

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

Improving the Scalability of a Continuous Approach for Shadow Flicker Prediction

Bachelor of Science Thesis

September 13, 2025

Presented by	Erik Schwarz matriculation number: 422557 erik.schwarz@rwth-aachen.de
First examiner	Prof. Dr. Dr. h. c. Erika Ábrahám Lehr- und Forschungsgebiet: Theorie der hybriden Systeme RWTH Aachen University
Second examiner	Prof. Dr.-Ing. Georg Jacobs Lehrstuhl und Institut für Maschinenelemente und Systementwicklung RWTH Aachen University
Supervisor	Prof. Dr. Dr. h. c. Erika Ábrahám Lehr- und Forschungsgebiet: Theorie der hybriden Systeme RWTH Aachen University
Advisor	Nicolai Radke, M.Sc. Lehr- und Forschungsgebiet: Theorie der hybriden Systeme RWTH Aachen University

Abstract

Shadow flicker from wind turbines is typically estimated with discrete-time methods that either become computationally expensive (for small time steps) or cause large time-step discretization errors for large time steps.

This thesis builds on a method that uses a continuous function to model shadow flicker and avoids time-step discretization. We extended the existing approach and its implementation and improved them in various ways. First, we replaced the root-finding algorithm, reused shadow intervals from the preceding day as initial guesses for a subsequent day, used an early-stopping criterion to stop minimization without losing accuracy, and applied a heuristic to reduce the computational effort on a grid. To ensure scalability of the approach, we replaced the static data structure for storing the results with a dynamic one. This means that instead of allocating memory for every day, we allocate memory only for days on which shadow flicker occurs, reducing memory consumption.

For the second part of the thesis, we increased usability by adding support for calculating shadow flicker occurrences for multiple turbines. We also added support for using geographic coordinates instead of turbine-centered coordinates.

Further, we precomputed shadow maps, which are grids where we calculate the shadow flicker durations at grid points spaced at a defined step size for various heights. Using these, we can then look up the shadow flicker duration for receivers inside the grid and use interpolation to provide durations for an arbitrary height within a user-defined interval.

Acknowledgments

I would like to thank Prof. Erika Ábrahám for supervising this thesis and for her academic guidance during the course of the work.

I also thank Prof. Dr.-Ing. Georg Jacobs for serving as the second examiner.

Special thanks to my advisor, Nicolai Radke, for his extensive support, for providing material and resources, and for connecting me with Patrick De Smet and the team at CWD. This thesis is based on a joint publication by Patrick De Smet and Nicolai Radke. I thank both for sharing their implementation, answering my questions, and helping me get started with the topic. I also thank the team at CWD, in particular Lucas Blickwedel, for the opportunity to integrate my work into a real-world project, the WiFO. Lastly, I thank my family and friends for their support and motivation.

Contents

Abstract	3
Acknowledgments	4
1 Introduction	6
Nomenclature	9
2 Preliminaries	1
2.1 Problem Statement	1
2.2 Main Concept	1
2.3 Solar Position Algorithm	2
2.4 Ray Tracing Algorithm	3
2.5 Distance Function	4
2.6 Shadow Flicker Computation	5
2.7 Correctness and Validation	12
3 Performance Improvements to the Novel Approach	14
3.1 Technical Improvements	14
3.2 Root Finding	14
3.3 Reusing the Prior Day's Guess	15
3.4 Reusing the Midpoint of Prior Day's Interval	16
3.5 Early Termination of Minimization Subroutine	16
3.6 Relationship Between Features	17
3.7 Flowchart of the Algorithm with Optimizations	17
3.8 Reducing the Computational Effort	19
3.9 Heuristic - Grid Property	19
3.10 Correctness	22
3.11 Validation	22
4 Memory Usage Optimization to the Implementation	30
5 Precomputation	37
5.1 Wind Farm Layout Optimization	37
5.2 Precomputing using Shadow Maps	37
5.3 Methodology	38
5.4 Interpolation	38
5.5 Correctness and Validation	39
6 Conclusion	41
7 Future Work	42
Bibliography	43

1 Introduction

As the need for green energy becomes more and more prevalent, institutions like the European Investment Bank [1] advance and support many projects in the green energy sector. The two biggest renewable energy sources in Europe are solar energy and wind energy, as they are scalable and commonly available resources that have a lot less environmental impact than fossil fuels. Europe currently operates 285 GW of installed wind capacity. The production for 248 GW is placed onshore, and 37 GW is placed offshore. Current turbines can produce up to 15 MW and still grow in size and energy production. The capacity is expected to increase by 187 GW between 2025 and 2030. At least 42.5% of energy consumption must come from renewable sources, according to a goal set by the EU. In order to achieve this goal, 425 GW of wind energy would need to be produced. In the year 2024 Europe expanded the wind energy production by 16.4 GW. The largest addition has been made in Germany with 4 GW, of which 84% are located onshore [2]. Planning where to build a wind turbine is a difficult process, as we want to maximize the energy production, place it in areas where the energy is actually in demand, and also minimize effects on wildlife and humans. In this thesis we focus on a phenomenon that affects humans called shadow flicker or shadow throw. It occurs when the sun shines and the light passes through the rotating blades of the wind turbine. As this effect is perceived as unpleasant to residents [3], governments therefore limit the amount of exposure to shadow flicker that may occur. For instance, Germany restricts exposure to 30 minutes per day and 30 hours per year [4].

Therefore, it is advised to determine the exposure to shadow flicker prior to the construction of the wind turbine. If those limits are exceeded, wind turbines are shut down and not able to produce energy in those time frames. To avoid this, tools such as WindPro [5] or WindFarmer [6] offer functionalities to measure the occurrence of shadow flicker. Those compute the sun's position in certain time steps and evaluate whether shadow flicker occurs at that moment. This introduces a time-step discretization error that leads to either results that are not precise enough or require high computational effort.

The method that this thesis builds on models the occurrence of shadow flicker intervals as a continuous function, which is then solved using numerical methods to increase the precision of the solution compared to the discrete approach while keeping the computational effort as low as possible.

► **Original Method** This thesis is based on the paper and implementation proposed by Radke, De Smet, and Ábrahám [7]. Their work introduces a novel approach to shadow flicker simulation that improves upon techniques commonly used in industry-standard tools. The goal of this thesis is to apply various optimization strategies to reduce the runtime and memory consumption while increasing robustness and usability. Therefore, we introduce their approach in Section 2.

► **Outline** In Section 2 we provide an overview of the different components of the shadow flicker calculation. We explain how the continuous function is defined, consisting of the raytracing algorithm and the computation of the sun’s position.

To reduce the running time of the novel approach, we propose various optimization strategies of the implementation in Section 3.

In Section 4 we take a look at different data structures to find a suitable one that reduces the memory consumption without increasing the running time.

Building on these, Section 4 introduces the computation of shadow flicker for multiple turbines.

Finally, Section 5 describes how we can compute shadow maps and use them to look up durations.

List of Figures

1	Illustration of the ray-tracing algorithm from [7]	1
2	Special case where $dist_{prec}(\psi, \theta) < r_{tur}$ and $\theta(t) < 0$ from [7]	4
3	Example trajectory of the $dist_{prec}^+(t)$ for a day at $\{-975.0, -525.0, 0.0\}$ using the setup described in Section 2.6	6
4	Flowchart that models the algorithm of the shadow flicker computation, adapted from [7]	8
5	Computed shadow map for the annual shadow flicker exposure	9
6	Computed shadow map with classes for the annual shadow flicker exposure	10
7	Shadow flicker maps on selected days throughout the year.	10
8	Computed shadow flicker exposure with timestamps for day 2	11
9	Example for $dist_{prec}^+(t)$ having two shadow flicker occurrences	12
10	Simplified showcase of the projected solar path	13
11	Flowchart based on added features	18
12	Flowchart for computation of the shadow map	21
13	Example trajectory of $dist_{prec}^+(t)$, where the minimizer required 54 function evaluations to determine t_{min}	25
14	Memory consumption between the three data structures over time measured per second over a whole run	33
15	Error between computed shadow flicker duration and interpolated value .	40

Nomenclature

Sun's position

$\theta(t)$	elevation angle at time t	$[\theta(t) \in [-90^\circ, 90^\circ]]$
$\psi(t)$	solar azimuth angle at time t	$[\psi(t) \in [0^\circ, 360^\circ]]$
$\delta(t)$	solar declination angle t	$[\psi(t) \in [-23.45^\circ, 23.45^\circ]]$
$\zeta(t)$	solar zenith angle t	$[\zeta(t) \in [0^\circ, 180^\circ]]$
$\omega(t)$	solar hour angle t	$[\omega(t) \in [-180^\circ, 180^\circ]]$

Normal Vectors

$n_{sun}(t)$	the normal vector from the receiver position in the direction of the sun at time t
$n_{tur}(t)$	the normal vector from the receiver position in the direction of the turbine's rotor at time t

Turbine and Receiver Parameters

p_{tur}	position of the center of the turbine defined as center (usually also defined as the center of the coordinate system)	$[p_{tur} \in \mathbb{R}^3]$
p_{rec}	position of the receiver	$[p_{rec} \in \mathbb{R}^3]$
r_{tur}	radius of the turbine's rotor	$[r_{tur} \in \mathbb{R}]$
λ	latitude of the turbine	$[\lambda \in [-90^\circ, 90^\circ]]$
p^*	point where the normalvector n_{sun} hits the plane defined by n_{tur}	$[p^* \in \mathbb{R}^3]$
h_{tur}	height of the turbine	$[h_{tur} \in \mathbb{R}]$

Distance Function

$dist_{p_{rec}}(\psi, \theta)$	distance from p^* to p_{tur} , $\theta(t) \in [-90^\circ, 90^\circ]$
$dist_{p_{rec}}^+(t)$	distance from p^* to p_{tur} , where $\theta(t) \in [0^\circ, 90^\circ]$ and additionally the rotor radius r_{tur} has been subtracted

Measure of Time

$time_interval$	description of a time interval in hours with endpoints t_{start} and t_{end}	$[t_{start}, t_{end} \in [0, 24], t_{start} < t_{end}]$
------------------	--	---

Grid Settings

x_{min}	minimum x -coordinate of the grid
x_{max}	maximum x -coordinate of the grid
y_{min}	minimum y -coordinate of the grid
y_{max}	maximum y -coordinate of the grid
$step_size$	horizontal resolution of the shadow profile grid
min_height	lowest height layer in the simulation
max_height	highest height layer in the simulation
num_height_layers	number of vertical layers sampled

2 Preliminaries

2.1 Problem Statement

We are given a geographic location of the receiver(s) relative to the wind turbine, the height and rotor radius of the wind turbine, and the necessary input parameters to run the Solar Position Algorithm (SPA) [8] to calculate the position of the sun. The goal is to compute the daily and annual shadow flicker exposure as either a duration in minutes or a timeframe experienced at the receiver.

2.2 Main Concept

The fundamental idea behind the approach is to express the shadow flicker phenomenon as a continuous function.

For this, we compute a normal vector n_{sun} from the receiver to the sun, along with a second vector n_{tur} representing the turbine rotor's orientation. Following standard regulatory practice, one assumes that the turbine faces the sun to account for the worst-case scenario and over-approximate shadow flicker duration.

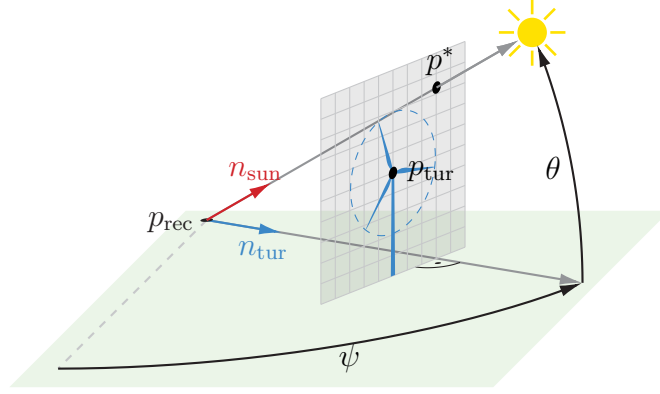


Figure 1: Illustration of the ray-tracing algorithm from [7]

The receiver position is denoted by p_{rec} , while the position of the turbine's center (hub) is given by p_{tur} .

The point where the vector n_{sun} intersects the plane defined by the turbine's rotor is referred to as p^* .

We define a continuous function that describes the distance between p^* and p_{tur} .

Shadow flicker occurs when this distance is less than the rotor radius.

To implement this functionality, we need a method to calculate the sun's position (Section 2.3), as well as a ray tracing procedure to determine geometric intersections (Section 2.4). By combining these two components, we obtain the executable function (Section 2.5) that models shadow flicker at a given time step t .

We compute the shadow flicker duration using numerical tools in Section 2.6.

