

#Importing pyplot
from matplotlib import pyplot as plt
from matplotlib import style
from matplotlib.dates import date2num
from matplotlib.dates import datestr2num

import matplotlib.dates as mdates
from mpl_toolkits.mplot3d import Axes3D

mpl.rcParams['examples.directory'] =
'/Users/janvanstaalduinen/Documents/Universiteit/Scriptie/test-python-plot/'

style.use('ggplot')

plt.title('Movement␣of␣Przewalski\'s␣horses')
plt.ylabel('Latitute')
plt.xlabel('Longitude')

# some that are close
animals = ['mundol','orchon','shagai','soir','tayan','todook']

# all of them
#animals = ['mondor','mundol','orchon','shagai','shandas','soir','tayan',
'todook']

# without outlier
#animals = ['mondor','mundol','orchon','shagai','soir','tayan','todook']

# these are very close
#animals = ['todook','shagai']

animals_data = []

for animal in animals:
  tmp = np.loadtxt('csv/'+animal+'.csv',
                 unpack=True,
                 delimiter = ',',
                 skiprows=1,
```

```
                converters={2: datestr2num},
                usecols=(4,5,2))
  animals_data.append(tmp)

fig = plt.figure(figsize=(10, 10))
ax = fig.gca(projection='3d')

years = mdates.YearLocator()   # every year
months = mdates.MonthLocator()  # every month
days = mdates.DayLocator()
yearsFmt = mdates.DateFormatter('%Y-%m-%d')
ax.zaxis.set_major_locator(years)
ax.zaxis.set_major_formatter(yearsFmt)
ax.zaxis.set_minor_locator(months)

ax.set_xlabel('longitude')
ax.set_ylabel('latitude')
ax.set_zlabel('date')

for index,animal in enumerate(animals_data):
  ax.plot(animal[0],animal[1],animal[2], label=animals[index], marker='.')

ax.legend()
plt.show()
```

D KML converter script

This Python script was used to convert KML files without a height attribute to a KML file with graph lines with the timestamp as height value. As can be observed, the output file contains a KML header and footer string, between which coordinates are printed.

```
###################### HEAD ######################
from time import mktime
from time import strptime
import xml.etree.ElementTree as ET
#import cElementTree as ET    # Safer, but has to be installed

################## DEFINITIONS ##################
date_format = "%Y-%m-%d␣%H:%M:%S"
ns = "{http://earth.google.com/kml/2.2}"    # namespace

minimum_date = strptime("2001-09-01","%Y-%m-%d")
date_factor = float(1)/7000

colors = ['6b65c0ff','33b694ff','f7a5d9ff','243ab4ff','b05810ff','625
e50ff','09e3b4ff','9e4a3dff','51a599ff','27cb50ff','b725fdff','b66737ff','07a977ff','c1ff1bff']

file_header1 = '<?xml␣version="1.0"␣encoding="UTF-8"?>\
<kml␣xmlns="http://earth.google.com/kml/2.2">\
<Document><Style␣id="style-1">\
<LineStyle>\
␣␣␣␣␣␣␣␣<color>'
file_header2 = '</color>\
␣␣␣␣␣␣␣␣<width>1</width>\
␣␣␣␣␣␣␣</LineStyle>\
␣␣␣␣␣␣␣<PolyStyle>\
␣␣␣␣␣␣␣␣<color>'
file_header3 = '</color>\
␣␣␣␣␣␣␣</PolyStyle>\
\
<IconStyle><color>ffff00ff</color><scale>1</scale><Icon><href>http://
maps.google.com/mapfiles/kml/pal4/icon57.png</href></Icon></IconStyle></Style>\
\
␣␣␣␣␣␣␣␣<Placemark>\
␣␣␣␣␣␣␣␣␣␣<name>Absolute</name>\
␣␣␣␣␣␣␣␣␣␣<visibility>1</visibility>\
␣␣␣␣␣␣␣␣␣␣<description>Transparent␣purple␣line</description>\
\
␣␣␣␣␣␣␣␣␣␣<styleUrl>#style-1</styleUrl>\
␣␣␣␣␣␣␣␣␣␣<LineString>\
```

```
        </tessellate>0</tessellate>\
        </extrude>0</extrude>\
        </altitudeMode>absolute</altitudeMode>\
        </coordinates>'

file_closer = '  </coordinates>\
      </LineString>\
    </Placemark>\
  </Document>\
</kml>'

