

The present work was submitted to the LuFG Theory of Hybrid Systems

Wind Turbine Shadow Cast Simulation using Raycasting

Simulation von Schlagschatten von Windturbinen mittels Raycasting

Bachelor of Science Thesis

November 2024

Presented by Vorgelegt von	Matei Cristian Nedelcu Matrikelnummer: 422871 matei.cristian.nedelcu@rwth-aachen.de
First examiner Erstprüfer	Univ.-Prof. Dr. rer. nat. Erika Ábrahám Lehr- und Forschungsgebiet: Theorie der hybriden Systeme RWTH Aachen University
Second examiner Zweitprüfer	Univ.-Prof. Dr. rer. nat. Jürgen Giesl Lehr- und Forschungsgebiet: Programmiersprachen und Verifikation RWTH Aachen University
Supervisor Betreuer	Dr. rer. nat. Pascal Richter Lehr- und Forschungsgebiet: Theorie der hybriden Systeme RWTH Aachen University

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Related work	2
1.3	Contribution	2
1.4	Outline	2
2	Preliminary data and setup	4
2.1	Guidelines for determining and assessing optical immissions from wind turbines	4
2.2	Sun position calculation	5
2.3	GIS data and square grid model	7
2.4	JSON files definitions	8
2.5	Simulation workflow	9
3	Modelling the shadow cast of wind turbines	13
3.1	Astronomical maximal shadow cast	13
3.1.1	Starting the simulation	13
3.1.2	Sun vector and rotor plane normal vector	14
3.1.3	Forwards raycasting approach	14
3.1.4	Backwards raycasting approach	17
3.2	Simulation results	20
3.3	Realistic weather data shadow cast	21
3.4	Turbine shutdown	23
4	Results verification	26
4.1	Importing custom projects via the API	26
4.2	Testing scenario	26
4.3	Results	27
5	Profiling and execution time	32
5.1	Profiling	32
5.2	Process parallelization	33
5.3	Runtime	34
5.4	Results	35
	Conclusion and Outlook	36
5.1	Future work	36
	Appendix	38
	Bibliography	39

1 Introduction

1.1 Motivation

Wind power has emerged as one of the leading renewable energy sources in the global effort to transition away from fossil fuels. It harnesses the kinetic energy of wind to generate electricity, offering a clean, abundant, and renewable alternative to traditional energy sources. Wind power contributes significantly to reducing greenhouse gas emissions, helping to combat climate change. Moreover, wind energy is becoming increasingly cost-competitive, making it an attractive option for many countries looking to diversify their energy mix and reduce their reliance on imported fuels.

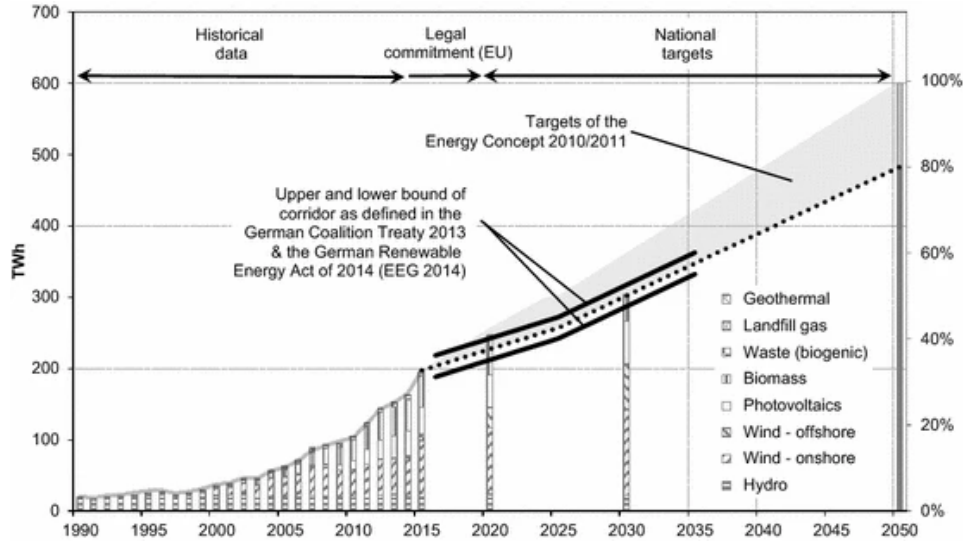


Figure 1: Planned energy source transition in Germany [1].

Germany is at the forefront of the global race for wind power, being the third largest producer of wind power worldwide by installations, behind only China and the United States [2]. As of 2023, wind power makes up the biggest energy source in Germany at around 27 percent [3], ahead of traditional fossil fuel sources such as coal or gas. The number of on-shore wind farms and the total energy output has been steadily increasing which clearly shows that wind energy is of big importance in Germany. This fact is further established by Germany's plan to become carbon neutral by 2050 as supported by Figure 1.

While the benefits of wind as a clean energy source can hardly be disputed, wind turbines as energy harvesting machines have a few downsides. In Germany, there has been increasing resistance from both NGOs and residents [4] against wind turbine farms, due to the negative effects of wind turbines on the surrounding nature and residential areas. Besides the possibility of damaging wildlife populations, wind turbines can also have a negative impact on residents in the vicinity of the turbine due to effects such as the emission of low-frequency noise and shadow flicker.

Shadow flicker occurs when the rotating blades of a wind turbine pass in front of the sun, casting a periodic shadow on the ground that may result in a flickering phenomenon which can be bothersome to humans. While most countries have little to no regulations in place that deal with wind turbine shadow flicker, Germany has exact values defined for the maximal allowed total shadow cast on a residential home. While the German Federal Immission Control Act [5] does not specifically address shadow flicker, there exists an industry-standard document that limits the yearly and daily wind turbine shadow cast and provides additional guidelines on the simulation parameters and required output.

1.2 Related work

There exist several specialized software tools designed for wind turbine farms layout design and optimization which sometimes include a shadow cast module such as WindPRO by EMD¹, WindFarm by ReSoft² and Shadow by WIND-consult³. These tools are usually aimed at professionals, require a lot of input parameters and are also expensive. As far as scientific papers go, research mostly focuses on the effect of shadow-flicker on humans [6], not on actual methods on how the shadow cast can be accurately computed.

1.3 Contribution

This work aims to develop a robust, fast, and accurate model for calculating wind turbine shadow cast areas. To achieve this, we use two distinct raycasting methods that determine the shadowed regions efficiently and precisely. The simulation includes both worst-case and real weather data scenarios, allowing for flexible and realistic shadow simulations. Additionally, it features a turbine shutdown calendar, enabling precise scheduling of operational hours to mitigate shadow impact, and generates a detailed report output that visualizes shadow duration and affected areas. Finally, the model is validated against the industry-standard WindPRO software to ensure its reliability and accuracy, providing a comprehensive tool for assessing and managing wind turbine shadow impacts.

1.4 Outline

Section 2 covers the data sources and processing steps required for the shadow cast simulation and an overview of the simulation workflow. Section 3 describes the simulation methodology, providing details on the worst-case and real weather scenarios, the two distinct raycasting methods, and the shutdown model. Section 4 compares the simulation results from three wind turbine layouts with the WindPRO software. Section 5 examines runtime performance, profiling, and the multi-processing approach employed by the simulation to improve runtime significantly. Finally, the paper concludes with a summary

¹<https://www.emd-international.com/windpro/>

²<https://www.resoft.co.uk/>

³<https://www.wind-consult.de/shadows-casted-on-areas/?lang=en>

of the contributions presented in this thesis and examines potential improvements and features.

2 Preliminary data and setup

Before analyzing the details of modeling the shadow cast by onshore wind turbines, it is essential to understand the building blocks that lie at the simulation’s foundation and contribute to its accuracy and functionality. This section explores the key components required for setting up the simulation environment, from understanding the sun’s position to using geospatial data in an efficient and useful way. German guidelines on determining and assessing wind turbine shadow cast are also presented.

2.1 Guidelines for determining and assessing optical immissions from wind turbines

In Germany, the guidelines for assessing shadow cast by wind turbines are detailed in an 11-page document titled *Hinweise zur Ermittlung und Beurteilung der optischen Immissionen von Windenergieanlagen (WEA-Schattenwurf-Hinweise)* [7]. These guidelines outline key simulation assumptions and provide a standardized methodology to evaluate, mitigate, and control shadowing effects of wind turbines. To model shadow cast effectively, the German guidelines recommend two distinct shadowing scenarios:

- **Worst-case (astronomical maximal) scenario:** This scenario uses a set of idealized and simplified assumptions designed to estimate the maximum potential shadowing impact. The sun shines from sunrise to sunset without any atmospheric obstruction from clouds and the turbine rotors are always perpendicular to the sun’s ray, thus maximizing the shadow.
- **Meteorological accurate scenario:** The second scenario uses historical weather data to compute a more accurate shadowing simulation, accounting for cloud cover and varying sunlight area.

The document guidelines further specify so-called *protection zones* that should be prioritized for shadow impact assessments. These areas include residential buildings and public facilities such as schools or hospitals.

Lastly, specific regulatory shadowing thresholds are given. For a *protection zone* the shadowing exposure in the worst-case scenario is limited to:

- 30 hours per year
- 30 minutes per day

For the calculation methodology itself, the guidelines state that a whole year in 1-minute time steps has to be simulated, sun positions where the elevation angle is below 3° can be omitted and 120 W/m^2 lower limit for the considered solar irradiance. The guidelines reference two calculation methods that are foundational to shadow cast modeling, but due to the age of the documents and their referenced studies, no digital versions of these papers could be found.

2.2 Sun position calculation

The first step in the computation of the shadowing of a real-life object is identifying the main light source, which in our case is the sun. As the sun's position in the sky changes, it casts shadows of different lengths on wind turbines. When the sun is low, close to the horizon, the light rays will be almost parallel to the ground, thus producing the longest shadow. In contrast, when the sun is highest in the sky, the light rays fall practically perpendicular to the earth's surface, thus casting the smallest shadow. Figure 2 shows this effect.