2.3 Solar Position Algorithm

To calculate the position of the sun at a given time point t , we use the Solar Position Algorithm by Meeus [8], which is commonly applied to determine the sun's position accurately, with uncertainties as low as 0.0003° [9].

We require the following parameters for the Solar Position Algorithm to compute the sun's position accurately:

- latitude and longitude (degrees)
geographic coordinates of the receiver
- timezone (hours)
the local time zone offset from coordinated universal time (UTC), which is used worldwide
- elevation (meters)
elevation of the observation site above sea level
- pressure (millibars)
atmospheric pressure at the site
- temperature ($^\circ\text{C}$)
air temperature at the observation site
- atmospheric refraction (degrees)
atmospheric refraction at the horizon
- delta_ut1 (seconds)
difference between the most direct measure of Earth rotation time (UT1) and UTC
- delta_t (seconds)
difference between Terrestrial Time and Universal Time.
- year
calendar year of the observation

Note that delta_ut1 accounts for fluctuations in Earth's rotation, and delta_t deals with the difference between terrestrial time (TT) and universal time (UT1).

To define the position of the sun, we typically specify its location relative to an observer on Earth using two primary angles shown in Figure 1:

- elevation angle $\theta(t)$ ranges from 0° to 90° :
describes the angle between the sun and the local horizon
- azimuth angle $\psi(t)$ ranges from 0° to 360° :
represents the horizontal angle measured clockwise from true north
(north = 0° , east = 90° ...)

To calculate these angles, we apply the following two formulas.

$$\sin(\theta) = \sin(\delta)\sin(\lambda) + \cos(\delta)\cos(\lambda)\cos(\omega) \quad \sin(\psi) = -\frac{\sin(\omega)\cos(\delta)}{\sin(\zeta)} \quad (1)$$

The variables introduced in Equation (1) are defined as follows:

- solar declination angle $\delta(t)$ ranges from -23.45° to 23.45° :
describes the angle between the sun direction and the equatorial plane of the earth
- geographic latitude λ ranges from -90° to 90°
- solar hour angle $\omega(t)$ ranges from -180° to 180° :
The angular measure of time is defined as the angle between the sun's current position and the local solar noon, measured westward along the celestial equator. It represents how far the sun has moved across the sky from its highest point.

While the latitude is provided as an input parameter, both the solar declination $\delta(t)$ and the solar hour angle $\omega(t)$ are computed using the SPA Algorithm.

2.4 Ray Tracing Algorithm

As the positions of the receiver and turbine are given, and the position of the sun can be calculated for a given time step t , we can determine when the sun intersects the plane of the wind turbine and thus when shadow flicker occurs. In other words, this involves tracing a line from the receiver to the sun and testing for intersection with the turbine's rotor. This corresponding direction is represented by the normal vector n_{sun} . In addition, we require the orientation of the turbine rotor, denoted by n_{tur} as shown in Figure 1. We determine n_{sun} and n_{tur} based on the following equations:

$$n_{sun} = \begin{pmatrix} \cos(\theta(t)) \cos(\psi(t)) \\ \cos(\theta(t)) \sin(\psi(t)) \\ \sin(\theta(t)) \end{pmatrix} \quad \text{and} \quad n_{tur} = \begin{pmatrix} \cos(\psi(t)) \\ \sin(\psi(t)) \\ 0 \end{pmatrix}. \quad (2)$$

We define p_{tur} as the center of the hub and also utilize it as the origin of the coordinate system, such that $p_{tur} = (0, 0, 0)^T \in \mathbb{R}^3$.

Furthermore, p^* denotes the intersection point of the normal vector n_{sun} with the plane spanned by the turbine rotors, as shown in Figure 1.

Our continuous function evaluates the distance between p^* and p_{tur} . Shadow flicker occurs if this distance is less than the rotor radius.

Since the vector $p^* - p_{tur}$ lies within the plane whose normal vector is n_{tur} , it follows that $p^* - p_{tur}$ is perpendicular to n_{tur} . By definition, two vectors are orthogonal if their dot product is equal to zero. Thus, we can deduce:

$$0 = (p^* - \underbrace{p_{tur}}_{=(0,0,0)}) \cdot n_{tur} = p^* \cdot n_{tur} \quad (3)$$

We further know that the point p^* lies on a straight line connecting the receiver and the sun. Thus, it can be expressed as:

$$p^* = p_{rec} + d \cdot n_{sun} \quad (4)$$

for some $d \in \mathbb{R}$.

Substituting this expression into the previously derived equation yields:

$$0 = (p_{rec} + d \cdot n_{sun}) \cdot n_{tur} \quad \Rightarrow \quad d = -\frac{p_{rec} \cdot n_{tur}}{n_{sun} \cdot n_{tur}}. \quad (5)$$

Substituting the expression for d of equation (5) into (4) yields:

$$p^* = p_{rec} - \left(\frac{p_{rec} \cdot n_{tur}}{n_{sun} \cdot n_{tur}} \right) \cdot n_{sun} \quad (6)$$

Thus we can check whether the distance between p^* and p_{tur} is larger than r_{tur} :

$$\|p^* - p_{tur}\|_2 \leq r_{tur} \quad \xrightarrow{p_{tur}=(0,0,0)} \quad \|p^*\|_2 \leq r_{tur} \quad (7)$$

From (7) we now know that shadow flicker occurs if the distance between p^* and p_{tur} is smaller than the rotor radius r_{tur} .

2.5 Distance Function

Based on Section 2.4, we define our distance function in the following way:

$$dist_{p_{rec}}(\psi, \theta) := \|p^*\|_2 \quad (8)$$

The condition $dist_{p_{rec}}(\psi, \theta) < r_{tur}$ may also be satisfied when the receiver lies between the sun and the turbine, i.e., when $\theta(t)$ is negative, as shown in Figure 2.

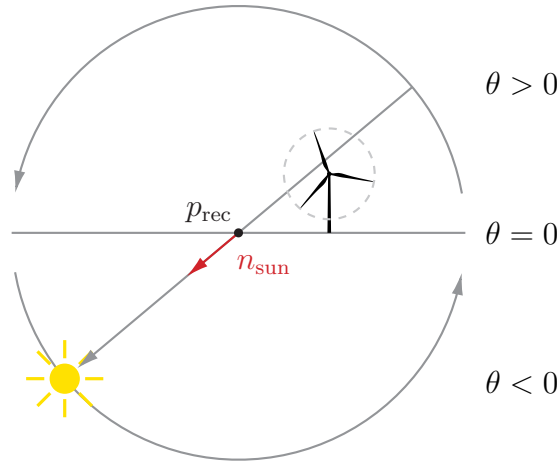


Figure 2: Special case where $dist_{p_{rec}}(\psi, \theta) < r_{tur}$ and $\theta(t) < 0$ from [7]

Therefore, we add the condition $\theta > 0$ and define the distance function:

$$dist_{prec}^+(t) = \begin{cases} dist_{prec}(\psi(t), \theta(t)) - r_{tur} & \theta(t) \geq 0 \\ dist_{prec}(\psi(t), 0) - r_{tur} & \text{otherwise} \end{cases} \quad (9)$$

We subtract the radius of the turbine r_{tur} , since then shadow flicker occurs if and only if $dist_{prec}^+(t) < 0$. Note that we assume that the receiver is placed lower than the lowest point of the rotor to guarantee that $\|p^*\|_2 > r_{tur}$ for $\theta = 0$.

This function depends on the azimuth angle $\psi(t)$ and the elevation angle $\theta(t)$, therefore solving the problem analytically is not possible with current models.

Another edge case is $\theta \in \{-90^\circ, 90^\circ\}$ as the sun is then positioned exactly above or below the turbine, causing n_{tur} to be undefined. This can be excluded using case distinctions because shadow flicker does not occur in those cases.

2.6 Shadow Flicker Computation

In this section we use the defined continuous function $dist_{prec}^+(t)$ and calculate the shadow flicker exposure based on it.

For this, we make the following worst-case assumptions:

- neglecting that clouds may obstruct sunlight
- wind turbines are assumed to operate continuously
- rotors of the wind turbine always face the sun for maximum shadow flicker exposure

We characterize shadow flicker by an indicator function $\mathbb{1}_{SF}(t)$, which evaluates to 1 when the conditions for shadow flicker are satisfied at time t , and 0 otherwise.

$$\mathbb{1}_{SF}(t) = \begin{cases} 1 & 0^\circ < \theta(t) < 90^\circ \wedge dist_{prec}^+(t) \leq 0 \\ 0 & \text{otherwise} \end{cases} \quad (10)$$

(Cases where $\theta \in \{-90^\circ, 90^\circ\}$ are excluded.)

We calculate the shadow flicker duration T for a day $d = [t_{start}, t_{end}]$ using

$$T = \int_{t_{start}}^{t_{end}} \mathbb{1}_{SF}(t) dt \quad (11)$$

Note that this equation cannot be solved analytically using standard integrable models.

To determine whether shadow flicker occurs, we first minimize the function using Brent's minimization algorithm.

► **Example Trajectory** We demonstrate how the function’s trajectory evolves over time. For this we extracted an using the setup described in Section 2.6.

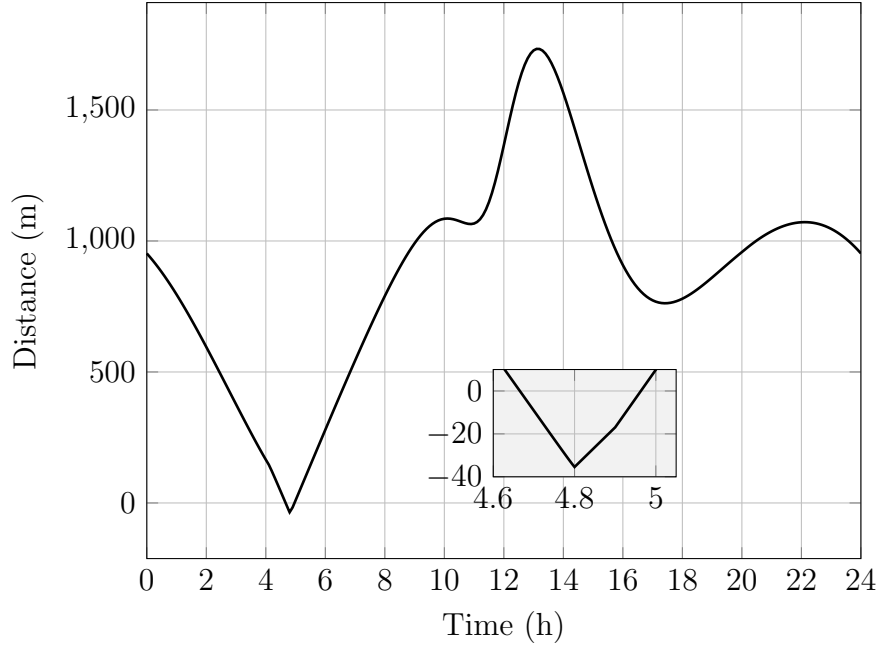


Figure 3: Example trajectory of the $dist_{prec}^+(t)$ for a day at $\{-975.0, -525.0, 0.0\}$ using the setup described in Section 2.6

Step	x	$f(x)$
0	12.0000	1365.73
1	7.4164	649.11
2	4.5835	21.79
3	2.8328	413.26
4	4.8596	-27.68
5	5.8362	235.41
6	4.9718	2.33
7	4.7994	-35.31
8	4.7519	-23.03
9	4.8118	-37.54
10	4.8197	-37.29
11	4.8142	-37.65
12	4.8146	-37.65
13	4.8146	-37.65
14	4.8145	-37.65

Table 1: Brent algorithm evaluations for minimizing the function in Figure 3

While the algorithm converges efficiently, it potentially can get stuck at local minima, such as $x \approx 11$ in Figure 3.

If we apply Brent's minimization algorithm for this example, the algorithm finds the global minimum at $x \approx 4.8$.

► **Example Evaluation** The initial minimization is performed over the interval $[0, 24]$, with an initial guess of 12 and gives us the result after 15 function evaluations. Note that it takes about 4 iterations to locate the general region of the minimum, and the remaining iterations gradually refine the estimate to reach the desired precision.

► **Minimization Subroutine** We assume that shadow flicker occurs only during at most one continuous time interval per day, either when the sun rises or when the sun sets. Therefore, the function also should have two local minima. In case the algorithm does not find a minimum with $dist_{prec}^+(t) < 0$, we need to minimize again to find the second minimum.

If the first minimum t_{min} is occurring before noon, then we search in the interval $[t_{min} + 6, t_{min} + 18]$ with $t_{min} + 12$ as initial guess.

If the first minimum occurs after noon, then we search in the interval $[t_{min} - 18, t_{min} + 6]$ with $t_{min} + 12$ as initial guess.