###################### MAIN ######################
for i in range(1,10):
  handle = open("line"+str(i)+".kml",'w')

  handle.write(file_header1+colors[i]+file_header2+colors[i]+file_header3+'\n')

  path = "data/"+str(i)+"/points.kml"
  root = ET.parse(path).getroot()
  placemarks = root.find(ns+'Document').findall(ns+'Placemark')

  for placemark in placemarks:
    coordinates = placemark.find(ns+'Point').find(ns+'coordinates').text
    timestamp_text = placemark.find(ns+'Snippet').text
    timestamp_struct = strptime(timestamp_text, date_format)
    timestamp_adjusted = (mktime(timestamp_struct) -
    mktime(minimum_date)) * date_factor
    handle.write(coordinates+str(timestamp_adjusted)+'\n')

  handle.write(file_closer)

  handle.close()
```

E Przewalski's horses study

This table contains detailed information about the study on Przewalski's horses in Mongolia conducted by Petra Kaczensky. This data set was used in Chapter 2: Data visualization. **General**

Study Name	Przewalski's horse Mongolia 2002-2007
Contact Person & Principal Investigator	Petra Kaczensky (profile: `https://www.movebank.org/node/14589`
Citation	Kaczensky, P., Ganbaatar, O. and Walzer, C. 2008. Przewalski's horse GPS telemetry 2002-2007 dataset.
Acknowledgements	This research was conducted within the framework of the Przewalski's horse reintroduction project of the International Takhi Group (ITG)
Grants used	Funding was provided by the Austrian Science Foundation (FWF) projects P14992 and the Austrian National Bank (Jubileums Fonds).
License Terms	employ specific citation and acknowledgement terms
Study Summary	The aim of the study was to understand habitat and space use of reintroduced Przewalski's horses in the Mongolian Gobi. The data was needed as a basis for science based decision making within the framework of the Przewalski's reintroduction program.

Study reference location

Longitude	93.455
Latitude	45.341
Movebank ID	14291872

Study statistics (last updated > 1 year ago as of July 25th, 2016)

Number of Animals	10
Number of Tags	10
Number of Deployments	10
Time of First Deployed Location	2001-11-01 04:00:00.000
Time of Last Deployed Location	2005-07-04 23:01:00.000
Taxa	Equus caballus
Number of Deployed Locations	3813
Number of Records	Deployed (outliers) / Total (outliers)
GPS	3813 (0) / 0 (0)
Argos Doppler Shift	0 (0) / 0 (0)

F Voronoi plot script

This Python script was used to generate Voronoi diagrams out of CSV files containing spatiotemporal data. The script was adapted from the Git project 'Colorized Voronoi'[3].

Note: the image 'map-flattened.bmp' was added to the local Python Cookbook Samples folder, to be able to quickly access it from different locations and scripts.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial import Voronoi, voronoi_plot_2d
import csv
import os
import sys
import time
from scipy.misc import imread
import matplotlib.cbook as cbook


def formatNumberToThreeDigits(i):
  if (i > 99):
    return str(i)
  elif (i > 9):
    return "0"+str(i)
  else:
    return "00"+str(i)

def voronoi_finite_polygons_2d(vor, radius=None):
    """
    Reconstruct infinite voronoi regions in a 2D diagram to finite
    regions.

    Parameters
    ----------
    vor : Voronoi
        Input diagram
    radius : float, optional
        Distance to 'points at infinity'.

    Returns
    -------
    regions : list of tuples
        Indices of vertices in each revised Voronoi regions.
    vertices : list of tuples
        Coordinates for revised Voronoi vertices. Same as coordinates
```