The sun's position is a function of both time and geographic location. Throughout the year, as the earth orbits around the sun, from the perspective of an observer on the earth's surface, the sun follows a fixed path also known as the ecliptic path. Thus, using a mathematical model, it is possible to calculate the position of the sun for a given geographical location, date, and time with an accuracy as low as $\pm 0.0003^\circ$. The most commonly used algorithm for calculating the sun's position is the Sun Position Algorithm (SPA) [8], developed by the US National Renewable Energies Laboratory (NREL). While this method is probably the most accurate, depending on the implementation, it can also be quite slow, especially when using a Python implementation. Because for this project a full calendar year of sun positions in 1 minute time step (525.600 positions) has to be computed, we chose a different sun position algorithm called SolTrack [9]. While the accuracy is only slightly and negligibly lower than an NREL SPA algorithm, it is much faster.

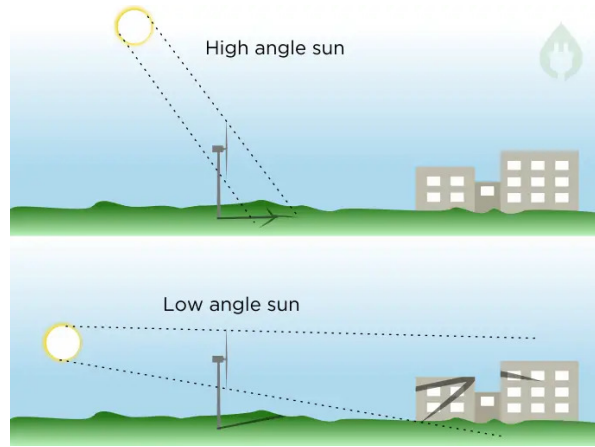


Figure 2: Sun position affects wind turbine shadow [10].

Before we describe how the SolTrack algorithm was used, we have to explain the azimuth and altitude (or elevation) angles of the sun. The solar azimuth angle is a horizontal coordinate that defines the sun's relative direction along the local horizon. The convention most commonly accepted when analyzing solar irradiation is to measure the azimuth with 0° corresponding to the North and angles increasing clockwise: 90° corresponds to the East, 180° to the South, and 270° to the West. The altitude angle

is the angular height of the sun measured from the horizontal. This means the altitude angle is 0° at sunrise and 90° when the sun is directly overhead.

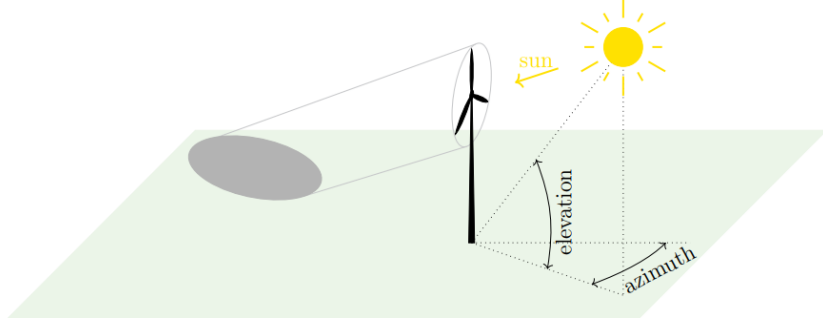


Figure 3: Turbine shadow dependent on sun azimuth and elevation angle [11].

First, we create a Pandas⁴ Data Frame using the date information from the Shadow Cast Settings JSON file including the number of relevant days (for how many days of the year we will later compute the shadow cast) and the time step (how often should a new solar position be computed) We then create a SolTrack object for a turbine's latitude and longitude and compute the sun's azimuth and altitude angles in degrees for the entire date range. Lastly, we save this data in a multi-level indexed Pandas Data Frame. For an entire year using one-minute time steps, the data frame will look like Table 1.

Day	Time Step	Time	Azimuth	Altitude
2021-01-01	0	00:07:00	344.250341	-61.478069
	1	00:08:00	344.724124	-61.520213
	2	00:09:00	345.199025	-61.561096
	3	00:10:00	345.675017	-61.600714
	4	00:11:00	346.152067	-61.639060
...	...	...	...	...
2021-12-31	1434	23:54:00	338.247739	-60.844829
	1435	23:55:00	338.704951	-60.902663
	1436	23:56:00	339.163607	-60.959322
	1437	23:57:00	339.623684	-61.014799
	1438	23:58:00	340.085163	-61.069087

Table 1: Sun position data for specific days, showing azimuth and altitude at various time steps.

⁴<https://pandas.pydata.org/>

2.3 GIS data and square grid model

Geographic Information Systems (GIS) are computer systems designed to capture, store, aggregate, and process data associated with geographic locations on the Earth's surface. In the context of this simulation, GIS data provides the critical spatial framework that defines the project area, including the locations of wind turbines, receivers, residential buildings, and relevant topographic features, each as a separate layer in the GIS data structure. Figure 4 taken from [12] showcases an example data layering system.

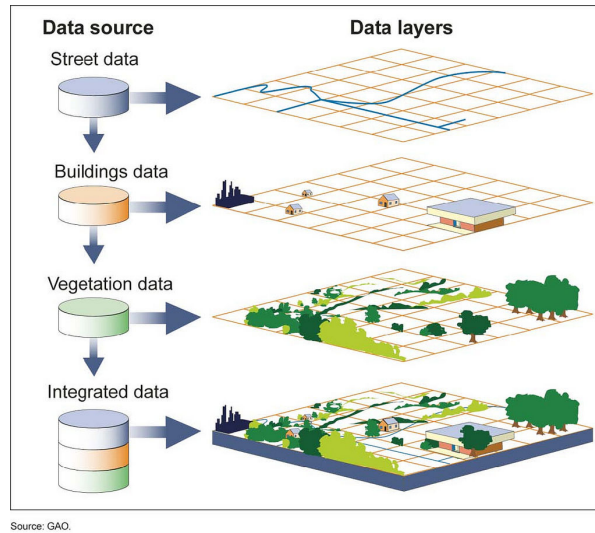


Figure 4: GIS Data Layers including street, buildings and vegetation data.

The first step in processing GIS data is defining the project area, which will later accommodate the wind turbines and receivers. This involves specifying the minimum and maximum latitude and longitude coordinates and then creating a custom coordinate projection where the origin of the coordinate system is set at the lower-left corner of the project area. Using a coordinate transformer, latitude and longitude pairs can then be converted into (x, y) coordinates, representing distances in meters relative to the system's origin, ensuring accurate spatial representation for the simulation.

The next step involves creating a custom square cell grid for our project area, represented as a 2-dimensional array. Using the defined project area and variable square grid size, we construct the array such that each (x, y) coordinate in the custom coordinate projection corresponds to the center of each cell. The value of each cell captures topographic information downloaded from the Viewfinder Panoramas⁵ Digital Elevation Model, which improves the accuracy of our shadow cast simulation by accounting for variations in terrain height.

⁵<https://www.viewfinderpanoramas.org/>

For the buildings layer, Open Street Map⁶ was used to download buildings as polygons with lat/long coordinates for each polygon point. Using this layer, we then compute a boolean mask for the grid, marking exactly which cells contain a residential building. This mask will be used to determine whether a wind turbine shadows a residential area.

In the simulation, the grid and boolean mask are essential for evaluating how shadows move across the terrain throughout the day and year. Incorporating topography ensures that shadows are modeled realistically, while the building mask helps assess the potential impact on residential buildings, making the simulation both accurate and practical for real-world applications.

2.4 JSON files definitions

In this wind turbine shadow cast simulation, several JSON files are used to define key parameters and input data. These files provide essential information such as the project area's geographic boundaries, the location and technical specification of wind turbines, and the layout of residential buildings. By organizing the input data in separate JSON files, we ensure that the simulation is both flexible and easily modifiable. The following section goes through each input JSON file, briefly explaining the role that each field plays in the simulation.

► **Project settings** This file contains general information necessary for setting up the initial project area, custom coordinate projection, and square grid.

- `lat_long_boundaries` min/max latitude and longitude coordinates.
- `square_size_m` Dimensions of the square grid cell in meters.

► **Receivers** This file contains information about the receivers where the shadow duration has to be analyzed. Receivers are rectangles and can either represent buildings or simple geographic locations.

- `id` String identification of the receiver.
- `latitude` Latitude coordinate.
- `longitude` Longitude coordinate.
- `full_address` City, postal code, street and house number of the receiver.
- `x` X coordinate in the custom coordinate projection.
- `y` Y coordinate in the custom coordinate projection.
- `elevation_m` Topographic elevation in meters.
- `dimensions_m` Dimensions of the receiver. rectangle in meters.

⁶<https://www.openstreetmap.org/about>

► Turbine placements

- `placement_id` String identification of the turbine.
- `turbine_type` Make and model of the turbine. Separate `TurbineType` JSON includes more technical information such as cut-off/cut-in speed, rotor diameter etc.
- `latitude` Latitude coordinate of the turbine.
- `longitude` Longitude coordinate of the turbine.
- `x` X coordinate in the custom coordinate projection.
- `y` Y coordinate in the custom coordinate projection.
- `ground_level_m` Topographic elevation in meters.
- `hub_height_m` Height of the turbine tower.

► Shadow cast settings

- `max_shadow_distance_m` Maximum distance (in meters) for which the turbine's shadow is considered.
- `time_delta_minutes` Time step in minutes used in the simulation to calculate shadow movements.
- `shadow_time_thresholds_minutes` An array defining shadow time thresholds in minutes, used for categorizing shadow exposure durations.
- `relevant_days` Number of days considered in the simulation (typically set to 365 for a full year).
- `backtrack_rotor_multiplier` Multiplier applied to turbine rotor diameter.
- `concurrent_process_amount` Number of processes running concurrently to speed up the simulation.
- `consider_topography` Boolean flag indicating whether topography is considered in the shadow simulation.

2.5 Simulation workflow

The following workflow outlines the step-by-step procedure for the wind turbine shadow cast simulation, detailing the data flow from input to final results. This process uses two main approaches: the Forwards Raycasting Method and the Backwards Raycasting Method. We also differentiate between the astronomical maximal case, also known as the worst case and the real weather data case. Figure 5 is also used to display each simulation step and the entire simulation workflow.

1. Import Data JSONs

- The simulation begins by loading essential data from JSON files. These include the turbine specifications, geographic coordinates, project area boundaries.