In case $dist_{prec}^+(t) > 0$, for all $t \in [0, 24]$ and therefore $t_{min} > 0$, then no shadow flicker occurs on that day, therefore we return 0 as the duration.

► **Root Finding Subroutine** We assume that we found a minimum t_{min} with $dist_{prec}^+(t_{min}) < 0$, which implies that shadow flicker happens on this day.

The goal is now to determine the duration of the shadow flicker occurrence.

As we subtracted the rotor radius from the distance to the hub, we calculate t_{start} and t_{end} by calculating the roots.

For that we consider the absolute value of $dist_{prec}^+(t)$.

By that, we can reuse the previously introduced minimization algorithm, as the roots will now be the new minima of the function.

► **Edge Case Near Poles** Near the north or south pole, the shadow flicker interval might happen at midnight, which means that shadow flicker can be observed twice a day. We do not address this edge case in this thesis, as it is impractical and unlikely to occur in real-world scenarios.

With this method in place, we can now determine the shadow flicker exposure for each day. This involves up to four minimizations per day, after which the annual exposure is obtained by summing up the individual daily durations.

► Flowchart of the Algorithm

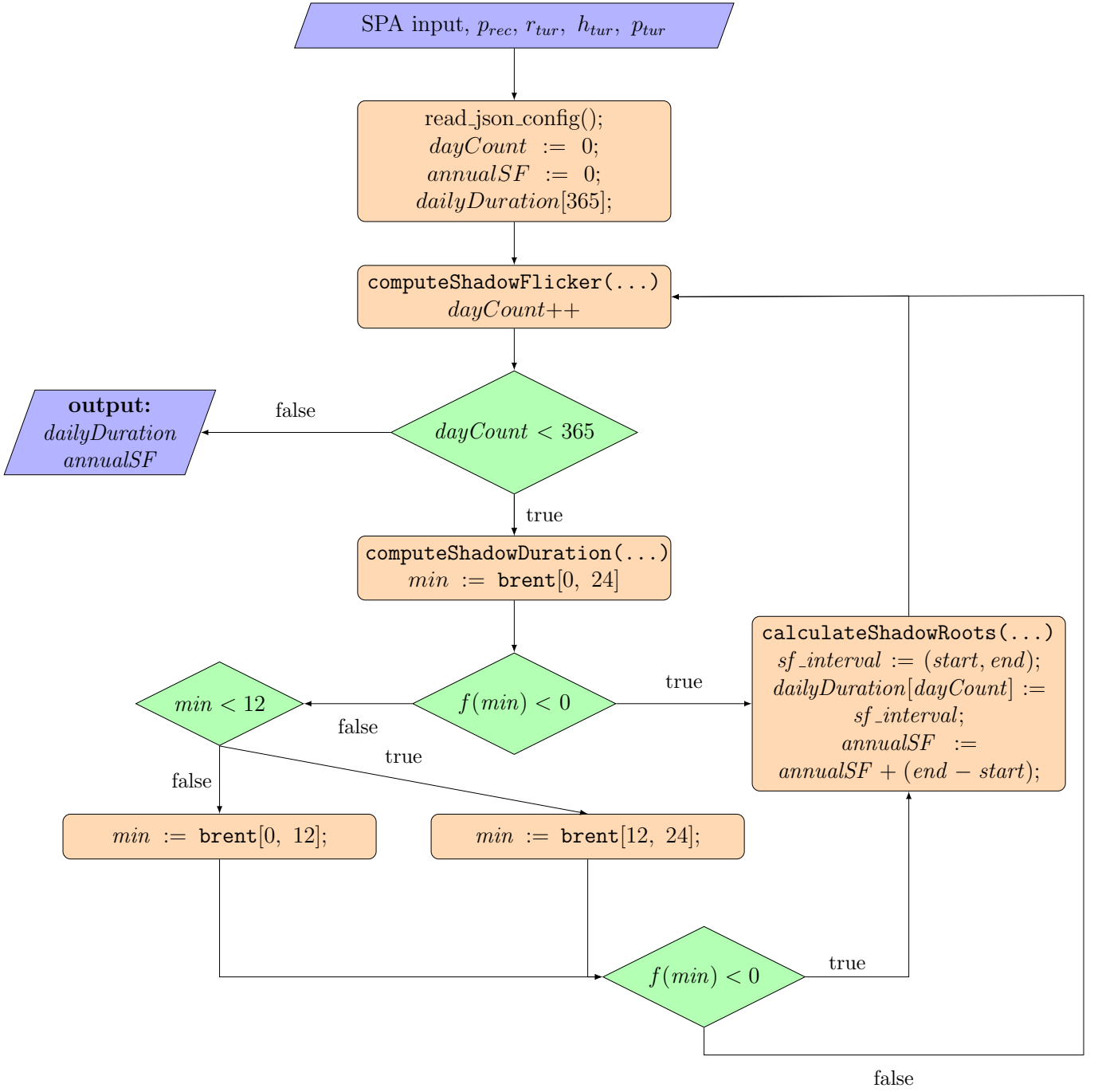


Figure 4: Flowchart that models the algorithm of the shadow flicker computation, adapted from [7]

► **Case Study** In order to stay consistent and ensure comparability, this thesis employs a single case study across all sections.

We are going to use it to illustrate examples, validate correctness, and evaluate runtime performance. As a location, we choose Aachen ($50.7753^\circ N, 6.0839^\circ E$). We place a wind turbine with a $100m$ height and a rotor diameter of $80m$ in the center of a map that has a size of $3000m$ by $3000m$. The receivers are placed in a grid with a step size of $50m$. As for the ground height we choose $0m$ and the year of interest is 2025.

► **Shadow Map** In the following, we provide a plot for a shadow map. We define it as a grid with a defined height and a defined step size.

The position of the turbine is in the center of the grid.

We use the dark blue color for grid points that have not been exposed to shadow flicker. The duration goes up to 18819.27 min for a grid point, and we notice that the duration rises a lot close to the turbine.

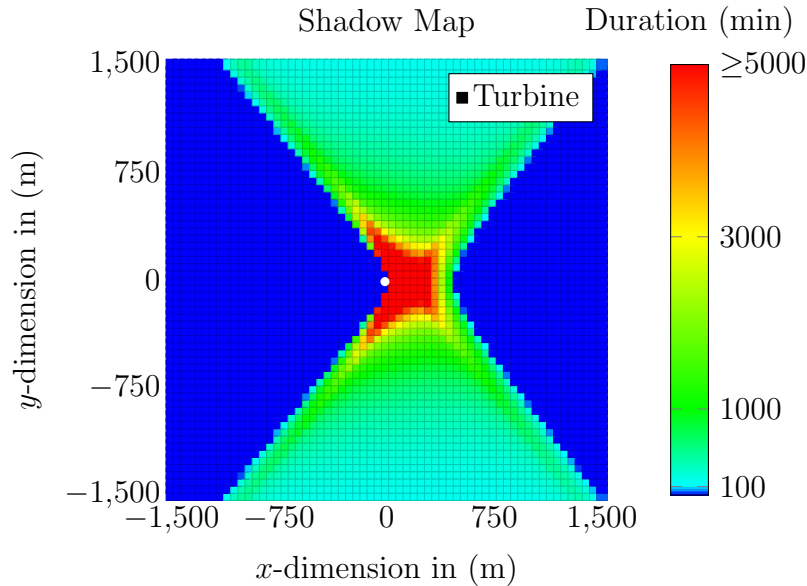


Figure 5: Computed shadow map for the annual shadow flicker exposure

► **Shadow Map with Classes** It is useful to know when certain grid points exceed the daily or yearly threshold. For this we can choose to visualize the durations depending on the thresholds. In case of Germany, we want to know whether the durations exceed 30 min for the daily and 30 hours for the annual duration.

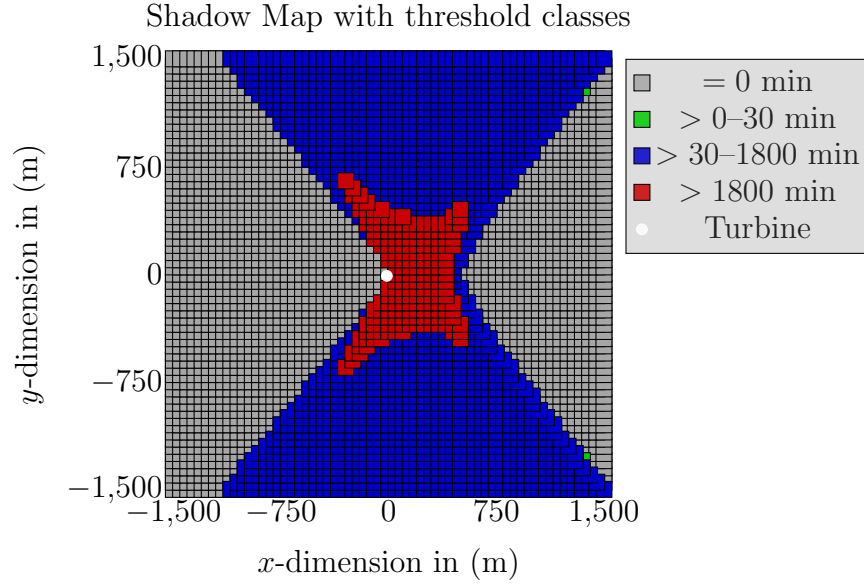


Figure 6: Computed shadow map with classes for the annual shadow flicker exposure

► **Shadow Flicker Movement across the Year** We illustrate how shadow flicker moves across the year. It is noticeable that the sun moves in a cone-like structure that originates close to the turbine. This explains why the shadow flicker exposure close to the turbine is higher than outside the turbine.

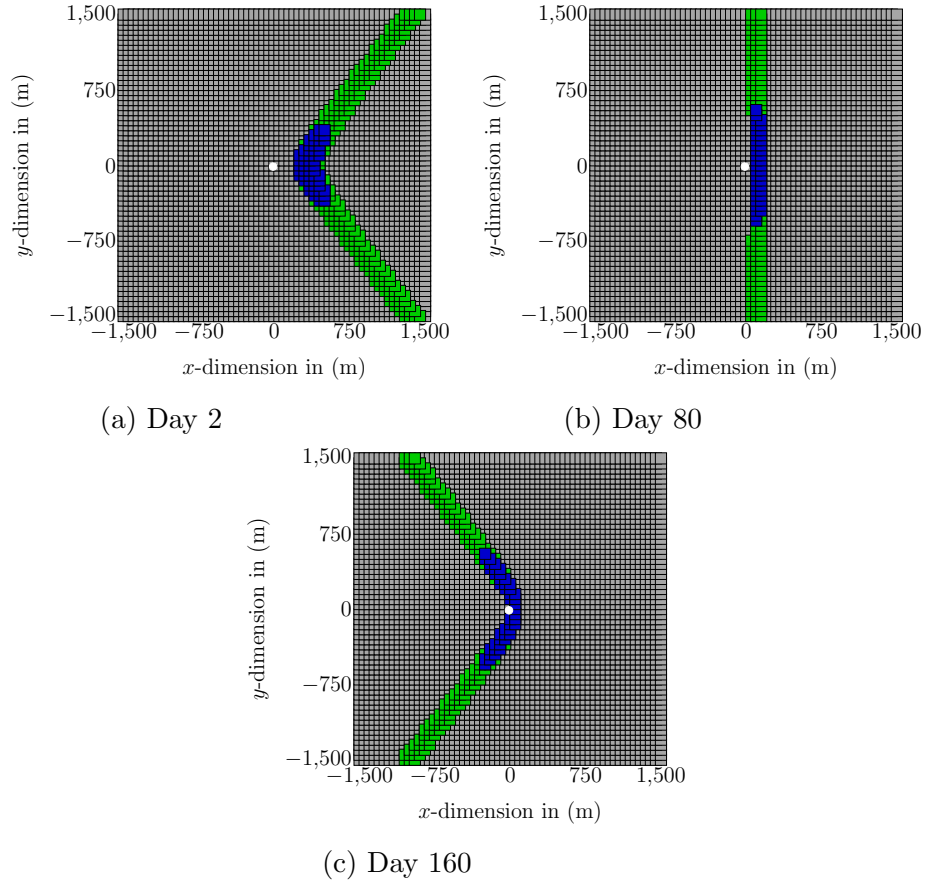


Figure 7: Shadow flicker maps on selected days throughout the year.

Note that this pattern is not universal across the earth. Depending on the location of earth, the sun moves with different patterns. For example in Canada, the northernmost inhabited place, the sun remains either above or below the horizon most of the year. Only in March, April, September and October the sun actually sets [10].

► **Shadow Flicker Movement across the Day** In the following we show how shadow flicker occurs throughout the day. For days where no shadow flicker occurs, we denote 0 as time which is represented by a dark blue color. We observe that the earliest shadow flicker exposure that is visible on the map occurs at 8:56, while the latest exposure happens at 16:21.