[3] See https://gist.github.com/pv/8036995, last consulted August 14th, 2016

```
        of input vertices, with 'points at infinity' appended to the
        end.

    """

    if vor.points.shape[1] != 2:
        raise ValueError("Requires 2D input")

    new_regions = []
    new_vertices = vor.vertices.tolist()

    center = vor.points.mean(axis=0)
    if radius is None:
        radius = vor.points.ptp().max()

    # Construct a map containing all ridges for a given point
    all_ridges = {}
    for (p1, p2), (v1, v2) in zip(vor.ridge_points, vor.ridge_vertices):
        all_ridges.setdefault(p1, []).append((p2, v1, v2))
        all_ridges.setdefault(p2, []).append((p1, v1, v2))

    # Reconstruct infinite regions
    for p1, region in enumerate(vor.point_region):
        vertices = vor.regions[region]

        if all(v >= 0 for v in vertices):
            # finite region
            new_regions.append(vertices)
            continue

        # reconstruct a non-finite region
        ridges = all_ridges[p1]
        new_region = [v for v in vertices if v >= 0]

        for p2, v1, v2 in ridges:
            if v2 < 0:
                v1, v2 = v2, v1
            if v1 >= 0:
                # finite ridge: already in the region
                continue

            # Compute the missing endpoint of an infinite ridge

            t = vor.points[p2] - vor.points[p1] # tangent
```

```
82              t /= np.linalg.norm(t)
83              n = np.array([-t[1], t[0]])  # normal
84
85              midpoint = vor.points[[p1, p2]].mean(axis=0)
86              direction = np.sign(np.dot(midpoint - center, n)) * n
87              far_point = vor.vertices[v2] + direction * radius
88
89              new_region.append(len(new_vertices))
90              new_vertices.append(far_point.tolist())
91
92          # sort region counterclockwise
93          vs = np.asarray([new_vertices[v] for v in new_region])
94          c = vs.mean(axis=0)
95          angles = np.arctan2(vs[:,1] - c[1], vs[:,0] - c[0])
96          new_region = np.array(new_region)[np.argsort(angles)]
97
98          # finish
99          new_regions.append(new_region.tolist())
100
101     return new_regions, np.asarray(new_vertices)
102
103 def getHorses(path):
104
105   horses = []
106   n = 0
107   for file in os.listdir(datapath):
108       if (file.endswith(".csv") and n < 140000):
109         horses.append(file[:-4])        # remove '.csv'
110         n += 1
111   return horses
112
113 def getData(horses,path):
114   horses_data = {}
115   for horse in horses:
116       tmp_longs = []
117       tmp_lats = []
118       with open(datapath+horse+'.csv', 'rb') as csvfile:
119           csvreader = csv.reader(csvfile,delimiter=',')
120           csvreader.next()    # skip first row with column names
121           for row in csvreader:
122               tmp_longs.append(float(row[0]))
123               tmp_lats.append(float(row[1]))
124       horses_data[horse] = { 'longitude' : tmp_longs, 'latitude' : tmp_lats }
125   return horses_data
```

```
126
127 def getPoints(data):
128   points = []
129   for i in range(0,int(len(data['0']['longitude']))):
130       tmp = []
131       for horse in data:
132           #print horses_data[horse]['longitude'][i]
133           new = np.array([data[horse]['longitude'][i],data[horse]['latitude'][i]])
134           #print new
135           tmp.append(new)
136       points.append(tmp)
137   return points
138
139
140 def makePlot(current_frame,points):
141   print "Plotting␣frame␣"+str(current_frame)+"/"+str(sys.argv[1])+"..."
142
143   fig = plt.figure(figsize=(80,60))
144
145   ax = fig.add_subplot(111)
146
147   vor = Voronoi(points)
148   regions, vertices = voronoi_finite_polygons_2d(vor)
149
150   plt.xlim(vor.min_bound[0] - 0.1, vor.max_bound[0] + 0.1)
151   plt.ylim(vor.min_bound[1] - 0.1, vor.max_bound[1] + 0.1)
152   ax.set_xlabel('longitude␣(decimals)')
153   ax.set_ylabel('latitude␣(decimals)')
154
155   # colorize
156   for region in regions:
157       polygon = vertices[region]
158       plt.fill(*zip(*polygon), alpha=0.4)
159
160   plt.plot(points[:], points[:], 'ko',zorder=1)
161   ax.set_title('Voronoi␣diagram␣of␣locations␣of␣horses␣(n='+str(len(horses))+',t␣=␣'+str(
        current_frame)+")")
162   voronoi_plot_2d(vor,ax=ax, show_points=True, show_vertices=False)
163   ax.axis([5.30 , 5.435, 52.402, 52.49])
164
165   plt.imshow(img, zorder=0, extent=[5.237333 , 5.478612, 52.401230, 52.494943])
166
167   plt.gca().set_aspect('equal', adjustable='box')
168
```

```
169   print "␣␣[Plot␣built,␣saving␣to␣png]"
170   plt.savefig('renders/faster.4096/t'+formatNumberToThreeDigits(current_frame)+'.png', bbox_inches='
          tight')
171   print "␣␣[Plot␣saved]"
172
173   plt.close()
174
175 ##### MAIN #####
176
177
178 # Common settings
179 datafile = cbook.get_sample_data('map-flattened.bmp')
180 img = imread(datafile)
181 font = {'size' : '35'}
182
183 lowerbound = int(sys.argv[1])
184 upperbound = int(sys.argv[2])
185 dataset = str(sys.argv[3])
186
187 datapath = "../massive-dataset/csv/outputcsv."+dataset+"/"
188
189 horses = getHorses(datapath)
190 print "File␣list␣loaded.␣(1/4)"
191 data = getData(horses,datapath)
192 print "Data␣set␣loaded.␣(2/4)"
193 points = getPoints(data)
194 print "Data␣points␣loaded.␣(3/4)"
195 for i in range(upperbound,lowerbound-1,-1):
196   makePlot(i,points[i])
197   print "␣␣[Plots␣left:␣"+str(i-lowerbound)+"]"
198 print "Plotting␣complete.␣(4/4)"
```