2. Data Processing

- Once the JSON files are loaded, the data is processed to set up the simulation environment. This step includes configuring the coordinate system, generating the square grid layer for the project area, and preparing the mask for residential building areas.

3. Run Simulation

- The core simulation process begins with the selected parameters and data inputs. Here, the user chooses between two primary simulation models:
 - **Worst Case** simulation assumes the sun is always shining and the turbines are always orthogonal to the sun.
 - **Real Weather Data** such as sun visibility and wind direction is used.

For both simulation models the sun position is computed for the entire period.

4. Forwards Raycasting Method

- Compute the sun's vector for a pair of azimuth and elevation angles.
- Compute the wind turbine's rotor plane normal vector
- Choose n equidistant points on the edge of the rotor circle.
- Shoot a ray through each point on the rotor edge in the direction of the sun vector and intersect it with the ground plane.
- Apply a Point in Polygon Algorithm to compute all cells inside that are shadowed.

5. Backwards Raycasting Method

- Compute the sun's vector for a pair of azimuth and elevation angles.
- Compute the wind turbine's rotor plane normal vector.
- Compute turbine edges.
- Compute bounding box for the cells that will be tested for shadowing.
- Test all cells in the bounding box for shadowing by shooting a ray from the cell in the reverse direction of the sun vector and checking where the intersection point lies on the turbine rotor plane.

6. Compute Building and Receiver Shadow

- For all the shadowed buildings and all receivers calculate the start and end of the shadow.

7. Compute Shadow Contour Lines

- For each yearly shadow duration threshold interval, shadow contour lines are generated to visualize and analyze the shadow shape on the map.

8. Process Results

- A full report including new and existing turbine information, receiver total yearly and daily maximal shadowing and a shadow calendar is generated.

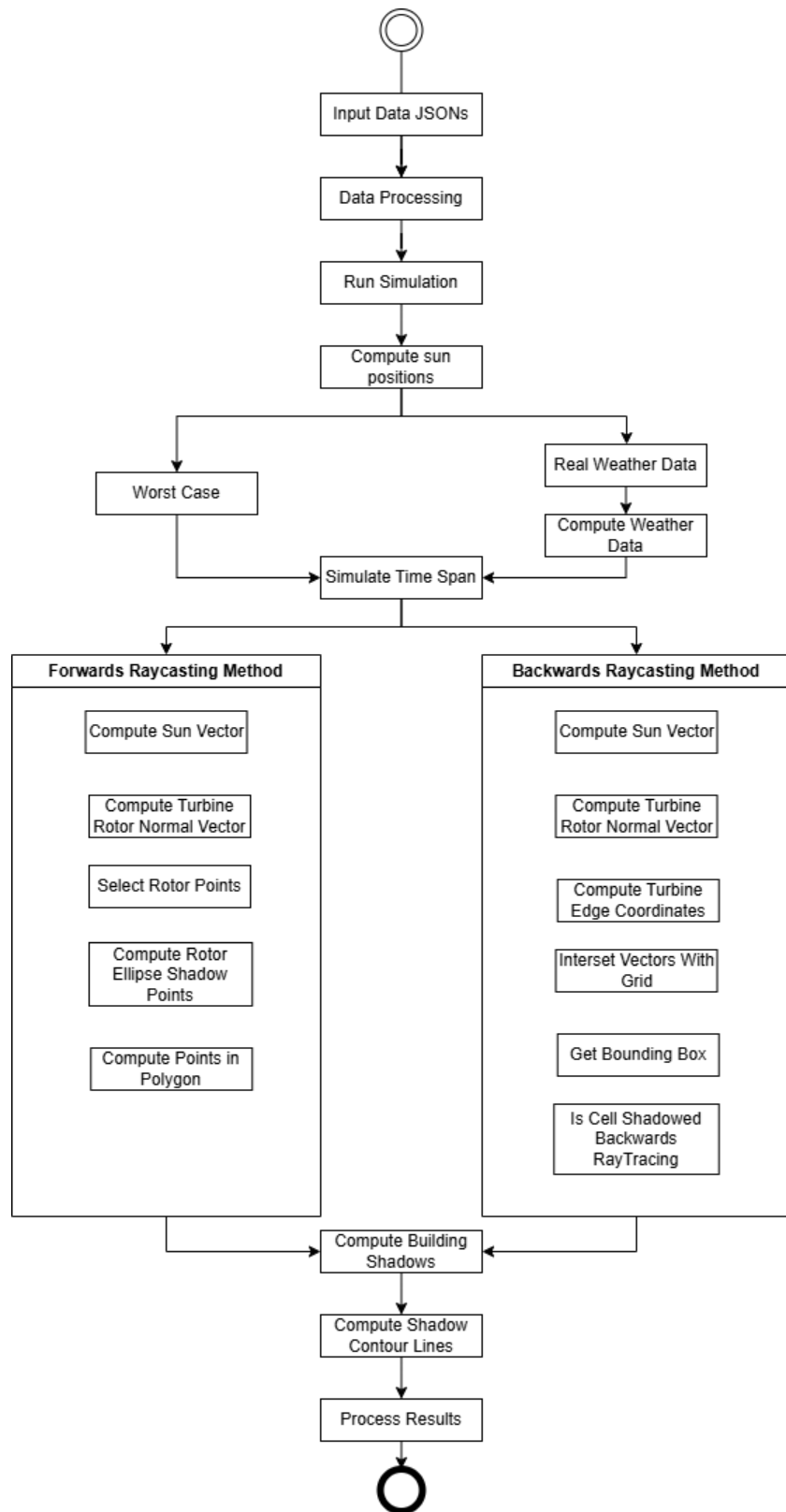


Figure 5: Activity Diagram for the simulation.

3 Modelling the shadow cast of wind turbines

3.1 Astronomical maximal shadow cast

The primary shadow cast calculation model used in this thesis is the worst-case or astronomical maximal model. This model operates under a set of idealized and simplified assumptions designed to estimate the maximum potential shadow impact:

- sun is shining from sunrise to sunset without any atmospheric obstruction from clouds
- turbine rotor is always perpendicular to the sun's rays, thus maximizing the shadow area
- no turbine operational shutdown mechanism

As explained in Section 2, GIS data for the project area is prepared before the simulation starts. By importing necessary data from the input JSON files, a `ShadowCast` object is initialized to set properties such as the square grid, turbine placements (both new and existing), and simulation settings.

3.1.1 Starting the simulation

The simulation begins by using the `SolTrack` package to compute a multi-level `SolTrack` Data Frame containing the sun's azimuth and altitude. Since the shadow loads need to be updated after each iteration, three arrays are defined for different shadow load types: `pre_load`, `additional_load`, and `total_load`. Here, `pre_load` represents the shadows cast by existing turbines, `additional_load` represents shadows cast by new turbines, and `total_load` is the combined effect.

The simulation iterates over the sun position Data Frame using only the `day` index calling the `simulate_single_day()` function, as seen in Figure 6. This function calculates the shadow cast by all turbines for a single day and returns shaded cells as three separate arrays. Finally, these arrays are summed into their respective result arrays on each iteration.

```
1 for day_index, day_d in enumerate(sun_pos.groupby(level='day')):  
2  
3     pre_load, additional_load, total_load = \  
4     simulate_single_day(shadow_obj, day_df)  
5     result_pre_load += pre_load  
6     result_additional_load += additional_load  
7     result_total_load += total_load
```

Figure 6: Simulating the entire year.

In this simulation code, two distinct raycasting methods, forward and backward, can be selected to model shadow casting from wind turbines.

- **Forward Raycasting Method:** This method models shadow casting by projecting rays from the wind turbine toward the ground. It computes which ground cells fall within the shadow polygon created by the turbine at each timestep.
- **Backward Raycasting Method:** Instead of projecting rays outward from the turbine, this method reverses the process by casting rays from each cell in the grid (i.e., potential shadow locations) back toward the sun's position.

The core difference between the methods lies in their approach to raytracing, but both share the same initial steps to set up the sun vector and the rotor plane normal vector.

3.1.2 Sun vector and rotor plane normal vector

Although the sun does not actually emit perfectly parallel light rays, its vast distance from Earth and the limited spatial scale of this simulation allow us to approximate that all rays are parallel. From the azimuth θ and altitude ϕ angles in radians, we can compute the sun vector $\vec{v}_{\text{sun}}$ as follows:

$$\vec{v}_{\text{sun}} = \begin{pmatrix} -\cos(\phi) \cdot \sin(-\theta) \\ -\cos(-\phi) \cdot \cos(\theta) \\ -\sin(\phi) \end{pmatrix}$$

The second vector we need for both raycasting methods is the normal vector of the plane spanned by the turbine rotor $\vec{v}_{\text{rotor}}$, which can be computed by only using the azimuth θ angle:

$$\vec{v}_{\text{rotor}} = \begin{pmatrix} \cos(90^\circ - \theta) \\ \sin(90^\circ - \theta) \\ 0 \end{pmatrix}$$

3.1.3 Forwards raycasting approach

The program now starts simulating one day using the forward raycasting approach. We begin iterating over the sun position Data Frame for a single day and each azimuth and altitude angle pair, we compute the sun vector $\vec{v}_{\text{sun}}$. Because in the astronomical maximal model, all the turbine rotor planes are perpendicular to the sun's light rays, the rotor plane's normal vector $\vec{v}_{\text{rotor}}$ will be the same for all turbines and we can only compute it once. Each turbine is then simulated one by one.