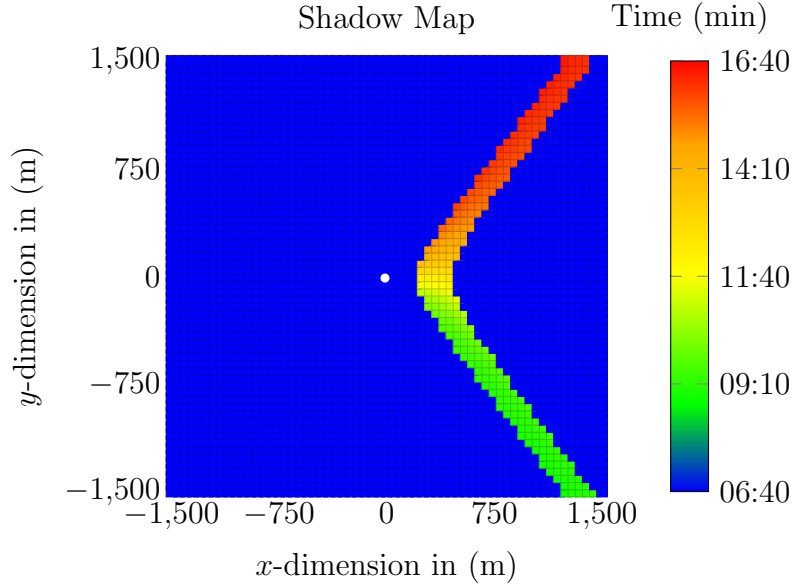


Figure 8: Computed shadow flicker exposure with timestamps for day 2

2.7 Correctness and Validation

As the underlying models are too complex to be solved using analytical models, proving correctness is also technically challenging.

There are two assumptions that need to be proven to show correctness.

► **Assumption 1** Shadow flicker occurs each day continuously during a single interval $[t_{start}, t_{end}]$.

It is argued that, under all but the most extreme conditions, the sun's directional movement is too gradual to result in two separate entries into the shadow flicker region.

► **Example for two Shadow Flicker Intervals in one Day** Consider the setup of the case study in Section 2.6. We choose the receiver to be positioned at $(0, 75, 0)$ relative to the turbine.

If we analyze our continuous function $dist_{prec}^+(t)$ on the 1st July 2025, we encounter the following trajectory:

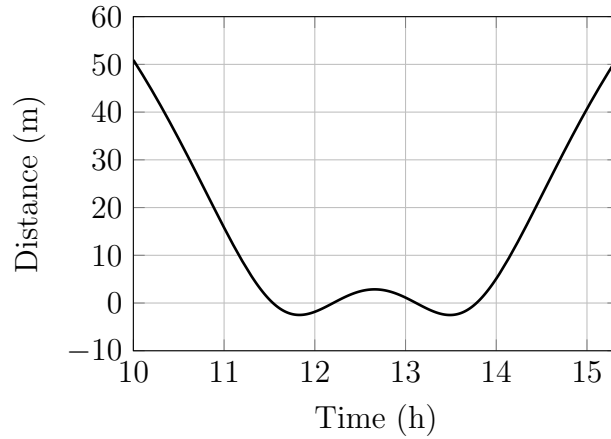


Figure 9: Example for $dist_{prec}^+(t)$ having two shadow flicker occurrences

We observe that shadow flicker actually occurs twice on this day. To understand why we need to observe how the vector n_{sun} is able to intersect the rotor twice.

For this we developed a small visualization tool that needs the *time*, *azimuth*, and *elevation* based on the given receiver for that day. We evaluated those in a *step_size* of *3min*. Based on that, we can visualize the orientation of the normal vector n_{sun} and the orientation of the turbine n_{tur} .

To visualize this in 2D, we projected the points p^* on a two dimensional plane to show the solar path on this day for a positive elevation angle.

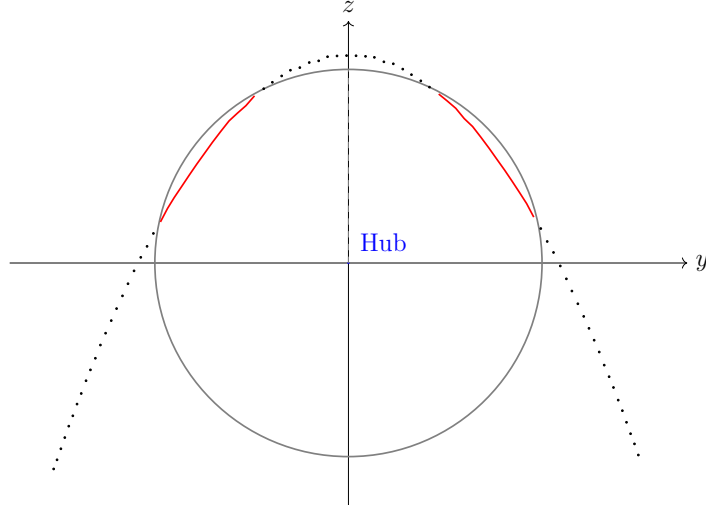


Figure 10: Simplified showcase of the projected solar path

As we can see, n_{sun} intersects with the rotor in the top left corner and leaves the area again. At the right corner it reenters the rotor area again, causing the second shadow flicker exposure. After that, it leaves the area.

Therefore, we can conclude that not only the movement of the sun is a factor, but also the geometry between the receiver, the turbine, and the sun.

During the course of the thesis, we encountered such an edge case only once.

► **Assumption 2** The numerical minimization algorithm succeeds in finding the minimum for a day.

The argument presented is that the function $dist_{prec}^+(t)$ depends solely on the sine and cosine of the hour angle ω , combined through operations such as squaring, square rooting, addition, multiplication, and division. Since these are smooth operations applied to trigonometric functions, and the number of local minima of a trigonometric function over a bounded interval is finite, it follows that the number of minima of $dist_{prec}^+(t)$ is also bounded. We conclude that the function is smooth enough to reliably find the minima.

3 Performance Improvements to the Novel Approach

To calculate the shadow flicker duration for a day, we need to evaluate our continuous function $dist_{prec}^+(t)$ frequently in order to find its minimum and eventually calculate the roots of the function. In this section, we apply techniques to reduce the number of function evaluations necessary to improve the performance without losing solution quality.

3.1 Technical Improvements

We apply some changes to the given implementation to extend the usability.

► `time_interval`

The whole implementation uses variables of the type double to log timestamps and durations. We define a struct that allows us to store shadow flicker occurrences more precisely. A *time_interval* consists of a double *start* and a double *end*, which we can use to store an interval.

So instead of calculating the interval and storing the duration for a day, we store the *time_interval* in which shadow flicker occurs.

A *time_interval* of (900, 930) represents 3PM to 3 : 30PM (900/60 , 930/60).

We provide 3 different output formats, which the user can select:

- 1: annual duration (minutes)
- 2: annual duration (minutes) and daily time intervals (minutes)
- 3: annual duration (minutes) and daily duration (minutes)

► More Robust Interval Estimation

Currently, after calculating the minimum of the function t_{min} , we search in a static interval $\pm 2h$. In most cases, shadow flicker is a local phenomenon, but there might be extreme cases where it exceeds this limit, especially close to the wind turbine. As we want the algorithm to perform robustly and work as precisely as possible, we add a fallback that extends this interval by half an hour till we find a positive function value.

3.2 Root Finding

First, we replace the current root finding with a more efficient and robust root-finding algorithm. Currently we minimize twice over the absolute value function in order to find the roots.

In general, minimizing is a harder problem than root finding. Therefore, we still decided to use Brent's method, but with an extension for root finding [11].

► **Brent's Method** Brent's method for root finding is a robust algorithm that combines the bisection method, the secant method, and inverse quadratic interpolation to find a root of a real-valued function $f(x)$ in a given interval. However, certain requirements must be met for it to work correctly:

- executable function $f(x)$ (compute $f(x)$ at arbitrary $x \in [a, b]$)
- continuous function $f(x)$
- bracketed interval $[a, b]$ such that $f(a) \cdot f(b) < 0$

As shown in Section 2.5, the first two items are already fulfilled. For the third item, we need to find a value a such that $dist_{prec}^+(a) < 0$ and a value b such that $dist_{prec}^+(b) > 0$ or vice versa. By minimizing $dist_{prec}^+$, we will only apply root finding if we find a value $dist_{prec}^+(t) < 0$. We also assume that since shadow flicker events are brief and they typically occur no more than once per day, at some point, we will find $dist_{prec}^+(b) > 0$. So we can conclude that Brent's method is suitable for our use case. The new root-finding algorithm calculates the roots with a precision of 10^{-7} instead of 0.05 in the original approach.

3.3 Reusing the Prior Day's Guess

The sun's path changes only gradually from day to day. Therefore, it is reasonable to log the last day's minimum $t_{last_day_min}$ and use it as an initial guess for the next day. If $dist(t_{last_day_min}) < 0$, then shadow flicker is guaranteed to occur. In this case, we bypass the minimization process for the current day and proceed directly to root calculation. We allocate a struct to store the necessary information.

```

1 typedef struct {
2     bool had_sf_yesterday; // 1 -> shadow flicker yesterday
3     double last_day_guess; // time of minimum from previous day
4     double last_day_guess_value; // value of last_day_guess
5 } sf_initial_guess;

```

Algorithm 1: Structure for initial guesses

We only need to allocate this struct once, as we calculate the results for each receiver individually for each day in succession. Therefore, we have to initialize it once at the beginning and also reset it to the following default values in case we change the receiver:

- $had_sf_yesterday = false$
- $last_day_guess = 12$
- $last_day_guess_value = 0;$

Note that we update *last_day_guess* and *last_day_guess_value* every time we minimize using Brent. In case shadow flicker occurs on a day, we set the boolean *had_sf_yesterday* to *true* for the next day inside the root-finding subroutine.

On the following day we use the minimum of the *last_day_guess* and evaluate the function value. If the function value is negative, then we can skip the minimization routine and directly evaluate the roots of the function.

Brent's method itself also accepts an initial guess. This also means that storing the *last_day_guess* is still beneficial if no shadow flicker occurs, as we can use it as an initial guess for Brent's method as the algorithm converges faster to the new minimum.

In case we use the procedure described in Section 3.9, we need to allocate this struct for every receiver at the same point instead of just once, which consumes more memory at runtime.

3.4 Reusing the Midpoint of Prior Day's Interval

Based on the same assumption that the sun's path does only change gradually from day to day, we can calculate the midpoint $t_{midpoint} = \frac{t_{start} + t_{end}}{2}$ and use it as an initial guess for the next day. For this we extend the definition of the initial guess struct as follows:

```

1 typedef struct {
2     bool had_sf_yesterday;
3     double last_day_guess;
4     double last_day_guess_value;
5     double midpoint_guess;
6     double midpoint_guess_value;
7 } sf_initial_guess;
```

Algorithm 2: Extended structure for initial guesses

We added the possibility to store a *midpoint_guess* and the *midpoint_guess_value*. Both values can be initialized with 0, as we only allow their usage in case shadow flicker occurred on the previous day. In case shadow flicker occurred on a day, we set the *had_sf_yesterday* to *true* and set the *midpoint_guess* to $\frac{t_{start} + t_{end}}{2}$. Since we can only use this feature if shadow flicker occurred the day before, the previous feature in Section 3.3 is more useful, as it also improves performance when no shadow flicker happens.

3.5 Early Termination of Minimization Subroutine

As discussed in Section 3.2, we only need to find a value such that $dist_{pre}^+(t) < 0$ to obtain a bracketed interval. It is therefore sufficient to stop minimizing once a negative value is found. In the given example in Section 1, 15 function evaluations were necessary to find the minimum. If we use the weaker condition, we would only need 8 function evaluations.

```

1  /* This is the one function evaluation per iteration. */
2  fu = (*f)(u, params);
3  if (fu < 0.0) {
4      *xmin = u;
5      return fu;
6  }

```

Algorithm 3: Check for negative function value in Brent iteration

In every iteration of Brent’s method, the function is evaluated once per iteration. Every time the function is evaluated, we check whether the function is negative. In case it is negative, we return early. Else we continue with Brent’s method till we converge to the desired precision.

3.6 Relationship Between Features

In the original version of the implementation, the root finding is done by minimizing the absolute function.

Assuming that the precise minimum is t_{min} , we would minimize the absolute function in the intervals $[t_{min} - 2, t_{min} - 0.1]$ and $[t_{min} + 0.1, t_{min} + 2]$. In case we do not compute the minimum precisely, we might end up converging to the same root twice, as Brent follows the local descent direction. In those cases the result interval (*start*, *end*) would have the same start and end, meaning we compute 0 as the duration although in fact shadow flicker occurred.

A way to detect such errors is by comparing results of a correct implementation to the results of the new implementation and inspecting the deviations.