G Cesium example

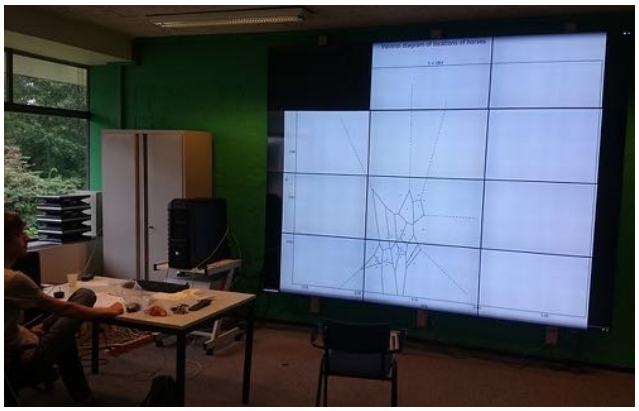
This JavaScript script was used in the example Cesium visualization in Chapter 2: Data visualization. The surrounding HTML code has been left out. The HTML element containing the Cesium widget is identified by identifier 'cesiumContainer'.

```
function loadCSVFile(filepath,startdate,datestepsize){
  var values = [];
  var startDateNumber = startdate;  // a date in Julian Date format
  var stepDateNumber = datestepsize;
  var txtFile = new XMLHttpRequest();
  txtFile.open("GET", filepath, false);
  txtFile.onreadystatechange = function(){
    if (txtFile.readyState === 4) {  // document is ready to parse.
      if (txtFile.status === 200) {  // file is found
        allText = txtFile.responseText;
        lines = txtFile.responseText.split("\n");
      for (var i=1; i < lines.length-1; i+=resolution){ // skip header row
        lineSplitted = lines[i].split(',');
        valueArray = [ startDateNumber + i*stepDateNumber, lineSplitted[0], lineSplitted[1] ];
        values.push(valueArray);
      }
      }
    }
  }
  txtFile.send(null);
  return values;
}

function newIntervalFromDaysNumber(startdate,stopdaysnumber){
  var timeInterval = new Cesium.TimeInterval({
      start : new Cesium.JulianDate(startdate),
    stop : new Cesium.JulianDate(startdate+stopdaysnumber),
      isStartIncluded : true,
      isStopIncluded : false
  });
  return timeInterval
}

function newWall(lon1,lat1,z1,lon2,lat2,z2,widget,interval,color){
  var timeIntervalCollection = new Cesium.TimeIntervalCollection([interval]);
  return widget.entities.add({
      name : 'Green wall from surface with outline',
    availability : timeIntervalCollection,
      wall : {
          positions : Cesium.Cartesian3.fromDegreesArrayHeights([
        lon1,lat1,z1, // A
        lon2,lat2,z2, // B
      ]),
          material : Cesium.Color.WHITE.withAlpha(0.25),
      outline : true,
      outlineColor : color
      }
  });
}

function plotAgent(values,color){
  var wall;
  var i = 0;
  prev_lon = values[i][1];
  prev_lat = values[i][2];
  for (var i; i < values.length; i++){
    wall = newWall(prev_lon, prev_lat, i*heightStepSize, values[i][1], values[i][2],
            (i+1) * heightStepSize, viewer,
            newIntervalFromDaysNumber(values[i][0],retentionDaysNumber),color);
    prev_lon = values[i][1];
    prev_lat = values[i][2];
  }
}

var heightStepSize = 0.5;   //
```