► **Selecting points on rotor edge**

Using the Turbine Placements Object, we can set the center point of the turbine at

$$\mathbf{C} = (\text{turbine.x}, \quad \text{turbine.y}, \quad \text{turbine.ground_height_with_hub_height_m})$$

Then using the computed center point $\mathbf{C}$, the turbine rotor radius $\mathbf{r}$ and the rotor plane normal vector $\vec{v}_{\text{rotor}}$ we compute $\mathbf{n}$ equidistant points on the rotor circle edge as follows:

1. Normalize the normal vector:

$$\hat{N} = \frac{\vec{N}}{|\vec{N}|}$$

2. Determine orthogonal basis vectors $\vec{v}_1$ and $\vec{v}_2$:

$$\text{if } \hat{N} = [0, 0, 1] \text{ or } [0, 0, -1]$$

$$\text{then } \vec{v}_1 = [1, 0, 0]$$

$$\text{else } \vec{v}_1 = \frac{\vec{N} \times [0, 0, 1]}{|\vec{N} \times [0, 0, 1]|} \text{ and } \vec{v}_2 = \vec{N} \times \vec{v}_1$$

3. Compute points on rotor edge

$$\theta = \frac{2\pi}{n}$$

For $i = 0, 1, \dots, n - 1$:

$$P_i = \mathbf{C} + \mathbf{r} \cdot (\cos(i \cdot \theta) \cdot \vec{v}_1 + \sin(i \cdot \theta) \cdot \vec{v}_2)$$

4. Round P_i to the nearest integer coordinates

Because we want to use these equidistant points on the rotor edge to later draw a polygon resembling an ellipse on the ground, the number of points affects the shape of the polygon. From our testing $\mathbf{n} = 10$ is a good number for both accuracy and performance.

Once we have our rotor edge points, we can trace a light ray through each point using the sun vector as seen in Figure 7 and intersect it with the ground to get the polygon shape.

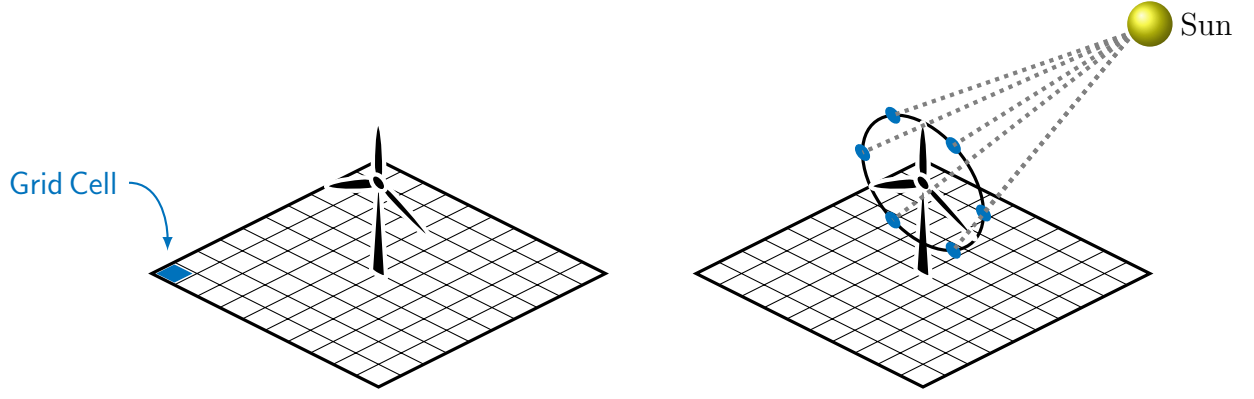


Figure 7: Wind turbine with equidistant rotor points on square cells grid.

► Rotor ellipse shadow points

The objective is to project a point downwards along the direction of the sun vector $\vec{v}_{\text{solar}}$ and find its intersection with the terrain grid. Our custom square grid represents the terrain grid, which includes elevation values for each cell. The approach can be thought of as an adaptive, iterative stepping process that leverages binary search to find the precise intersection point while avoiding redundant calculations. It is a form of ray-casting that dynamically adjusts its step size based on the current distance to the terrain instead of iterating in fixed and uniform steps. The function stops when the intersection point lies outside of the square grid or when the distance between it and the turbine is greater than the `max_shadow_distance` parameter in the shadow cast settings JSON file.

First, the minimal and maximal step sizes are set based on the grid cell size. Then we perform a safety check on the z component of $\vec{v}_{\text{solar}}$ to ensure the vector is pointing upwards. To make calculations easier, we also normalize $\vec{v}_{\text{solar}}$.

Then, the current point to be checked is set to the starting rotor edge point and the initial height difference between the starting rotor edge point z value and the terrain height at the location is computed. A variable to track the last valid point is also set.

The search for the intersection point begins in variable steps along the direction of $\vec{v}_{\text{solar}}$ and checks if the max distance was exceeded in each iteration. For each next point along $\vec{v}_{\text{solar}}$ the algorithm computed, it evaluates the height difference between the point's z coordinate and the terrain height, and halves the step size if the difference sign changes. Once the step size falls below the minimal step size we initially set, the algorithm locates the nearest grid point corresponding to its coordinates as the final intersection point.

► Points in Polygon

Once we have computed the intersection points on the 3D square grid of all rotor points, we obtain a polygon that approximates an ellipse. The accuracy of this ap-

proximation depends on the number of points on the rotor edge (Figure 8). A larger number of points will create a polygon almost perfectly matching an ellipse but at a higher computational cost, so we balance the number of points for both accuracy and efficiency.

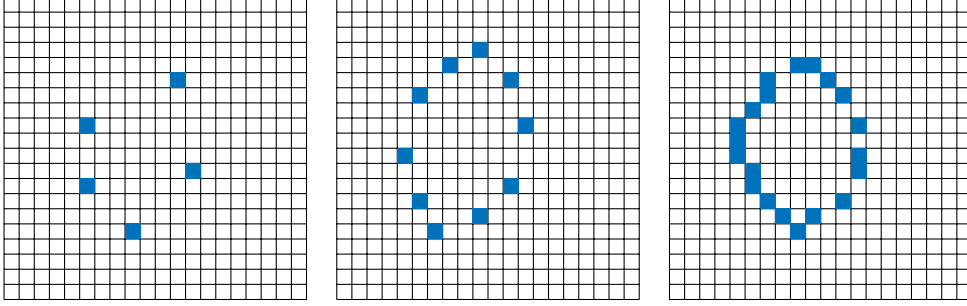


Figure 8: Effect of increasing the number of rotor edge points for shadow approximation. A higher number will produce a shape more closely resembling an ellipse.

The next step is to compute all the cells inside the shadow ellipse. For this, we use a point-in-polygon algorithm based on [13].

Steps in the computational process:

1. **Generate Bounding Box:** To minimize the points checked against the polygon, we first define a bounding box around the projected shadow points. This box provides a limited area of points within which we need to check for inclusion in the polygon, significantly reducing computation.
2. **Point-in-Polygon (PIP) Algorithm:** We use a Point-in-Polygon algorithm to test each point within the bounding box. The algorithm performs a ray-crossing method, where a ray extends from each point horizontally to count intersections with polygon edges. If the number of intersections is odd, the point is inside the polygon; otherwise, it's outside. The PIP function returns a boolean array indicating whether each point lies within the polygon, allowing us to generate a mask for the grid.

Finally, we obtain a set of cells that represent where the shadow of the turbine will be present at a point in time. For each shadowed cell, we add the time step length in each iteration to update the duration of the shadow for that cell.

3.1.4 Backwards raycasting approach

The second raycasting approach explored in this thesis is the backward approach. In this approach, we shoot rays from the grid cells toward the turbine using the sun vector and intersect the rays with the plane spanned by the turbine's rotor.

The first step is the same as for the other raycasting approach. We iterate over the sun position Data Frame for a single day and compute $\vec{v}_{\text{solar}}$ for each azimuth and altitude

pair, as well as the turbine rotor normal for one turbine only since in the astronomical maximal model all turbines face the sun.

Because it is unfeasible to perform backward raycasting for all the cells in the grid, we reduce the set of cells to check for shading by computing a bounding box where we are sure that the turbine rotor shadow will be.

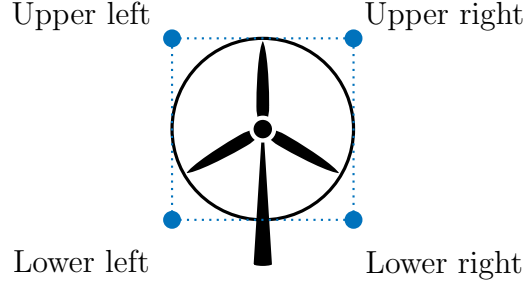


Figure 9: Wind turbine rotor edges. The four corners of the square are used to create a bounding box for the test cells on the ground.

► Turbine rotor edges

To determine the corner coordinates of the square enclosing the rotor circle, we first compute the turbine's left and right edges as seen in Figure 9. From the turbine's x and y coordinates, radius r , and sun azimuth angle θ we can obtain the edges as follows:

$$\begin{aligned} \text{Right Edge: } & \begin{cases} x_{\text{right}} = x + r \sin(\theta - 90^\circ) \\ y_{\text{right}} = y + r \cos(\theta - 90^\circ) \end{cases} \\ \text{Left Edge: } & \begin{cases} x_{\text{left}} = x + r \sin(\theta + 90^\circ) \\ y_{\text{left}} = y + r \cos(\theta + 90^\circ) \end{cases} \end{aligned}$$

Using the cell elevation, hub tower height, and radius we can easily calculate all 4 corners of the square enclosing the rotor circle.

► Bounding box for cell to check for shadowing

Although the cell shadowing test uses backward raycasting, forward raycasting is first applied on the 4 corners of the square enclosing the rotor circle to determine the intersection cells of the 4 points so we can then compute a bounding box. This avoids redundant calculations, by only checking cells in the area where we know for sure the shadow ellipse will be. Determining the intersection cell is done using the same method explained in subsection 3.1.3. Because determining the intersection of a ray with the elevation grid is computationally slow, the bounding box for the test cells is calculated using only the 4 corners of the square in Figure 9.

► **Checking which grid cells inside the bounding box are actually shadowed** For each grid cell in the bounding box, we perform the following algorithm to check if it is shadowed by the turbine's rotor:

1. Intersect Ray with rotor plane

Calculate the dot product of the rotor plane's normal vector and the sun's vector:

$$\text{dot} = \vec{v}_{\text{rotor}} \cdot \vec{v}_{\text{solar}}$$

If $|\text{dot}| < \epsilon$, the line is either parallel to the plane or lies within it, so there is no intersection point. ϵ is a very small threshold value.