Doing so, we found an error in the old implementation that we are going to inspect in Section 2.7.

To avoid such errors, we also need to use proper root finding in order to use any of the features from before.

3.7 Flowchart of the Algorithm with Optimizations

The algorithm still uses the same data structure. We introduced the technical changes, adapted the minimization subroutine, and changed the root finding that does not change the process itself. Both initial guess variations extend the implementation of the *computeShadowDuration(...)* function. We also need to add some data structure to maintain the guesses. Note that we have to loop over each receiver and apply the process described in the flowchart.

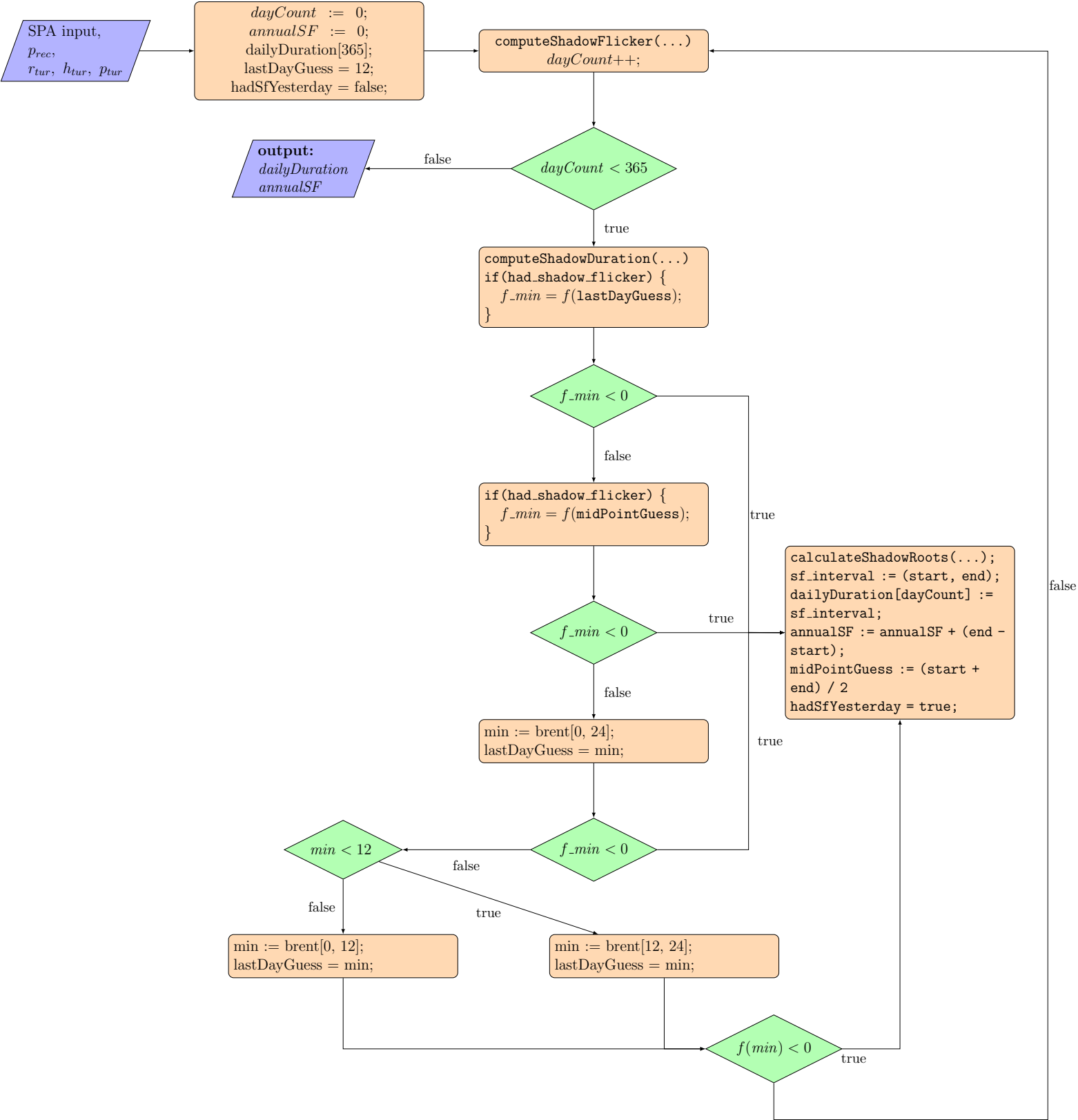


Figure 11: Flowchart based on added features

3.8 Reducing the Computational Effort

Finding a heuristic that minimizes the number of receivers that need to be evaluated by a meaningful amount would result in a huge performance gain.

In our case study, we defined a grid of $61 \cdot 61 = 3721$ receivers. For each receiver, it is necessary to calculate the shadow flicker exposure for 365 or 366 days. To calculate the duration precisely or to make sure that no shadow flicker occurs, we need to minimize the function 2 times in the worst case. This means to calculate the shadow flicker duration over the whole grid, we need to minimize our function $3721 \cdot 365 \cdot 2 = 2716330$ times.

3.9 Heuristic - Grid Property

We can make the following assumptions:

- all receivers are arranged in a grid:
This is fulfilled for our case.
- the sun moves in a continuous way across the sky:
The motion of the sun across the sky is caused by the Earth rotating at a constant angular velocity. Since this rotation is smooth and uniform, the sun's position changes continuously over time.

Based on these assumptions, we implement a propagation-based approach.

► **Initialize the Propagation** To initiate the search, we need to find a grid point whose shadow flicker duration is nonzero. For this, we start at the center of the turbine and compute the shadow flicker duration across the x and y axis. If we find a grid point whose duration is nonzero, we terminate the search and use this grid point as the start for the propagation search.

There is also the possibility that no grid point experiences any shadow flicker on a given day. In this case, the search terminates upon reaching the boundaries of the map.

► **Propagation** Assuming that we start the propagation-based search with a grid point whose shadow flicker duration is nonzero.

Now we employ a queue and enqueue every neighbor of a receiver whose shadow-flicker duration is greater than zero. We also need to ensure that grid points are not enqueued multiple times and that points whose durations have already been computed are not enqueued again. Therefore, we keep track of grid points that either have been queued already or whose duration has been computed already using an array.

► **Optimizations** Currently, we would need to perform the initial search every day to find a grid point that experiences shadow flicker exposure on that day. We can avoid this by storing the index of the grid point that experienced the longest shadow flicker duration on that day and using it as a guess for the next day. In case the duration is nonzero, we start the propagation from that grid point.

In addition, we also integrated the optimizations used for the novel approach. Since we now compute the duration for all receivers day by day, rather than calculating all days for a single receiver, we must allocate an instance of the *sf_initial_guess* struct for every receiver instead of just once. This leads to more memory consumption at runtime compared to the original approach.

► Flowchart of the Shadow Map computation

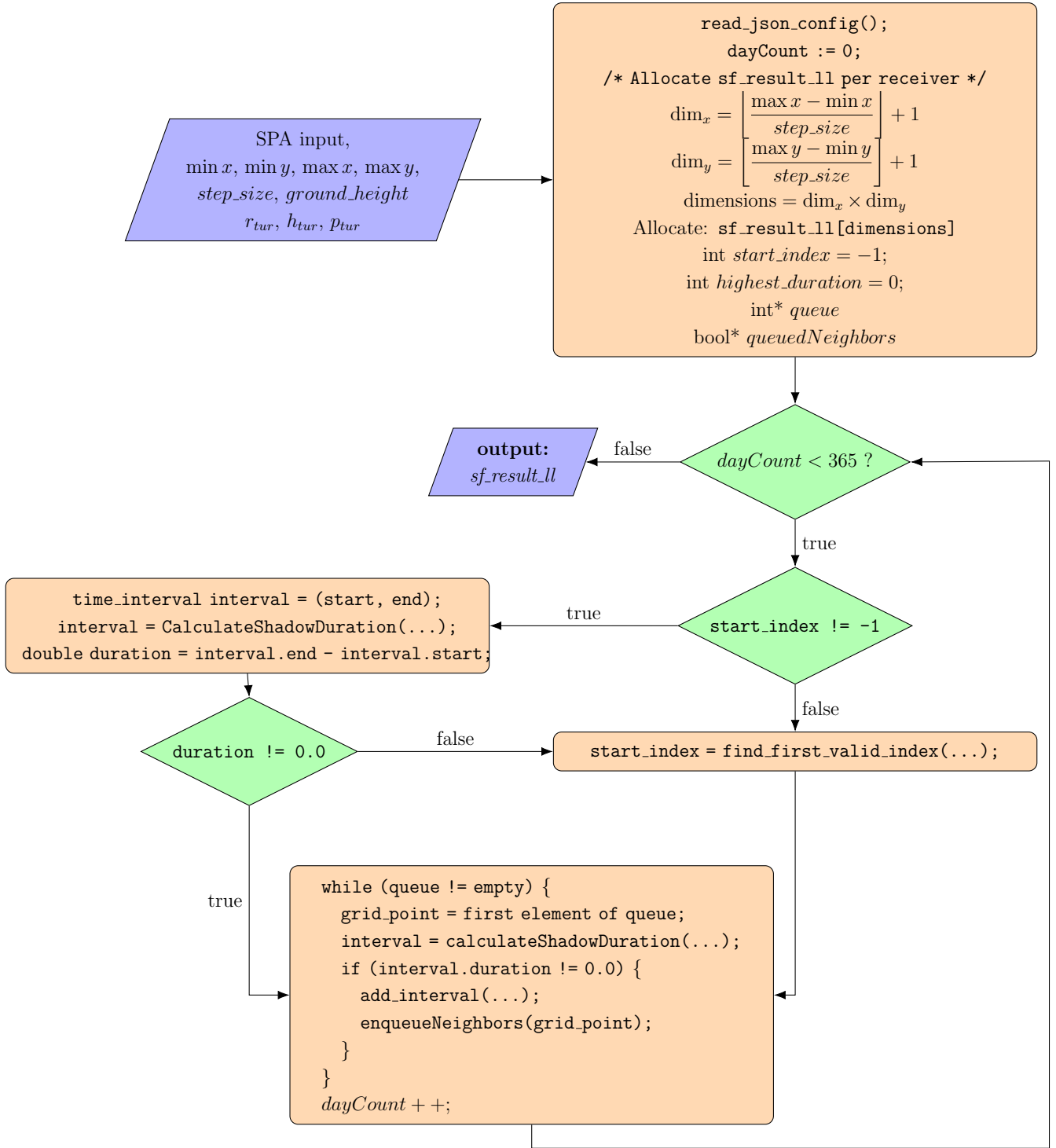


Figure 12: Flowchart for computation of the shadow map

3.10 Correctness

As the model for the solar position is very complex and it is not possible to solve it using current analytical methods, we take a different approach.

To validate that all features work as intended, we compare the results of the original implementation and the version that includes the optimizations. If the features work correctly, we get the same output while reducing the overall runtime. Therefore, we implemented and evaluated each feature independently, apart from the accurate root finding, to validate correctness based on the assumption that the provided implementation works correctly.

3.11 Validation

We implemented various features that should improve the runtime of the algorithm.

In the following, we measured the runtime of the original implementation and every feature individually. As described in Section 3.6 we need accurate root-finding to assure that the computed solutions are correct. That means that for all features, we test them with accurate root-finding. Note that the old implementation calculates the root with a precision of 0.05. The new root-finding algorithm calculates the roots with a precision of 10^{-7} .

We also chose three different locations to evaluate the runtime.

1. Aachen (50.7753°N, 6.0839°E)
2. Phoenix (33.4484°N, -112.0740°E)
3. Tokyo (35.6895°N, 139.69170740°E)

The map dimensions and the turbine position stay the same.

Method	Config	Value 1	Value 2	Value 3	Average
Original	1	174.23	174.73	174.72	174.56
	2	187.37	188.36	188.47	188.07
	3	191.14	191.36	192.22	191.57
Accurate Root Finding	1	170.31	170.57	170.46	170.45
	2	185.09	184.83	184.91	184.94
	3	191.42	190.95	189.90	190.76
Last Day Guess	1	157.93	158.79	158.65	158.46
	2	165.39	165.72	165.93	165.68
	3	176.88	174.28	177.30	176.15
Midpoint Guess	1	165.02	164.77	167.58	165.79
	2	180.12	180.14	180.54	180.27
	3	184.66	185.63	185.66	185.32
Early Term Minimization	1	172.54	170.72	171.73	171.66
	2	184.36	184.81	183.39	184.19
	3	191.31	190.46	190.78	190.85
Propagation	1	17.22	16.79	16.94	16.98
	2	12.21	12.19	12.23	12.21
	3	11.97	12.29	11.90	12.05
All Features (no Propagation)	1	153.31	153.78	152.71	153.27
	2	164.71	164.61	164.33	164.55
	3	170.38	170.34	170.42	170.38

Table 2: Evaluated the runtime of each feature across three configuration files: (1) Aachen, (2) Arizona, and (3) Tokyo.