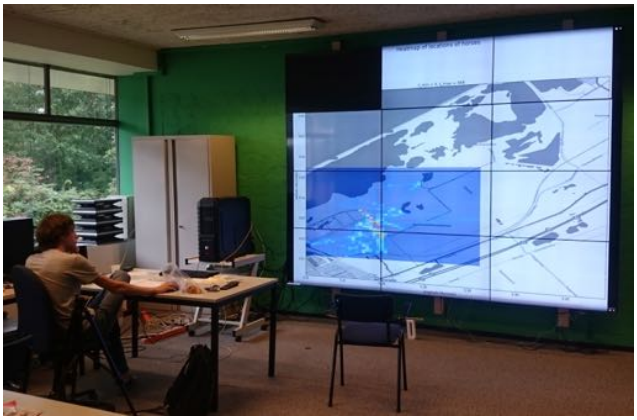
```
66  var resolution = 5;       // Every Xth step is skipped
67  var retentionDaysNumber = 5;  //
68
69  var viewer = new Cesium.Viewer('cesiumContainer');
70
71  var n = 25;
72  for (var i = 0; i < n; i++){
73    values = loadCSVFile('csv/'+i+'.csv',2457610.5,0.01);
74    plotAgent(values,Cesium.Color.fromRandom());
75    console.log("Loaded␣"+i+"/"+(n-1));
76
77  }
78
79  viewer.camera.flyTo({
80      destination : Cesium.Cartesian3.fromDegrees(5.3747,52.3922,1250.0),
81          orientation : {
82              heading : Cesium.Math.toRadians(-25.0),
83              pitch : Cesium.Math.toRadians(-25.0),
84              roll : 0.0
85          }
86      });
```

H Viewing on a large screen

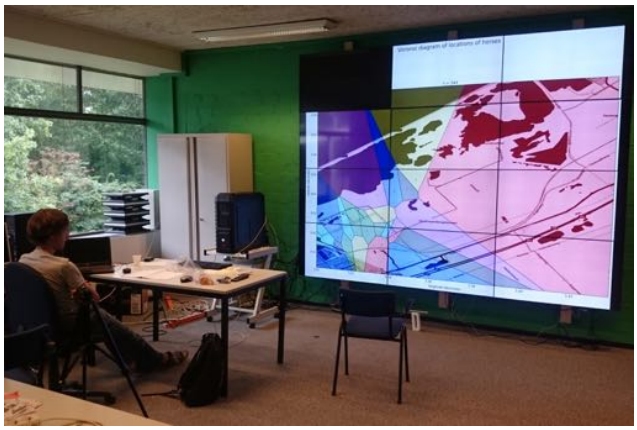
Following the generation and evaluation of the several visualizations that we discussed, we tested whether some of the more cluttered visualizations would be improved when viewed on a larger screen.



(a) Voronoi graph (without colors or map)



(b) Heatmap



(c) Voronoi graph

(d) Voronoi graph (large data set)

Figure 1: Inspecting the visualizations on a larger screen, consisting of twelve 32” televisions mounted to the wall of a room (unfortunately, one screen was not operational on the day these pictures were taken).

List of Tables

List of Figures

References

[AMM+07] Wolfgang Aigner, Silvia Miksch, Wolfgang Müller, Heidrun Schumann, and Christian Tominski. Visualizing time-oriented data—a systematic view. *Computers & Graphics*, 31(3):401–409, 2007.

[APN12] Stefan Aust, R Venkatesha Prasad, and Ignas GMM Niemegeers. Ieee 802.11 ah: Advantages in standards and further challenges for sub 1 ghz wi-fi. In *Communications (ICC), 2012 IEEE International Conference on*, pages 6885–6889. IEEE, 2012.

[Bal09] R. Balasubramaniam. On the confirmation of the traditional unit of length measure in the estimates of circumference of the earth. *Current Science (00113891)*, 96(4):547 – 552, 2009.

[BBSBC13] Willem Bouten, Edwin W. Baaij, Judy Shamoun-Baranes, and Kees C. J. Camphuysen. A flexible gps tracking system for studying bird behaviour at multiple scales. *Journal of Ornithology*, 154(2):571–580, 2013.

[BC00] A. S. Barry and J. Czechanski. Ground surveillance radar for perimeter intrusion detection. In *Digital Avionics Systems Conference, 2000. Proceedings. DASC. The 19th*, volume 2, pages 7B5/1–7B5/7 vol.2, 2000.