Define the vector $\vec{w}$ from the turbine rotor center point to the cell:

$$\vec{w} = \vec{v}_{\text{center}} - \vec{v}_{\text{cell}}$$

Compute the scalar s_i representing the scaled distance along the line's direction to the intersection point:

$$s_i = -\frac{\vec{v}_{\text{rotor}} \cdot \vec{w}}{\text{dot}}$$

Calculate the intersection point as follows:

$$p_{\text{intersection}} = \vec{w} + s_i \cdot \vec{v}_{\text{solar}} + \vec{v}_{\text{cell}}$$

2. Compute the distance between the intersection point and turbine rotor center

For this, we simply compute the norm of the vector $p_{\text{intersection}} - \vec{v}_{\text{center}}$

3. Check if the distance between the rotor center and the intersection point lies inside the rotor.

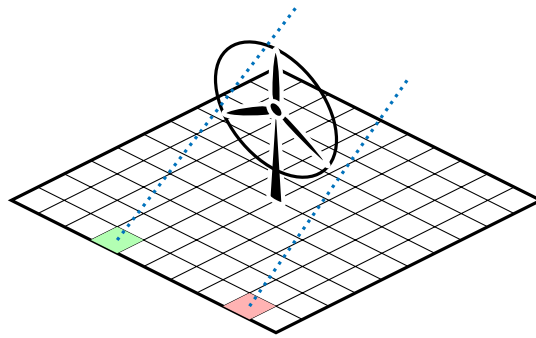


Figure 10: The ray shot from the green cell intersects the rotor plane inside the rotor circle and is thus shaded, while the red cell intersects outside and is thus not shaded.

3.2 Simulation results

The output of the shadow cast simulation can be divided in three main results:

- visual representation of shadowing areas based on thresholds
- shadowing time details for receivers

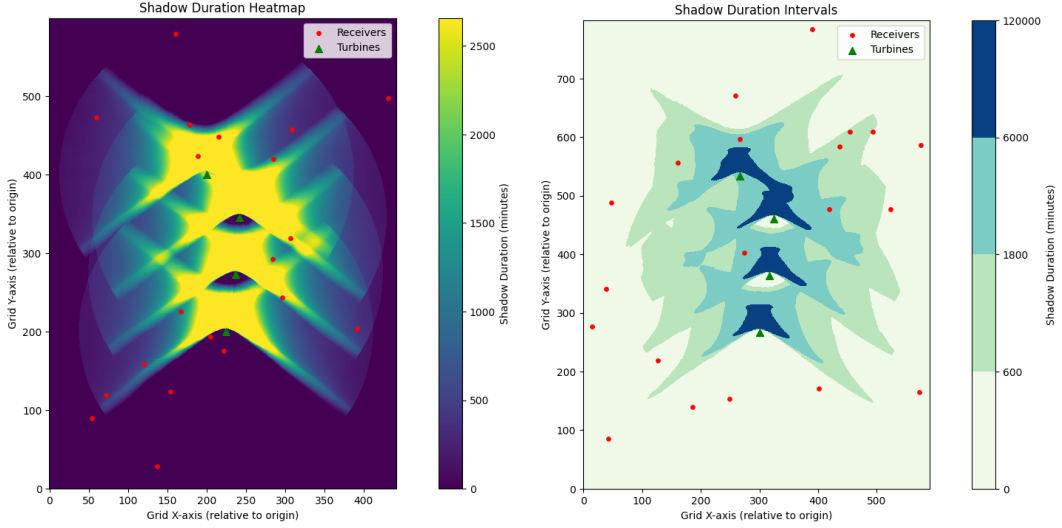


Figure 11: The two plots display the shadow shape caused by 4 turbines in one whole year. The left plot represents a heat map of the shadow duration for each grid cell, normalized around the average value. The right plot shows the different shadow sectors given by 4 intervals. For both plots, the turbines are marked with green triangles while receivers are marked with red dots.

The simulation result contains enough data to generate an in depth report on the total shadow cast. The report includes but is not limited to: daily and yearly shadowing times for each receiver, shadowing caused by existing and new turbines and a shadow calendar detailing which turbine shadows which receiver at a what time step.

ID	Yearly Shadow	Yearly Shadow Days	Max Daily Shadow
R1	7:30	40	0:16
R2	6:56	32	0:17
R3	12:29	39	0:25
R4	4:05	22	0:16
R5	17:22	56	0:28
R6	14:27	55	0:23
R7	7:29	40	0:15

Table 2: Receiver shadow data.

Table 2 depicts the total yearly shadowing hours, maximal shadowing time for one day and the total yearly shadow days for 7 test receivers in a shadow cast simulation. The first and last column in the table are used to check which receiver receives more shadow than the legal limit allows.

3.3 Realistic weather data shadow cast

Besides the astronomical maximal shadowing model, this thesis also includes a second model, which uses real weather data instead of simplified assumptions to simulate the shadow cast by wind turbines.

While the worst-case model is usually the benchmark used by companies or local committees to assess whether a wind turbine can be installed at a given location when accounting for the shadow cast, the realistic weather data model can also provide valuable information that reflects how much shadowing will be present in reality.

Key differences for the realistic weather data model:

- As is the case in reality, the sun does not shine uninterrupted from sunrise to sundown. More often than not, the sun is covered by clouds, and thus a direct shadow will not be cast.
- Instead of always facing perpendicular to the sun, turbines use a motor to turn in the direction of the wind. This allows the turbine to harness the wind power as efficiently as possible.
- Economical and environmental factors dictate when the turbine is actually operational. For this model, we will only consider the wind speed as a factor that influences whether the blades of a turbine are rotating or not. The wind speed has to be between the turbine's specified cut-in and cut-off speed.

► Getting weather data

EnergyPlus⁷ is an open-source whole-building energy modeling engine developed by the US Department of Energy which includes detailed weather data for more than 3000 locations worldwide. We can download weather data for a given geographic location using a simple HTTP request.

Although the weather data downloaded includes a lot of information, we are only interested in direct normal irradiance (DNI), `wind_speed`, and `wind_direction`. Direct normal irradiance is the power per unit area (surface power density) received from the sun in the form of electromagnetic radiation, excluding radiation that is scattered or reflected by atmospheric components. Because we are not interested in the actual numerical value of the DNI, we will consider a boolean flag called `sun_visibility` for DNI values above 120 W/m^2 , which is the lower sun visibility threshold specified in [7].

⁷<https://energyplus.net/weather>

Using these 3 parameters and the `time_delta_minutes` value from the shadow cast settings JSON we can compute a Data Frame displayed in Table 3 with four rows that will hold all the necessary realistic weather data needed for simulating an entire year.

<code>time_step</code>	<code>sun_visibility</code>	<code>wind_direction</code>	<code>wind_speed</code>
0	False	273.0	4.6
1	False	273.0	4.6
2	False	273.0	4.6
3	False	273.0	4.6
4	False	273.0	4.6
$\vdots$	$\vdots$	$\vdots$	$\vdots$
262798	True	20.0	1.5
262799	True	20.0	1.5
262800	True	30.0	1.0
262801	True	30.0	1.0
262802	True	30.0	1.0
$\vdots$	$\vdots$	$\vdots$	$\vdots$
525595	False	270.0	2.6
525596	False	270.0	2.6
525597	False	270.0	2.6
525598	False	270.0	2.6
525599	False	270.0	2.6

Table 3: Weather Data Frame. For each time step, there are 3 entries: whether the sun is visible or not based on the DNI value, wind direction in degrees, and wind speed in m/s .

► Changes to the shadow cast simulation

Several small changes have to be made to the worst-case model to accommodate the real weather data. The results of the shadow maps when accounting for real weather data is depicted in Figure 12

- The first check is done in the function which simulates a single day. Besides skipping time steps where the elevation is below 3° , we also skip time steps where the DNI is too low (`sun_visibility` flag is False)
- We set the turbine azimuth to the `wind_direction` value and use this angle instead of the sun's azimuth to compute the rotor plane's normal vector $\vec{v}_{\text{rotor}}$

- For each time step and each turbine, we verify if the `wind_speed` is between the turbine's cut-in and cut-off speeds. If not, we do not simulate the shadow for the turbine and time step as the blades are not rotating, thus no shadow is cast.
- In the forward raycasting approach, we use the turbine's azimuth instead of the sun's azimuth to compute the equidistant rotor edge points.
- Similarly, in the backward raycasting approach, the turbine's azimuth is used to determine the left and right edges.

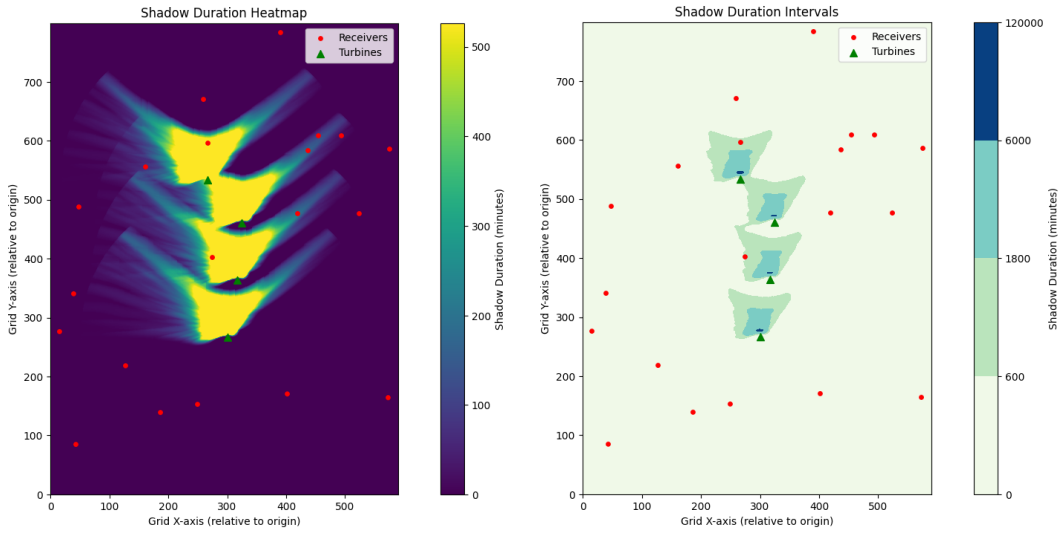


Figure 12: The shadow cast by four turbines was calculated using a realistic weather data model. The left figure shows a heat map of the shadow cells, normalized relative to the average value, while the right figure illustrates the shadow time intervals. For the selected test site in western Germany, only 166,500 out of the total 525,600 time steps (approximately 31%) have a DNI value exceeding 120 W/m^2 and are thus considered for the calculation.