► **Analysis of Runtime Results** Across the runs, we noticed that the difference in runtime is stable, as the values only differ by 2 to 3 seconds at most. Therefore, we evaluate the runtime 3 times and compute the average across those runs.

We first notice that the results for Aachen compared to Tokyo and Arizona differ because of the different latitude. Tokyo and Arizona score similar results, as they have similar latitudes.

► **Number of Function Evaluations** To analyze the performance of the proposed approach in greater depth, we introduce an additional set of evaluation metrics. Specifically, we consider the following:

- the number of function evaluations of $dist_{prec}^+(t)$ during (1) minimization and (2) root-finding,
- the number of Brent minimizer calls,
- the number of Brent root-finder calls,
- the number of successful Last Day Guesses,
- the number of successful Midpoint Guesses.

First we investigate how much the chosen location influences the performance:

Metric	Aachen	Phoenix	Tokyo
Brent minimizer calls	2 669 836	2 689 966	2 684 130
Function evaluations (min)	31 874 089	34 945 758	36 684 630
Brent root-finder calls	136 678	91 496	94 572
Function evaluations (root)	1 901 957	1 276 340	1 328 396
Days with flicker	68 339	45 748	47 286

Table 3: Performance measurements without optimizations across three locations.

We evaluated shadow flicker occurrence over a total of $3721 \cdot 365 = 1\,358\,165$ simulated days. Out of these, shadow flicker was detected on 68 339 days in Aachen, whereas for Phoenix and Tokyo the numbers were significantly lower, with 45 748 and 47 286 days, respectively. For each day with detected flicker, the shadow flicker interval must be determined by computing the two roots of the function.

The results for the number of Brent minimizer calls and the corresponding function evaluations indicate the need for further analysis. In particular, the minimizer requires a substantially larger number of evaluations to determine the minimum of the function in Phoenix and Tokyo compared to Aachen. This behavior can be explained by the geometry of the problem:

Depending on the relative position of the Sun with respect to the turbine and the receiver, the minimum of $dist_{prec}^+(t)$ may occur near the boundary of the considered day. In such cases, the minimizer struggles to converge efficiently and performs considerably more evaluations.

To illustrate this effect, we provide a representative example in the following:

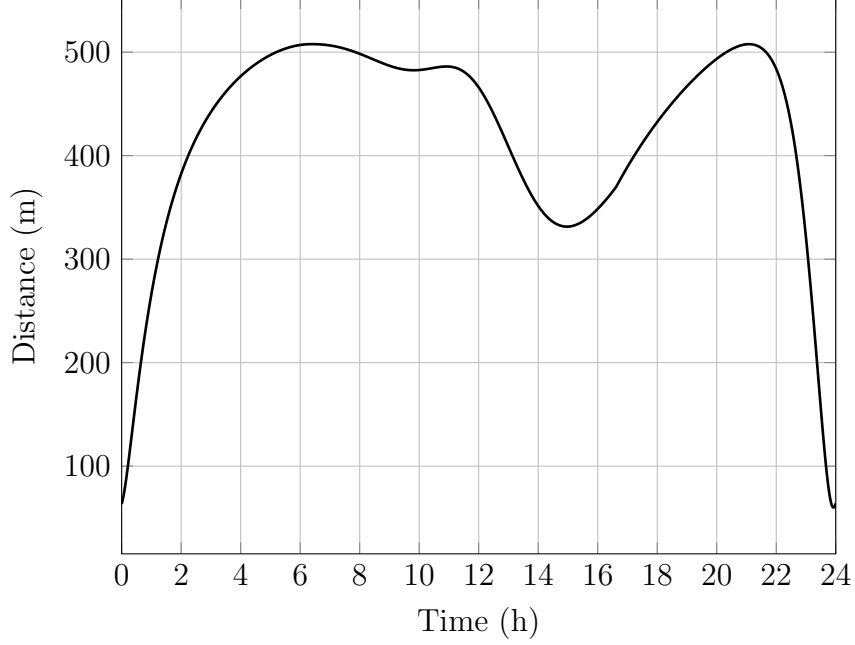


Figure 13: Example trajectory of $dist_{prec}^+(t)$, where the minimizer required 54 function evaluations to determine $t_{\min}$.

For Tokyo, we identified 143 881 instances where $t_{\min}$ occurs close to the border, while in Aachen we encountered such instances 105 547. This explains why more function evaluations are required to minimize the function for Tokyo and Phoenix.

► **Evaluation of Minimization Performance** As described in Section 2.6, the minimization routine may be used up to two times per day in order to determine whether shadow flicker occurs. We evaluate how often a shadow flicker occurrence is detected during the first minimization, how often it is only found during the second minimization, and in how many cases no shadow flicker is detected at all:

Location	Total flicker days	Found on 1st	Found on 2nd	No flicker
Aachen	68,339	45,764	22,575	1,289,461
Phoenix	45,748	25,634	20,114	1,312,052
Tokyo	47,286	31,470	15,816	1,310,514

Table 4: Shadow flicker detection outcomes per minimization iteration.

We observe that in most cases we need to minimize twice over the function to compute the shadow flicker occurrences for a day. This also explains the large number of function evaluations necessary to compute the minima for all days.

► **Reference Performance** To assess the effectiveness of each feature and gain a clearer understanding of the runtime, we evaluate additional metrics by employing the novel approach and use them as a basis for comparison across features. In this case we choose Aachen as the location.

Metric	Value
Brent Minimizer Calls	2,669,836
Function Evaluations (Min)	31,874,089
Brent Root Finder Calls	136,678
Function Evaluations (Root)	2,064,099

Table 5: Performance measurements of the original approach

► **Accurate Root Finding** As we use numerical methods to determine the roots, we introduce a numerical error. In case of the old implementation, the introduced error is 0.05 while we use 10^{-7} in the new implementation. This means that we not only reduced the average runtime by 4 seconds but also made the subroutine more precise.

Metric	Original Approach	Accurate Root Finding	Change
Brent Minimizer Calls	2,669,836	2,669,836	0%
Function Evaluations (Min)	31,874,089	31,874,089	0%
Brent Root Finder Calls	136,678	136,678	0%
Function Evaluations (Root)	2,064,099	1,061,642	-48.6%

Table 6: Comparison between original approach and accurate root finding

We notice that we halved the necessary function evaluations to compute roots of the functions. The following features rely on the robustness of this algorithm, as we now only need a sign change instead of a precisely calculated minimum.

► **Last Day Guess** We notice that this feature reduces the runtime by 17,97 seconds on average across all configurations. That’s a relative reduction of 9,7%.

Metric	Original	Last Day Guess	Change
Brent Minimizer Calls	2,669,836	2,585,637	−3.2%
Function Evaluations (Min)	31,874,089	29,466,143	−7.6%
Brent Root Finder Calls	136,678	136,678	0%
Function Evaluations (Root)	2,064,099	1,076,247	−47.9%
Last Day Guesses Succeeded		63,132	

Table 7: Comparison between the original approach and the last day’s guess

We reduced the function evaluations to compute the minima by 7.6%. Note that the evaluations to compute the roots of the function are different compared to the accurate root finding, as we do not always compute the precise minimum but also use the initial guess.

► **Midpoint Guess** As we do not improve the runtime for days where no shadow flicker was present, it only reduces the runtime by 7.61 seconds, which results in a relative reduction of 4.1%. We notice that the midpoint guess is not as effective as the last day’s

Metric	Original	Midpoint Guess	Change
Brent Minimizer Calls	2,669,836	2,583,633	−3.2%
Function Evaluations (Min)	31,874,089	30,803,908	−3.4%
Brent Root Finder Calls	136,678	136,678	0%
Function Evaluations (Root)	2,064,099	1,063,133	−48.5%
Midpoint Guesses Succeeded		64,907	

Table 8: Comparison between the original approach and the midpoint guess

guess. This can be explained by the fact that the midpoint guess can only be applied if shadow flicker occurred on the previous day, while the last day’s guess also can be used as an initial guess for Brent’s method on days where no shadow flicker occurred.

► **Early Term Minimization** As the new root-finding algorithm is robust and only needs a sign change to evaluate a root, we can early terminate the minimization subroutine in case we find a function value that is negative.

Performance-wise we only get an increase of an average reduction of 2.5 seconds, which means a relative reduction of 1.4% in runtime.

Metric	Original	Stop Early Minimization	Change
Brent Minimizer Calls	2,669,836	2,669,836	0%
Function Evaluations (Min)	31,874,089	31,347,612	−1.7%
Brent Root Finder Calls	136,678	136,678	0%
Function Evaluations (Root)	2,064,099	1,089,031	−47.3%

Table 9: Comparison between the original approach and the stop early minimization condition

The number of necessary function evaluations has been reduced only by 1.7%, as shadow flicker does not occur that much and the algorithm converges fast after finding a negative value.

► **Propagation** By reducing the numbers of receivers for which the shadow flicker duration has to be computed, we can reduce the runtime significantly. In this propagation-based approach we can reduce the runtime by 170.8 seconds, which comes to a relative reduction of 92.6%.

Note that this feature requires the receivers to be aligned in a grid. As buildings and therefore receivers are not aligned in a grid, we do not use it in our case and just show it as an example of how a heuristic that reduces the number of receivers whose shadow flicker occurrences need to be computed can impact the runtime significantly.

An adaptation to other topologies is left as future work.

► **All Features** To obtain the best results for a practical case, we combine all these features. As mentioned, we consider the propagation-based approach as impactful but not general and therefore do not include it here.

We observe that it makes sense to implement all features together, as we improve the runtime by 22 seconds and therefore achieve a relative reduction of 11.9%.

Metric	Original	All Features Combined	Change
Brent Minimizer Calls	2,669,836	2,583,722	−3.2%
Function Evaluations (Min)	31,874,089	29,411,058	−7.7%
Brent Root Finder Calls	136,678	136,678	0%
Function Evaluations (Root)	2,064,099	1,075,556	−47.9%
Initial Guesses Succeeded	—	56,335	—
Midpoint Guesses Succeeded	—	8,738	—

Table 10: Comparison between the original approach and all features combined

We reduced the number of minimizer calls by 3.2% and the necessary function evaluations to compute the minima by 7.7%. The accurate root finding also reduced the necessary function evaluations by 47.9%. It is worth mentioning that most features improve shadow flicker occurrences for receivers who experience shadow flicker. Therefore, their gain is not that significant compared to features like the propagation.

4 Memory Usage Optimization to the Implementation

Among various performance optimizations, memory efficiency is an important aspect of scalability. In this section we are improving the memory usage of the approach. Since the implementation is written in C, we are going to use the data structures and their specifications, like the efficiency of an operation or memory footprint, to guide our design decisions and optimizations.

► **Usage of Methods to Calculate Shadow Flicker Exposure** Depending on the aim, the user might only want to know whether there are receivers whose shadow flicker exposure exceeds a certain threshold for the daily and annual duration. We would then add up the duration daily and also store the receivers who exceeded the defined threshold. It could also be valid to decide when the best timing is to turn off the turbine to maximize, e.g., the energy production while still complying with the regulations. In this case it is necessary to store and return the shadow flicker intervals for each day. In this context we are trying to optimize the case where we need to output the following information for each receiver:

- daily shadow flicker exposure
- annual shadow flicker exposure
- day with the highest shadow flicker exposure

Field	Type	Size
maximum_daily	double	8 bytes
annual	double	8 bytes
intervals[366]	time_interval ($366 \times (8 + 8)$)	5856 bytes
interval_count	int	4 bytes

Table 11: Results are stored in a struct called *sf_results*

► **Current Approach using an Array** Currently we allocate such a struct for each receiver to calculate the shadow flicker exposure per year in an array. We therefore index every receiver based on its position in the grid.

For each receiver and every day we allocate memory, although the data is sparse, meaning that we do not have a lot of non-zero durations across a year. There might be even a large number of receivers who do not experience any shadow flicker throughout the whole year. This approach is also not very performance efficient either, since we have to evaluate all entries, even if on most days no shadow flicker occurs.

► **Requirements for Another Data Structure** (1) For the novel approach, we take a receiver and iterate over each day of the year to calculate its daily shadow exposure. This means we need a data structure where we can iterate efficiently over the neighbors.

(2) We do not need to allocate storage for each day, since there are days when no shadow flicker occurs.

Therefore, we would prefer to use a dynamic data structure, meaning that memory is allocated solely for intervals in which shadow flicker occurs .

(3) After calculating the durations, we need to be able to iterate over the calculated daily durations and write the output.

(4) Especially in the case of our propagation-based search, we need to be able to add nodes at the end since we calculate the duration per day, not per receiver .