[Bla15] Herbert Blankesteijn. Breekt het 'internet of things' nu eindelijk door? (dutch). *NRC.nl*, 08 2015.

[Cla89] D. D. Clark. Overview of the argos system. In *OCEANS '89. Proceedings*, volume 3, pages 934–939, Sept 1989.

[Col10] Christina Wynne Collins. Understanding the reproductive biology of the przewalski's horse (equus ferus przewalskii). 2010.

[DAXX] Andrew B. Davies and Gregory P. Asner. Advances in animal ecology from 3d-lidar ecosystem mapping. *Trends in Ecology & Evolution*, 29(12):681–691, 2016/06/13 XXXX.

[DEE+04] Holger Dettki, Göran Ericsson, Lars Edenius, et al. Real-time moose tracking: an internet based mapping application using gps/gsm-collars in sweden. *Alces*, 40:13–21, 2004.

[DH72] Richard O Duda and Peter E Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972.

[H+07] John D Hunter et al. Matplotlib: A 2d graphics environment. *Computing in science and engineering*, 9(3):90–95, 2007.

[HHWN02] Susan Havre, Elizabeth Hetzler, Paul Whitney, and Lucy Nowell. Themeriver: Visualizing thematic changes in large document collections. *IEEE transactions on visualization and computer graphics*, 8(1):9–20, 2002.

[JH15] Krissy van der Bolt Jens Greinert Sophie Brasseur Kees C. J. Camphuysen & Geert Aarts J.P.A. Hoekendijk, Jurre de Vries. Estimating the spatial position of marine mammals based on digital camera recordings. *Ecology and Evolution*, 5(3):578—589, 2015.

[Mat14] Mark P. Mattson. Superior pattern processing is the essence of the evolved human brain. *Front Neurosci*, 8:265, Aug 2014. 25202234[pmid].

[MKG+13] Aongus McCarthy, Nils J. Krichel, Nathan R. Gemmell, Ximing Ren, Michael G. Tanner, Sander N. Dorenbos, Val Zwiller, Robert H. Hadfield, and Gerald S. Buller. Kilometer-range, high resolution depth imaging via 1560 nm wavelength single-photon detection. *Opt. Express*, 21(7):8904–8915, Apr 2013.

[Moe04] Andrew Vande Moere. Time-varying data visualization using information flocking boids. In *Information Visualization, 2004. INFOVIS 2004. IEEE Symposium on*, pages 97–104. IEEE, 2004.

[QMdJ+12] Lorenzo Quaglietta, Bruno Herlander Martins, Addy de Jongh, António Mira, and Luigi Boitani. A low-cost gps gsm/gprs telemetry system: performance in stationary field tests and preliminary data on wild otters (lutra lutra). *PloS one*, 7(1):e29235, 2012.

[RHV] E. Williams R.J. Hijmans and C. Vennes.

[ROS07] Sahar Rahimi and Norman Owen-Smith. Movement patterns of sable antelope in the kruger national park from gps/gsm collars: a preliminary assessment. *South African Journal of Wildlife Research*, 37(2):143, 2007.

[Shr14] Ayush Shrestha. Visualizing spatio-temporal data. 2014.

[TAS04] Christian Tominski, James Abello, and Heidrun Schumann. Axes-based visualizations with radial layouts. In *Proceedings of the 2004 ACM symposium on Applied computing*, pages 1242–1247. ACM, 2004.

[vdH16] J.H. van den Herik. Big data. In *Leids-Haags Allegorieënlexicon*, pages 41–49. Leiden University Dual PhD Centre The Hague, Uitgeverij Eburon, Delft, 2016.

[vdZ13] Niels van der Zon. Kwaliteitsdocument AHN2 (dutch). Technical report, Actueel Hoogtebestand Nederland, May 2013.

[VMRF02] Cécile Vincent, Bernie J. Mcconnell, Vincent Ridoux, and Michael A. Fedak. Assessment of argos location accuracy from satellite tags deployed on captive gray seals. *Marine Mammal Science*, 18(1):156–166, 2002.

[vS16] J.H. van Staalduinen. De wet versus het internet — uitdagingen van een internet of things (dutch) (available on `https://www.liacs.nl/~jvstaald/iotpaper.pdf`). 05 2016.

[WK14] M Wikelski and R Kays. Movebank: archive, analysis and sharing of animal movement data. *World Wide Web electronic publication*, 2014.

[WP] Dominic Zeng Wang and Ingmar Posner. Two approaches to radar-based moving object detection.