3.4 Turbine shutdown

The shutdown module computes a yearly shutdown calendar which indicates when each wind turbine has to be momentarily turned off in order to comply with the regulations. According to the reference document [7] the limits for shadowing of residential buildings are:

- 30 minutes per day
- 30 hours per year

Computing the shutdown calendar is done similarly to the regular shadowing simulation, for either the real weather data case or worst case scenario. We employ the same raycasting approaches to determine where the shadow will be at each time step but in addition, we save a list of tuples that showcases the time step at which each building was shadowed and the turbine responsible for the shadowing

The list of tuples containing instances of shadowing and turbine IDs is then further processed by checking which buildings exceed the daily or yearly shadowing limit. Finally, the shutdown calendar for each turbine is computed.

Turbine ID	Shutdown at time steps
0	–
1	–
2	8890
3	10129, 10992
4	–
5	–
6	–
7	636, 644, 645, 1076, 1500, 1508, 1651, 1652, 1653, 2218, ...
8	–
9	11567, 11855, 14443, 15163, 15307, 32440, 33448, 33451, 34315, 34891, ...
10	6534, 6535, 10128, 10272, 10276, 10416, 10419, 10420, 10851, 10995, ...
11	62, 63, 64, 637, 638, 639, 640, 641, 1070, 1071, ...
12	1354, 1498, 2218, 45269, 45413, 48008
13	9898, 10186, 10330, 10474, 51788
14	658, 1667, 2666, 2810, 6140, 9896, 21571, 21713, 21714, 21715, ...
15	92, 93, 636, 637, 638, 639, 642, 643, 649, 650, ...
16	644, 645, 654, 656, 658, 659, 660, 1075, 1076, 1508, ...

Table 4: Shadow shutdown times per turbine. The table showcases the structure of the shutdown calendar for 17 turbines in the worst-case simulation. For example, the turbine with ID 3 has to be turned off twice, at time steps 10129 and 10992 for the duration of the time step length defined in the Shadow Cast Settings JSON file.

The shutdown calendar computed in this simulation is a key tool for managing compliance with shadowing regulations for wind turbine farms located near residential buildings. By providing specific times for each turbine to be shut down, this calendar helps ensure that the cumulative shadow cast does not exceed the legal limits set to protect nearby residents from excessive shadowing effects. Besides helping with regulatory compliance, the shutdown calendar can be a valuable tool for other types of wind farm simulations and analyses:

- **Energy Production Simulation:** By incorporating shutdown times, energy production models can more accurately estimate the actual output of a wind farm, taking into account operational downtimes required to comply with shadow flicker regulations.
- **Grid Supply & Demand Management:** By integrating the shutdown calendar, grid operators can better predict supply variations and plan for backup power when turbines are offline, thus improving grid stability.
- **Turbine Maintenance:** Turbine maintenance work can be coordinated with shutdown times, providing the financial benefit of not having to stop the wind turbine more than needed.

4 Results verification

WindPRO by EMD International is a popular software suite for designing and planning wind farm projects. The program covers everything from wind data analysis, calculating energy yields, quantifying uncertainties, and assessing site suitability to calculating and visualizing the environmental impact of wind turbines. The WindPRO SHADOW calculation module makes it possible to compute the shadowing duration for user-defined receivers or shadow maps for the area surrounding the wind turbine park. The output of the shadow cast simulation implemented in this thesis, including the shadowing maps, receivers' shadow times, and shadow calendars was configured to closely resemble the report calculated by the SHADOW module. For the validation in this section, we will compare the total shadowing time, the number of shadowed days, and the maximal daily shadow time for receivers. Furthermore, the shadow map contours will also be evaluated. All validation metrics will be computed for the worst-case model as defined in the reference document described in subsection 2.1.

4.1 Importing custom projects via the API

Because both the WindPRO SHADOW module and the program implemented in this thesis require a lot of input data and parameters, manually importing the set-up data into WindPRO proved to be laborious. Luckily, WindPRO provides a custom API that allows setting up a project directly via a Python script. Using the project area boundaries, turbine and receiver details, a WindPRO project can be automatically created to match the parameters used in our simulation, allowing for a very easy and fast testing interface.

4.2 Testing scenario

For this validation scenario, 4 different wind turbine models were placed in the vicinity of Herzogenrath, Germany. Using the turbines' locations and the max shadow distance, a bounding box was computed around the 4 turbines where 20 receivers were placed at random locations. Turbine and receiver details can be found in Table 8 and 9 in the Appendix. For both programs, a whole year in 1 minute time steps was simulated. Our simulation used an 6-meter resolution topographic model while WindPRO used a resampled 5-meter resolution model⁸.

⁸https://help.emd.dk/mediawiki/index.php?title=German_NRW_Elevation_Model

4.3 Results

► Receiver results

Tables 5 and 6 present the results for 20 randomly placed receiver locations, comparing the outputs of WindPRO and the simulation implemented in this thesis using the backward raycasting method. The analysis demonstrates the accuracy of the simulation, with an average error of 0.5355%, a minimal error of 0%, and a maximum error of 5.3097% for the yearly shadow hours metric. For the total number of shadowing days, the maximum discrepancy is 3 days, while for the maximum daily shadow duration, the difference does not exceed 1 minute. These results validate the backward raycasting method as a highly accurate approach for simulating shadow cast. Small errors in the backward raycasting method can occur due to slight differences in the calculated position of the sun. Even small differences of a couple degrees in solar azimuth or altitude angles, can lead to shifts in shadow contours. These minor differences, when compounded over tens of thousands of rays cast during the simulation, may result in small deviations in shadow duration, especially for receivers located near the edge of shadowed areas.

The forward raycasting approach was also tested on the same scenario but yielded significantly worse results, with an average error of 6.6258% for the yearly shadow hours metric. This higher error rate is primarily due to the inherent limitations of the forward raycasting method in this context given by the differences in the topographic models used by the two programs. In forward raycasting, shadows are calculated by projecting rays from the turbine rotor onto the terrain, making the accuracy highly dependent to the resolution and precision of the topographic model. A few meters of difference in elevation can significantly alter the intersection points of rays with the terrain usually leading to an under estimation in shadow durations. The forward raycasting approach is less suitable for scenarios requiring precise shadow calculations for individual receivers, as it prioritizes grid-based shading over specific point-based validations. As a result, while forward raycasting may prove useful in some cases, its limitations make it less reliable for applications requiring detailed, receiver-specific shadow assessments compared to the backward raycasting approach.

Receiver	Metric	Simulation	WindPRO	Difference
Receiver 1	Yearly Shadow (hh:mm/yr)	07:17	07:17	0%
	Shadow Days (days/yr)	39	39	0 days
	Max Daily Shadow (hh:mm/day)	0:15	0:15	0.00 min
Receiver 2	Yearly Shadow(hh:mm/yr)	44:25	44:22	0.112697%
	Shadow Days (days/yr)	134	136	2 days
	Max Daily Shadow (hh:mm/day)	0:31	0:31	0 min
Receiver 3	Yearly Shadow(hh:mm/yr)	00:00	00:00	0%
	Shadow Days (days/yr)	0	0	0 days
	Max Daily Shadow (hh:mm/day)	0:00	0:00	0 min
Receiver 4	Yearly Shadow(hh:mm/yr)	13:11	13:10	0.126582%
	Shadow Days (days/yr)	60	60	0 days
	Max Daily Shadow (hh:mm/day)	0:16	0:17	1 min%
Receiver 5	Yearly Shadow(hh:mm/yr)	00:00	00:00	0%
	Shadow Days (days/yr)	0	0	0 days
	Max Daily Shadow (hh:mm/day)	0:00	0:00	0 min
Receiver 6	Yearly Shadow(hh:mm/yr)	09:55	09:25	5.309735%
	Shadow Days (days/yr)	56	53	3 days
	Max Daily Shadow (hh:mm/day)	0:15	0:15	0 min%
Receiver 7	Yearly Shadow(hh:mm/yr)	00:00	00:00	0%
	Shadow Days (days/yr)	0	0	0 days
	Max Daily Shadow (hh:mm/day)	0:00	0:00	0 min
Receiver 8	Yearly Shadow(hh:mm/yr)	00:00	00:00	0%
	Shadow Days (days/yr)	0	0	0 days
	Max Daily Shadow (hh:mm/day)	0:00	0:00	0 min
Receiver 9	Yearly Shadow(hh:mm/yr)	45:44	45:44	0%
	Shadow Days (days/yr)	70	70	0 days
	Max Daily Shadow (hh:mm/day)	0:51	0:51	0 min
Receiver 10	Yearly Shadow(hh:mm/yr)	00:00	00:00	0%
	Shadow Days (days/yr)	0	0	0 days
	Max Daily Shadow (hh:mm/day)	0:00	0:00	0 min
Receiver 11	Yearly Shadow(hh:mm/yr)	00:00	00:00	0%
	Shadow Days (days/yr)	0	0	0 days
	Max Daily Shadow (hh:mm/day)	0:00	0:00	0 min

Table 5: Comparison of our simulation and WindPRO for shadow load metrics at receivers.

Receiver	Metric	Simulation	WindPRO	Difference
Receiver 12	Yearly Shadow(hh:mm/yr)	00:00	00:00	0%
	Shadow Days (days/yr)	0	0	0 days
	Max Daily Shadow (hh:mm/day)	0:00	0:00	0 min
Receiver 13	Yearly Shadow(hh:mm/yr)	17:52	18:05	1.198157%
	Shadow Days (days/yr)	60	59	1 day
	Max Daily Shadow (hh:mm/day)	0:28	0:28	0 min
Receiver 14	Yearly Shadow(hh:mm/yr)	05:06	05:03	0.990099%
	Shadow Days (days/yr)	24	24	0 days
	Max Daily Shadow (hh:mm/day)	0:16	0:17	1 min
Receiver 15	Yearly Shadow(hh:mm/yr)	03:10	03:10	1.198157%
	Shadow Days (days/yr)	18	19	1 day
	Max Daily Shadow (hh:mm/day)	0:13	0:14	1 min
Receiver 16	Yearly Shadow(hh:mm/yr)	00:00	00:00	0%
	Shadow Days (days/yr)	0	0	0 days
	Max Daily Shadow (hh:mm/day)	0:00	0:00	0 min
Receiver 17	Yearly Shadow(hh:mm/yr)	21:35	21:37	0.154202%
	Shadow Days (days/yr)	80	79	1 day
	Max Daily Shadow (hh:mm/day)	0:20	0:20	0 min
Receiver 18	Yearly Shadow(hh:mm/yr)	50:00	51:27	2.818270%
	Shadow Days (days/yr)	72	71	1 day
	Max Daily Shadow (hh:mm/day)	0:51	0:52	1 min
Receiver 19	Yearly Shadow(hh:mm/yr)	00:00	00:00	0%
	Shadow Days (days/yr)	0	0	0 days
	Max Daily Shadow (hh:mm/day)	0:00	0:00	0 min
Receiver 20	Yearly Shadow(hh:mm/yr)	00:00	00:00	0%
	Shadow Days (days/yr)	0	0	0 days
	Max Daily Shadow (hh:mm/day)	0:00	0:00	0 min

Table 6: Continuation of Table 5.