► **Singly-Linked List** An example of a dynamic data structure is a singly linked list. We can dynamically assign nodes when we encounter days where shadow flicker duration occurs, fulfilling (2). We can also iterate over all elements, satisfying (1) and (3). Since we can only traverse elements from the start to the wanted element, we need $\mathcal{O}(n)$ iterations to find it, but if we also keep track of the tail, we can efficiently add elements in $\mathcal{O}(1)$ to the back to satisfy (4).

But the linked list might also come with disadvantages. Since we allocate one node at a time rather than a contiguous block as with arrays, this might lead to memory fragmentation, which in turn results in poor cache locality and increased cache misses during iteration [12].

In terms of performance, we need to maintain a pointer to the head and the tail of the list. In addition, we need one pointer per added element, which means that we need 2 pointers, 8 bytes each, and additionally one pointer for each day with a nonzero shadow flicker duration.

► **Sparse Representation** We observe that for the majority of the receivers in the grid and the majority of days, no shadow flicker occurs, indicating that the data is in fact sparse.

This relation to data can be observed in areas like machine learning, where there are constructed large matrices where there are only a very few entries that deviate from a default value, e.g., 0.

For example, if we encounter this array: $[0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 5]$, we would store $[(4,4), (10,5)]$ as a sparse representation [13] and would not store any zeroes. In fact, the linked list is also a sparse representation, as we also only store nonzero durations.

► **Spatial Array** One tradeoff we can consider is to use the efficient lookup of the array but also use the sparse representation by declaring an array of pointers per day. The pointers are initialized with null, and we would calculate shadow flicker day by day. In case we find a day whose shadow flicker duration is nonzero, we would allocate a

time_interval to store the occurrence.

Requirements (1), (3), and (4) are satisfied since we can access each entry using its index. As we only allocate a *time_interval* in case we find a day with nonzero duration, we fulfill (2). The memory necessary for a pointer is 8 bytes, and the memory for a *time_interval* is 16 bytes. Therefore, the sparse pointer-based approach is more memory-efficient if shadow flicker occurs on fewer than half the days.

► **Case Study** To evaluate which data structure works best in this case, we are going to use the case study from Section 2.6 and determine whether how the choice of the data structure influences the performance of our approach.

In the case study we evaluate the shadow flicker duration for $61 \cdot 61 = 3721$ receivers.

We tracked that there were 68339 occurrences of shadow flicker over the whole year.

This means that per receiver we encountered $\frac{68339}{3721} = 18,37$ on average per receiver.

The receiver that experienced the most shadow flicker occurrences encountered 173 days with shadow flicker over the whole year.

In terms of memory consumption, we get the following result per receiver for our case study:

► **Array**

- `sf_results`: 5,856 bytes per receiver
- Total memory: 21 megabytes for the whole case study

► **Linked List**

- `annual`: double (8 bytes)
- `interval_count`: int (4 bytes)
- `maximum_daily`: double (8 bytes)
- $19 \times (\text{pointer} + \text{interval})$: $19 \times (8 + 16) = 456$ bytes
- Head and tail pointers: $2 \times 8 = 16$ bytes
- Total: 480 bytes per receiver
- Total memory: 1.8 megabytes for the whole case study

► **Sparse Array**

- `annual`: double (8 bytes)
- `interval_count`: int (4 bytes)
- `maximum_daily`: double (8 bytes)

- Pointer array: $365 \times 8 = 2,920$ bytes
- Time intervals: $19 \times 16 = 304$ bytes
- Total: 3,232 bytes per receiver
- Total memory: 12 megabytes for the whole case study

We conclude that switching to a sparse array halves the required storage, and switching to a linked list reduces the storage consumption by 90%.

► **Validation** To exclude that switching the data structure has no negative impact on the runtime, we implemented each data structure, executed the program 5 times, and measured the runtime of each run.

- array: $\{156.59s, 155.05s, 155.10s, 154.88s, 154.87s\} \implies \bar{x} = 155.3s$
- linked list: $\{155.82s, 156.15s, 157.22s, 155.51s, 155.59s\} \implies \bar{x} = 156.06s$
- sparse array: $\{155.09s, 155.34s, 156.29s, 155.22s, 155.31s\} \implies \bar{x} = 155.45s$

In addition, we measured the memory consumption every second. The figure shows how it changed over time:

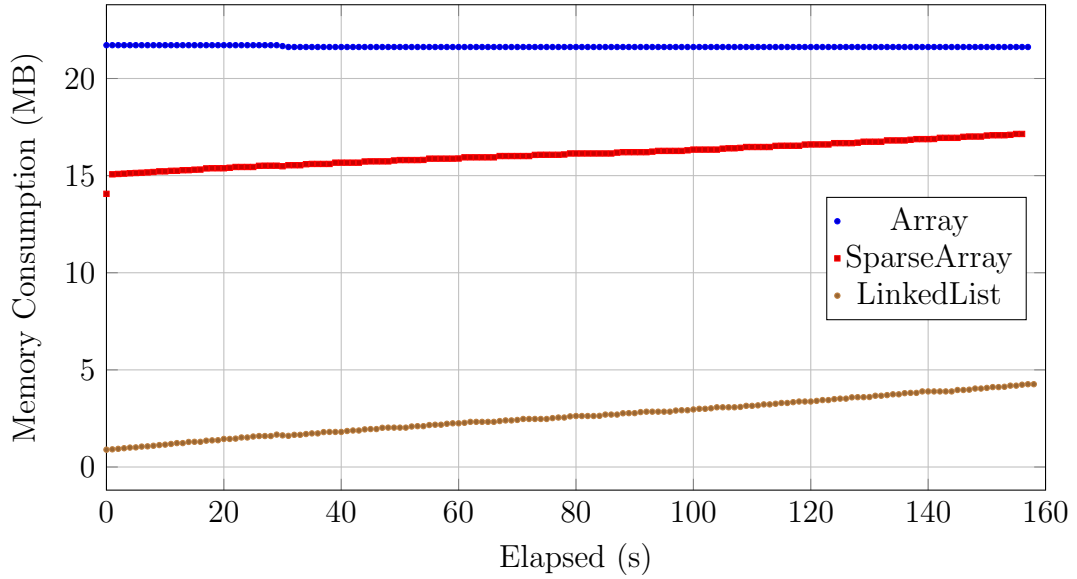


Figure 14: Memory consumption between the three data structures over time measured per second over a whole run

First, we note that the theoretical computations only accommodate for the storage of results. We need additional storage for internal computations, such as the Solar Position Algorithm or Ray Tracing Algorithm.

We notice that the array consumes 21.621 *MB* across the whole run, as we allocate the whole memory at the start.

For the sparse array, we allocate 14.063 *MB* at the start, which rises up to 17.145 *MB* linearly.

The linked list allocates 0.892 *MB* at the start, which grows up to 4.262 *MB*. At the end of the run we free all storage that has been allocated. In case we output the data externally, it depends on how detailed the results are and in which format they are stored.

We conclude that there is no significant difference in execution time when using those 3 data structures. Therefore, we use the linked list, as we can reduce the storage consumption by 90%.

Multiple Turbine Calculation

Having improved the performance and scalability of the novel approach, we now proceed to implement it in a practical setting, where there are multiple turbines present instead of just one.

Problem Statement

Given multiple wind turbines characterized by their geographic location and rotor diameter, a specified receiver location, and the necessary input parameters for the SPA (Sun Position Algorithm), determine the total annual shadow flicker duration, expressed in minutes, experienced at the receiver.

Calculation using the Novel Approach

Assuming we have one receiver r and n turbines t_1 to t_n .

First, we compute the shadow flicker exposure for each receiver-turbine pair (r, t_1) to (r, t_n) . It is not sufficient to just add up the duration, since there might be instances where shadow flicker exposure occurs by more than one turbine at the same time.

► **Implementation** We obtain the intervals $I_1, \dots, I_n$ after computing the shadow flicker exposure for each receiver-turbine pair. Note that those intervals might overlap. To compute the union of intervals:

1. sort all intervals by their start time ($\mathcal{O}(n \log n)$);
2. iterate over the intervals and merge overlapping intervals ($\mathcal{O}(n)$).

The procedure returns the total shadow-flicker duration for the day:

$$D(d, r) = \sum_{j=1}^m (\tilde{t}_{\text{end}}^{(j)} - \tilde{t}_{\text{start}}^{(j)}),$$

where $\tilde{I}_1, \dots, \tilde{I}_m$ with $\tilde{I}_i = [\tilde{t}_{\text{start}}^{(i)}, \tilde{t}_{\text{end}}^{(i)}]$ are the merged, pairwise-disjoint intervals produced by step 2. Since we calculate the shadow flicker exposure for receiver-turbine pairs, we use all optimizations except the propagation (Section 3.9).

Coordinate Transformation

In the original approach the coordinates of the receiver are given relative to the turbine position. In addition, we need to determine the ground height for both receivers and turbines. First we need to select a central point for which we want to calculate the sun's position and also provide the inputs required for the SPA algorithm.

► **Geodetic** Coordinates used in common GPS coordinates

The coordinates are described in longitude (λ) and latitude (ϕ)

► **Earth-Centered, Earth-Fixed Coordinates** We first convert the *geodetic* coordinates into *ecef* coordinates.

The idea is to use the planet's exact center of mass as the origin. We define 3 perpendicular projections (x, y, z).

- x -axis passes through the Equator at the Greenwich meridian
- y -axis passes through the Equator towards the Indian Ocean
- z -axis passes through the North Pole

This frame is right-handed and also rotates with the earth so that every location is fixed. The advantage is that a point's coordinates are the orthogonal projections onto these axes, expressed in meters [14].

► **East-North-Up** Now we want to choose an origin reference point, in our case the position for which we calculate the position of the sun.

From that reference point, every other point can be expressed in relation to that reference point. For that we used a flattened frame in 3 directions:

- East
- North
- Up

As we require precise height data, we drop the up coordinate for our implementation and just use the (East, North) coordinates as our custom map [15].

Correctness

Since we want to make sure that our implementation works as intended, we again take a similar approach as in Section 3.10.

For that, we take receivers that are present in the case study as well as the same wind turbine and calculate the shadow flicker exposure for this setup. Since we get the same output compared to the original approach implementation, we assume our implementation works correctly.

5 Precomputation

Usually a practical goal is to optimize the layout of a wind park, which consists of multiple wind turbines, such that the energy production is maximized while adhering to technical, environmental, and regulatory constraints.

Each turbine's performance can also be influenced by local wind conditions or the interaction of turbines (wake effects). Therefore, a central challenge lies in balancing efficiency and feasibility.

As there are many factors to consider, a lot of different layouts have to be considered as well.

5.1 Wind Farm Layout Optimization

To solve this wind farm layout optimization problem, there are multiple real-world projects such as WiFO, TOPFARM, or WindPRO.

Since such projects also need to comply with governmental regulations, shadow flicker exposure has to be considered as well.

To be more precise, we must not exceed the daily and annual exposure limits. Else the turbines that cause the shadow flicker for the receiver need to be shut down for the time frame in which shadow flicker occurs, which might lead to suboptimal productivity.

The complexity also scales high with the number of turbines, landscape constraints, and number of other decision variables such as turbine height or rotor diameter.

To calculate optimal solutions that account for all the described factors, a very large number of iterations for different layout evaluations are needed, depending on the complexity of the chosen models.

5.2 Precomputing using Shadow Maps

In Section 3.9, we implemented a way to efficiently calculate the shadow flicker map of a wind turbine for a given sun position. We make the assumption that if we move a wind turbine in a radius of 20 kilometers, the shadow flicker map does not change significantly.

By this we would need to compute an accurate shadow map once for each turbine type in case rotor diameter or height changes and then just look up the necessary shadow flicker durations. The lookup can be implemented efficiently, therefore saving a lot of computational effort compared to recalculating the shadow flicker exposure in every iteration.

► **Problem Statement** Given the SPA algorithm parameters, the turbine configuration, and the shadow-map configuration, precompute shadow maps for the specified receiver height layers. For each receiver, determine its position relative to the turbine and obtain the shadow-flicker duration from the nearest grid point. If the receiver height does

not coincide with a precomputed layer, interpolate the duration from the surrounding shadow-map layers to estimate its shadow-flicker value.

5.3 Methodology

We separate the program into two parts, which both require a *config_file*.

The user needs to specify for which position the location of the sun should be computed, as it depends on the elevation and azimuth angle. For precise results the location should be chosen centrally inside the area in which the turbine can be placed so we only introduce a negligible error. The parameters for the sun position have to be provided as well. In addition, the configuration of the turbine (height and rotor radius) and the dimensions of the shadow map have to be included.

- $x_{min}, x_{max}, y_{min}, y_{max}$: scale of the map
- *step_size*: how far receivers are placed apart
- *num_height_layers*: number of vertical layers at which shadow profiles are computed
- *min_height, max_height*: minimum and maximum receiver heights relative to the turbine

The result of this process is an array that includes the daily intervals as a linked list and the annual duration for each receiver and each layer.