► Shadow maps

For the shadow maps comparison, contour lines for four shadowing duration intervals were generated using a grid resolution of 6 meters in both WindPRO and the our simulation. Since WindPRO does not support the same elevation model utilized by our simulation, both shadow maps were calculated assuming a flat terrain. This approach ensures that the comparison focuses solely on validating the accuracy of the computed shadow maps, rather than introducing discrepancies caused by differences in elevation data between the two digital elevation models.



Figure 13: Shadow maps for the testing scenario. The four wind turbines are marked with a green pin while the receivers are marked with an orange dot.

To compute shadow map contours, the shadowing time for each grid cell is assessed in combination with intervals configured in the Shadow Cast Settings JSON. Using these intervals, contour lines for each shadowing region can be calculated. Skikit-image⁹ includes the `find_contours()` function which is used to find the list of coordinates for the shadow contour lines. Finally, the coordinates are converted from our custom coordinate system to latitude and longitude in order to display the shadows on a map properly.

Three images are compared in Figure 13. The top-left image shows the shadow map generated by the simulation developed in this thesis, while the top-right image displays the shadow map created by WindPRO. For clarity, dotted contour lines were used for the WindPRO shadow map. The bottom image overlays both shadow maps, allowing for a direct comparison of the results. The shadow simulation in this thesis achieves an almost perfect match to the shadow contours computed by WindPRO, demonstrating the simulation's high level of accuracy.

⁹<https://scikit-image.org/>

► Accuracy drop for coarser grid resolution

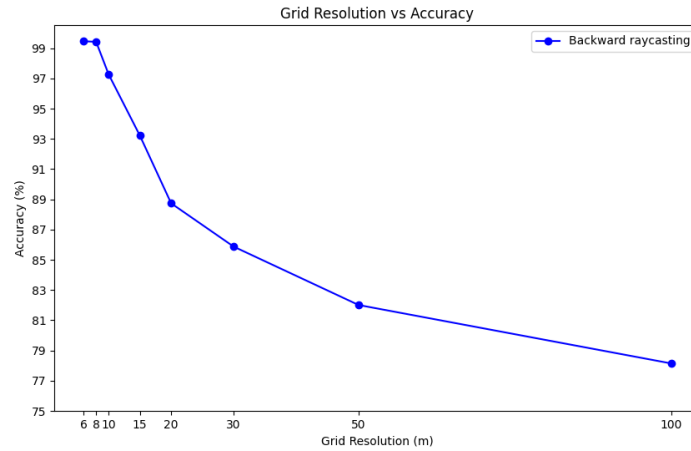


Figure 14: Effect of different grid resolutions step size on the simulation accuracy when using backwards raycasting.

Plot 14 showcases the relationship between grid resolution and simulation accuracy when using the backward raycasting method. The x-axis represents the grid resolution in meters, while the y-axis represents the simulation accuracy as a percentage. The graph shows that as the grid resolution becomes coarser (larger grid step sizes), the simulation accuracy decreases steadily. It is evident from the plot that grid resolutions of 6, 8, and 10 meters are fine enough to provide an accuracy above 95%, making them suitable for simulations where high precision is required. However, beyond these finer resolutions, accuracy drops significantly, emphasizing the importance of selecting an appropriate resolution to balance computational efficiency and accuracy.

5 Profiling and execution time

One of the primary objectives of this thesis was to develop a high-performance simulation capable of handling very small time steps, down to one-minute intervals. Achieving this efficiency level proved challenging, as simulating a full year at a one-minute resolution requires processing 525,600 individual time steps for each turbine. Additionally, depending on the size of the project and the resolution of the grid cells, there could be several million cells subject to potential shading calculations. Combined, these factors created a computationally demanding simulation environment.

5.1 Profiling

In software engineering, profiling is the process of gathering and analyzing different metrics of a program to improve its run time and memory usage. In simpler terms, using a profiling module we can analyze our simulation in detail and find out how many times each function has been called and how much time has been spent on each function.

We used the C extension of the pure Python profiling module called CProfile¹⁰ to analyze the runtimes and visualize them using SnakeViz¹¹. For simplicity reasons, we chose 10-minute time intervals and a single turbine for analyzing function runtime. Below are the SnakeViz icicle visualizations for the forward and backward raycasting methods.

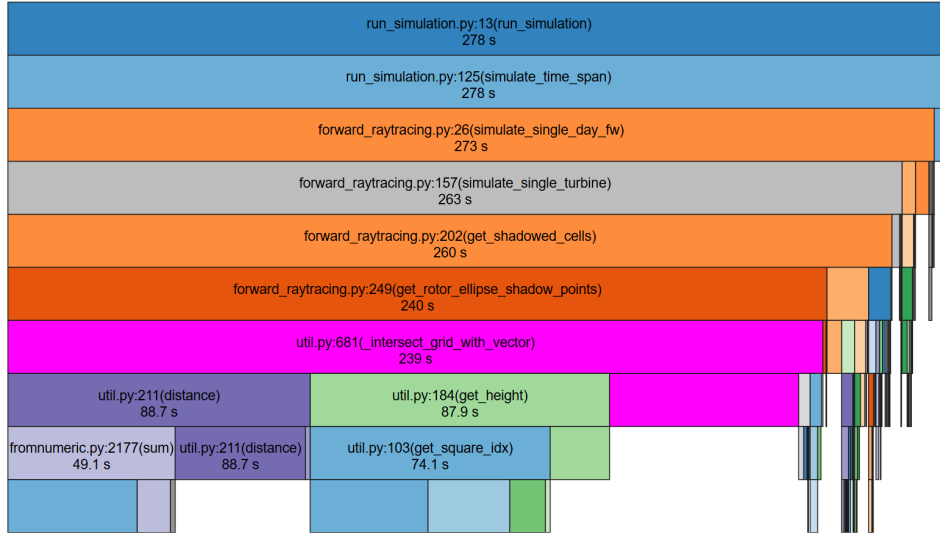


Figure 15: SnakeViz icicle visualization for forward raycasting.

Figure 15 displays each sub-function of the simulation with its cumulative run time. It is not surprising that around 85% of the total run time is spent on the function which computes the shadow ellipse points. What is surprising is that most time is spent on computing the intersection point of the shadow ray from the rotor edge points. With

¹⁰cProfile, URL <https://docs.python.org/3/library/profile.html>

¹¹SnakeViz, URL <https://jiffyclub.github.io/snakeviz/>

a per-call time of 0,000111 seconds, this function represents a potential performance bottleneck in the simulation.

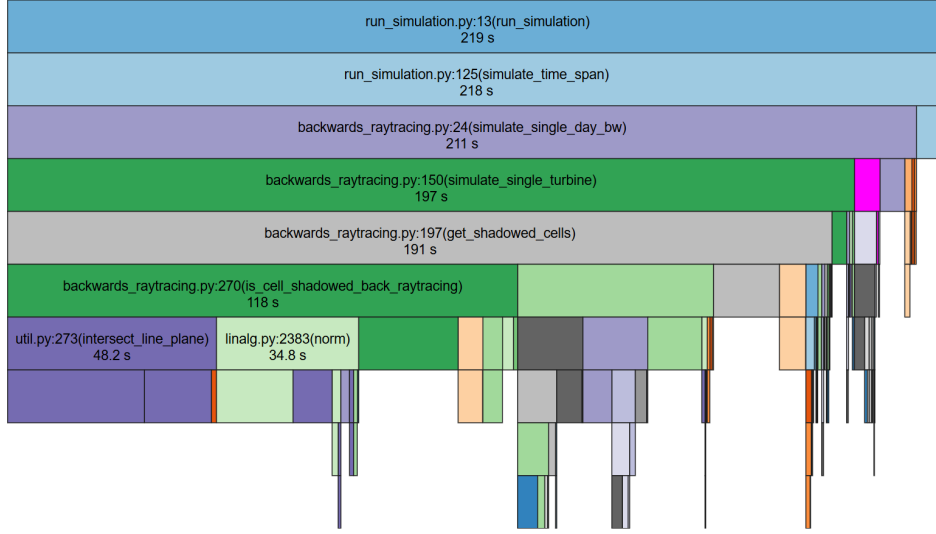


Figure 16: SnakeViz icicle visualization for backward raycasting.

Figure 16 displays all sub-functions in the backward raycasting approach and their respective cumulative run time. As expected, the function that tests the cells in the bounding box for shadowing takes up the most time. As in the other raycasting approach, the grid intersection function slows things down despite only being called four times per time step per turbine.

5.2 Process parallelization

One approach used in our quest to speed up the simulation time was running the simulation on multiple parallel processes.

We start by splitting the simulation period into almost equal chunks. The goal is to have n processes running in parallel and each process simulating approximately $\frac{365}{n}$ days. One important aspect to note here is that when splitting the `sun_positions` DataFrame into chunks, we make sure that rows corresponding to the same day get assigned to the same chunk.

The standard Python `ProcessPoolExecutor`¹² class offers easy-to-use pools of pools of child worker processes ideal for parallelizing loops of CPU-bound tasks which is exactly what is needed for our simulation.