For the second part, we need another *config_file*, which specifies the actual location of the turbines and receivers in geographic coordinates, including the height. To distinguish between receivers, we assign each one a unique identifier. Based on the geographic coordinates, the relative position to the turbine is calculated. Subsequently, we calculate the closest grid point in the shadow map and then interpolate across the heights if necessary. If the receiver lies outside the shadow-map domain, its shadow-flicker duration is set to zero. (Alternatively, one could mark such cases as “no data available.”).

► **Additional Complexity of Unknown Height** As we do not know the precise height of the turbines or the receivers in such an optimization scenario, it is important to calculate the relative position of the receiver to the rotor of the wind turbine to calculate the shadow flicker exposure.

Since it is computationally infeasible to calculate the shadow map exposure for every height, we need to specify in what height range the receiver can be compared to the turbine’s position.

5.4 Interpolation

To solve the problem of the unknown height of the receiver, we can apply interpolation. There are different interpolation methods available, such as linear, polynomial or spline

interpolation.

We going to select the method that gives us the most precision for our use case. Note that we introduce two types of discretization errors:

Firstly, we precompute the shadow maps in a grid with a certain step size. Additionally, we introduce an error since we use interpolation to estimate the duration for an unknown height based on the closest shadow maps.

► **Linear Interpolation** This technique works efficiently but is not precise. If Δh is the distance between two shadow maps, the introduced error is $\mathcal{O}(\Delta h^2)$ [16].

Assuming we want to estimate the duration d for a node given by the coordinates x, y, h . We look for the nearest neighbor in the grid $x_{\text{step}}, y_{\text{step}}, h_{\text{step}}$ and are given $h_a < h_{\text{step}} < h_b$, where h_a, h_b are the heights of the closest shadow maps.

Now the durations $d_a = d(x_{\text{step}}, y_{\text{step}}, h_a)$ and $d_b = d(x_{\text{step}}, y_{\text{step}}, h_b)$ are given using the precomputed shadow maps.

To calculate the duration $d = d(x, y, h)$, we apply the formula for linear interpolation:

$$d(x, y, h) \approx d(x_{\text{step}}, y_{\text{step}}, h_{\text{step}}) = d_a + (h_{\text{step}} - h_a) \frac{d_b - d_a}{h_b - h_a} \quad (12)$$

► **Spline Interpolation** A more accurate method for our use case is spline interpolation. Assuming we have n shadow maps with heights h_0 to h_m . We use low-degree polynomials in each of those intervals $[h_i, h_{i+1}]$ and choose polynomial pieces so that the resulting function is smooth. The resulting function is called a "spline" and can be evaluated efficiently [16]. We use an implementation to interpolate the durations in our case study [17].

► **Interpolation of Intervals** An interval that describes shadow flicker can be given by $[t_{\text{start}}, t_{\text{end}}]$. Instead of interpolating the duration of such an interval, we could interpolate t_{start} and t_{end} individually. By this we can not only provide the interpolated annual duration but also the daily shadow flicker intervals. We need to consider a special case. Interpolating between two adjacent layers, where one has a shadow flicker occurrence and one does not, would result in a wrong result.

Therefore, we can only interpolate between non-zero intervals or when two intervals where no shadow flicker occurs are adjacent to each other.

5.5 Correctness and Validation

As we already validated the correctness of the shadow maps with respect to the original approach. In this case we now need to show how big the introduced error is through the interpolation. The error introduced by using the nearest neighbor in the grid depends highly on the chosen step_size. We are going to use our case study, using 6 shadow maps between 0 to 30m.

To show how big the introduced error is, we are going to interpolate the shadow map for $h = 2.5m$ and also precisely calculate it. The following map illustrates the error:

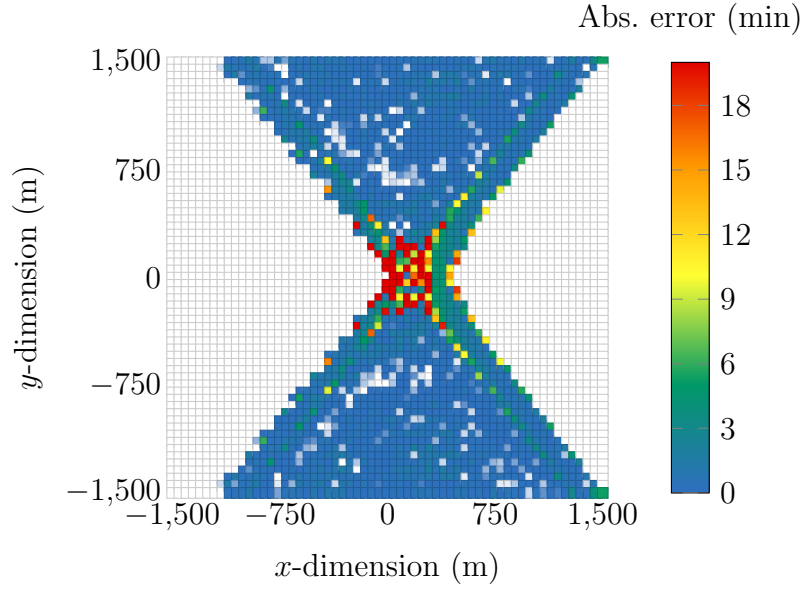


Figure 15: Error between computed shadow flicker duration and interpolated value

In the white regions, no shadow flicker occurred, or the interpolation estimated the duration very precisely.

At the borders, inaccuracies appear because the method interpolates between zero durations and nonzero durations, which leads to sharp transitions.

In the center, the absolute error values are high and vary strongly between height layers. This explains the occurrence of the red regions close to the turbine, where the model struggles most to represent the rapid changes in shadow flicker.

6 Conclusion

This thesis extended a previous method in several aspects. The goal was to improve the scalability and usability.

First we applied various optimization strategies to reduce the computational effort. We notice that the expensive part to compute the shadow flicker duration is the minimization subroutine.

Strategies such as the propagation or prior day's guess reduced the computational effort in the minimization subroutine the most.

Replacing the root-finding algorithm allows for more precise results while also reducing the necessary function evaluations by half. Other features that allow us to exit the minimization earlier or the midpoint guess offer improvement only if shadow flicker occurs and are beneficial as well but did not improve the runtime tremendously.

In addition, we noticed that on most days no shadow flicker occurs. Therefore, we switched the static data structure, which allocates memory for every day, to one that only allocates memory in case shadow flicker occurs. By this we could save the storage consumption for results up to 90%.

To increase the usability of the approach, we extended the method to support the shadow flicker computation for multiple turbines. For correct results, we need to handle the overlap of intervals from different turbines. In addition, we also extended the method to use geographic coordinates.

The last part of the thesis focuses on the precomputation of the shadow flicker durations. For that we precomputed shadow maps, calculated the relative coordinates compared to the turbine in the precomputed grid, and used interpolation to accommodate for unknown receiver heights.

We observe that the method works precisely in most cases but can be imprecise in close vicinity of the turbine or close to the border of a shadow map.

It must be considered that we introduce two types of discrete errors due to interpolation and grid snapping. Using these results as input to an optimizer is a reasonable choice, but the final solutions should be validated with a more precise calculation to ensure accuracy.

Despite these advantages, the continuous approach still has limitations. The minimization subroutine can only compute one shadow flicker occurrence per day, and interpolation near turbine proximity or shadow map borders introduces errors. Moreover, the model relies on simplifying assumptions such as the rotor orientation, which reduce realism in practical applications. Altogether, the results show that continuous methods can make shadow flicker analysis both more precise and more efficient, thereby offering a viable alternative to discrete models currently used in industry.

7 Future Work

As this approach offers a better runtime and more precise results compared to the discrete approach, which is used in current tools such as WindPRO. Therefore, it would be beneficial to integrate this method into tools that require the computation of shadow flicker durations.

Further research should address the minimization subroutine. Extending the algorithm to reliably capture multiple flicker intervals per day would make it robust. In addition, we encountered difficult instances for the current minimization process. We suggest optimizing the subroutine in general to make the method more efficient.

Research should also address the accuracy of interpolation, particularly near turbine proximity.

Heuristics that would reduce the number of receivers or the number of minimization steps necessary would greatly improve the efficiency. They could be based on the latitude of the receiver, placement of the turbine, or receiver.

Besides vertical interpolation across heights, future work could also explore interpolation between different grid points to increase robustness.

In this thesis we compute geometrical shadow flicker, i.e., the maximum possible shadow flicker under clear-sky conditions. Meteorological shadow flicker, which accounts for actual weather and sunshine statistics, would also increase the precision of the results.

References

- [1] Knight, C., Nuno, A., and Robotti, C. *What is the future of wind power?* Essay, European Investment Bank. 2024. URL: <https://www.eib.org/en/essays/what-is-future-wind-power>.
- [2] WindEurope. *Wind energy in Europe: 2024 Statistics and the outlook for 2025–2030*. Report from the WindEurope Intelligence Platform. WindEurope, 2025. URL: <https://windeurope.org/intelligence-platform/product/wind-energy-in-europe-2024-statistics-and-the-outlook-for-2025-2030/>.
- [3] Haac, R. et al. “In the Shadow of Wind Energy: Predicting Community Exposure and Annoyance to Wind Turbine Shadow Flicker in the United States”. In: *Energy Research & Social Science* 87 (2022), p. 102471.
- [4] LAI – Bund/Länder-Arbeitsgemeinschaft Immissionsschutz and LUBW. *Hinweise zur Ermittlung und Beurteilung der optischen Immissionen von Windkraftanlagen*. Germany’s shadow-flicker guideline: max 30h/year, 30min/day. 2020. URL: https://www.lai-immissionsschutz.de/documents/wka_schattenwurfhinweise_stand_23_1588595757.01.
- [5] EMD International A/S. *WindPRO SHADOW module*. Version 4.0. Shadow-flicker calculation and visualization tool. 2025. URL: <https://www.emd-international.com/software/windpro/modules/shadow>.
- [6] DNV International AS. *WindFarmer: Environment module (shadow flicker analysis tool)*. Version latest available. Optional Environment module providing shadow flicker and noise impact assessment. 2025. URL: <https://www.dnv.com/software/services/windfarmer/>.
- [7] Radke, N., De Smet, P. E. M., and Abraham, E. “Numerical Modeling and Analysis of Shadow Flicker Using Solar Path Functions for Enhanced Predictive Accuracy”. In: *Energies* 18.2 (2025), p. 352.
- [8] Meeus, J. H. *Astronomical Algorithms*. Richmond, VA, USA, 1991.
- [9] Reda, I. and Andreas, A. “Solar Position Algorithm for Solar Radiation Applications”. In: *Solar Energy* 76.5 (2004), pp. 577–589.
- [10] Jenkins, A. “The Sun’s position in the sky”. In: *European Journal of Physics* 34.3 (2013), pp. 633–652. URL: <https://iopscience.iop.org/article/10.1088/0143-0807/34/3/633>.
- [11] Brent, R. P. and Burkardt, J. *brent.c — Brent’s Root-Finding Algorithm (C version)*. https://people.math.sc.edu/Burkardt/c_src/brent/brent.c. C translation of Brent’s algorithm for scalar root finding. 2019. (Visited on 08/06/2025).
- [12] Sonntag, B. and Colnet, D. “RIP Linked List: Empirical Study to Discourage You from Using Linked Lists Any Further”. In: *arXiv preprint arXiv:2306.06942* (2024). Version 3. arXiv: 2306.06942 [cs.DS]. URL: <https://arxiv.org/abs/2306.06942>.

- [13] Davis, T. A. *Direct Methods for Sparse Linear Systems: SIAM Short Course Slides*. Slide deck, 25 pages. Slide 4. 2006. URL: https://people.engr.tamu.edu/davis/research_files/siam06_short.pdf.
- [14] Zhu, J. “Conversion of Earth-Centered Earth-Fixed Coordinates to Geodetic Coordinates”. In: *IEEE Transactions on Aerospace and Electronic Systems* 30.3 (1994), pp. 957–961.
- [15] Hofmann-Wellenhof, B., Lichtenegger, H., and Collins, J. *GPS: Theory and Practice*. 5th ed. Vienna; New York, 2001.
- [16] Rashid, H. H. *Interpolation, Extrapolation, and Inverse Interpolation*. Lecture notes for “Numerical Analysis,” 1st semester, Mustansiriyah University. 2019. URL: https://uomustansiriyah.edu.iq/media/lectures/9/9_2019_10_10%2110_16_06_AM.pdf (visited on 08/05/2025).
- [17] Press, W. H. et al. *Numerical Recipes in C: The Art of Scientific Computing*. 2nd ed. See §3.3 for natural cubic spline interpolation algorithm. Cambridge, United Kingdom, 1992.