¹²<https://docs.python.org/3/library/concurrent.futures.html>

```

1 from concurrent.futures import ProcessPoolExecutor, as_completed
2
3
4 with ProcessPoolExecutor(max_workers=nr_chunks) as executor:
5
6     futures = []
7
8     for i in range(nr_chunks):
9         future = executor.submit(simulate_chunk)
10        future.add_done_callback(callback)
11        futures.append(future)
12
13    for future in as_completed(futures):
14        result = future.result()
15        additional_load += result[0]
16        additional_load_with_shutdown += result[1]
17        pre_load += result[2]
18        pre_load_with_shutdown += result[3]
19        total_load += result[4]
20        total_load_with_shutdown += result[5]
21        building_shadows += result[6]

```

Figure 17: Submitting and executing parallel processes. The results are updates in the order in which processes finish, thus no time is wasted on waiting for unfinished processes.

As seen in the code listing in Figure 17 we first initialize a `ProcessPoolExecutor` with the desired number of chunks/parallel processes. A custom partial function callback is also added to each process, logging the progress for each individual process. We then start the processes, each simulating a chunk of the entire simulation period. Because processes do not necessarily finish in the same order they were started, we update the shadow arrays and other results in the order in which the processes finish, thus no time is wasted waiting for unfinished processes.

5.3 Runtime

For the runtime analysis, the simulation for each parameter configuration was run 10 times and the result was averaged. The machine used for testing was a Windows Laptop with an Intel I5 1340p CPU (12 Cores, 16 Threads, Clock Speed up to 4.8 GHz) and 16 GB RAM.

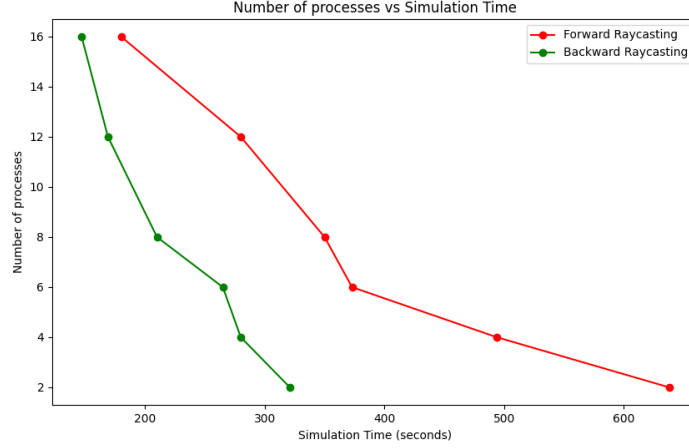


Figure 18: Effect of number of parallel processes on simulation time using 10 meter grid resolution and 1-minute time steps.

As seen in Figure 18, distributing the computation on several processes significantly improves the simulation time. With 16 processes computing about 21 days each, simulating 1 turbine using the faster backward approach takes less than 2 minutes. The number of processes also plays a role in the simulation time. On the machine used for testing, there are 16 total processor threads which can each run a process instance.

5.4 Results

Time Improvement	Method	
	Forwards Raycsting	Backwards Raycasting
Absolute (s)	715.37 s	345.79 s
Percentage	296.82%	336.89%

Table 7: Improvement times with parallelization for both raytracing methods.

The runtime improvements achieved through process parallelization using 12 processes for simulating a single turbine are illustrated in Table 7. The percentage improvement indicates that both raycasting methods experience approximately threefold reductions in runtime. As seen from the comparison of run times for different numbers of processes, there is a cap on the improvement a couple of extra processes offer. To further speed up the simulation, just-in-time Python compilation with Numba¹³ and NumPy¹⁴ vectorized functions can be used. Due to the complex codebase and use of many unsupported custom classes and data types, this proved to be challenging.

¹³<https://numba.pydata.org/>

¹⁴<https://numpy.org/>

Conclusion

Wind turbine planning is a very complex process requiring careful considerations of economic, technical, legal, and environmental factors. Among the environmental impacts of wind turbines is the shadow flicker effect, caused by the turbine blades intermittently obstructing sunlight and casting moving shadows on nearby areas. Germany, among a few other European countries, enforces strict regulations limiting the total annual and daily shadow duration on residential buildings. These regulations, along with the necessary models and assumptions for accurate shadow cast assessments, were central to the development of the simulation presented in this thesis.

In this thesis, a performant, accurate, and highly customizable Python-based shadow cast simulation tool was created, adhering closely to German guidelines for wind turbine shadow assessments. The simulation incorporates two shadowing models: one using simplified assumptions about sun visibility and turbine orientation to estimate the worst-case shadowing scenario, and another that accounts for real-world factors such as sun visibility, wind direction, and wind speed.

The simulation employs two distinct raycasting methods, one of which demonstrated very good accuracy when compared to the industry-standard WindPRO tool. Additionally, this method was up to 5 times faster than WindPRO when run under the same settings, making it a highly efficient alternative for shadow cast assessments. The enhanced performance was achieved by parallelizing computations across multiple processors.

Thanks to its flexible configuration options and detailed output capabilities, this simulation tool is well-suited for a variety of wind farm assessment scenarios. It provides detailed shadow information for specific receivers, visual shadow maps for customizable time intervals, and turbine shutdown calendars. These features make it a valuable tool for professionals and for community members living near wind turbines, offering clear insights into the potential shadowing effects of planned turbines.

5.1 Future work

For the development of this simulation tool, there were two main challenges encountered.

Firstly, handling high-resolution elevation data models with resolutions as fine as a few meters proved to be computationally demanding. For example, a project area of 10 km x 10 km can easily consist of several million cells, each with its own elevation value, which can quickly overwhelm memory resources. A potential improvement for the current tool could involve more efficient elevation data handling, such as only storing the elevation data needed for a specific time step in the simulation.

Secondly, simulating an entire year using 1-minute time steps requires 525,600 computational steps per turbine. Since runtime increases linearly with the number of turbines, simulating multiple turbines can become very time-intensive, even with parallel processing. One way to address this issue is by interpolating shadow values at intervals instead of calculating them for every time step. Interpolation could be applied to both receiver-specific shadow details and shadow maps, potentially saving significant runtime while

maintaining acceptable accuracy. Depending on the use case, a faster but slightly less accurate simulation that uses interpolation could be highly beneficial.

In addition to the areas where computational efficiency could be improved, further features could enhance the tool. Currently, receivers are defined as flat rectangles, with only their elevation above the ground and side dimensions adjustable. For even greater accuracy, additional characteristics, such as tilt angles or non-rectangular shapes, could be introduced. Furthermore, the visualization of shadow maps could be enhanced by incorporating a 3D rendering of the entire area, including the topographic model and the colored shadow map.

Appendix

Turbine	Latitude	Longitude	Rotor Diameter (m)	Z level (m)	Hub(m)
W1	50.8314	6.0093	80	172.1	100
W2	50.8367	6.0107	80	180.2	100
W3	50.8419	6.0113	80	175.4	100
W4	50.8458	6.0064	80	178.5	100

Table 8: Turbine Coordinates with Rotor Diameter, Ground Level, and Hub Height.

Receiver	Latitude	Longitude
R0	50.8428	6.0283
R1	50.8428	6.0194
R2	50.8260	6.0325
R3	50.8499	6.0224
R4	50.8532	6.0058
R5	50.8499	6.0257
R6	50.8487	6.0327
R7	50.8216	5.9874
R8	50.8388	6.0070
R9	50.8253	6.0049
R10	50.8593	6.0169
R11	50.8246	5.9995
R12	50.8470	5.9974
R13	50.8288	5.9945
R14	50.8434	5.9877
R15	50.8355	5.9870
R16	50.8486	6.0209
R17	50.8492	6.0063
R18	50.8320	5.9850
R19	50.8262	6.0178

Table 9: Receiver Coordinates.

References

- [1] Matthes, F. C. “Energy transition in Germany: a case study on a policy-driven structural change of the energy system”. In: *Evolutionary and Institutional Economics Review* 14.1 (2017), pp. 141–169.
- [2] Sahu, B. K., Hiloidhari, M., and Baruah, D. “Global trend in wind power with special focus on the top five wind power producing countries”. In: *Renewable and Sustainable Energy Reviews* 19 (2013), pp. 348–359.
- [3] Statistisches Bundesamt. *Gross electricity production in Germany*. <https://www.destatis.de/EN/Themes/Economic-Sectors-Enterprises/Energy/Production/Tables/gross-electricity-production.html>. Accessed: 2024-09-23.
- [4] Lintz, G. and Leibenath, M. “The politics of energy landscapes: the influence of local anti-wind initiatives on state policies in Saxony, Germany”. In: *Energy, Sustainability and Society* 10.1 (2020).
- [5] BMJ. *Gesetz zum Schutz vor schädlichen Umwelteinwirkungen durch Luftverunreinigungen, Geräusche, Erschütterungen und ähnliche Vorgänge*.
- [6] Bolin, K. et al. “An investigation of the influence of the refractive shadow zone on wind turbine noise”. In: *The Journal of the Acoustical Society of America* 148.2 (2020), EL166–EL171.
- [7] Immissionsschutz, L. für. “Hinweise zur Ermittlung und Beurteilung der optischen Immissionen von Windenergieanlagen”. In: *Beschlüsse der 103. LAI-Sitzung* (2002).
- [8] Reda, I. and Andreas, A. “Solar position algorithm for solar radiation applications”. In: *Solar energy* 76.5 (2004), pp. 577–589.
- [9] Sluys, M. van der and Kan, P. van. “SolTrack: a free, fast and accurate routine to compute the position of the Sun”. In: *arXiv preprint arXiv:2209.01557* (2022).
- [10] Energyfollower.com. *Wind Turbine Shadow Flicker: What Causes This Phenomenon*.
- [11] Racz, F. T. B. “Wind Farm Layout Optimization near Urban Areas Using Approximated Shadow Flicker and Noise Emissions”. In: (2023). Bachelor Thesis FH Aachen.
- [12] Ibraheem, A. and Daham, A. “Development and Use of Large-Scale Land Information System (LIS) by Using Geographic Information System (GIS) and Field Surveying”. In: *Iraqi Journal of Civil Engineering* 8 (2011), pp. 29–43.
- [13] Schwinger, E. et al. “Point in polygon calculation using vector geometric methods with application to geospatial data”. In: *arXiv preprint arXiv:2306.04316* (2023).