Diese Arbeit wurde vorgelegt am
Lehr- und Forschungsgebiet Theorie der hybriden Systeme

Modellierung und Simulation von Tilt-and-Roll Heliostaten in Solarturmkraftwerken

Modeling and Simulation of Tilt-and-Roll Heliostats in Central Receiver Systems

Bachelorarbeit
Informatik

Mai 2024

Vorgelegt von Presented by	David Elias Walter Schütte Matrikelnummer: 422572 david.schuette@rwth-aachen.de
Erstprüfer First examiner	Prof. Dr. rer. nat. Erika Ábrahám Lehr- und Forschungsgebiet: Theorie der hybriden Systeme RWTH Aachen University
Zweitprüfer Second examiner	Prof. Dr. rer. nat. Torsten Kuhlen Lehr- und Forschungsgebiet: Virtuelle Realität & Immersive Visualisierung RWTH Aachen University
Betreuer Supervisor	Dr. rer. nat. Pascal Richter Lehr- und Forschungsgebiet: Theorie der hybriden Systeme RWTH Aachen University

Contents

1 Introduction

As the impacts of climate change become increasingly apparent, renewable energy sources have taken on greater significance than ever before. In Germany, 50 % of electrical energy is currently generated from renewable sources, with the target to increase this figure to 80 % by 2030[1].
However, several challenges must be addressed to achieve this goal. One major concern is the intermittency of most renewable energy sources. To overcome this issue, central receiver systems (CRS) offer a promising solution for storing thermal energy and generating electricity even on cloudy days or at night. CRS technology uses mirrors mounted on heliostats that reflect sunlight onto a receiver, as depicted in Figure 1, which shows the Gemasolar plant in Spain with its 2650 heliostats. The concentrated radiation heats a liquid, causing it to boil water and power an electricity-generating turbine. The liquid can also directly be used in industrial processes.
The design and construction of such systems is a complex and costly process. Multiple factors (e.g. location and weather) can significantly impact the plant's efficiency, emphasizing the importance of accurate energy production simulation and optimization, before its implementation. Optimizing a new CRS upfront is significantly more cost-efficient than attempting to fix issues after construction. However, this process relies on an accurate physical model that accounts for all relevant variables. Moreover, due to the numerous variants of CRS that are typically tested during optimization, the models must also be computationally efficient. Substantial advancements have been made in the research of modeling and optimizating aspects of CRS design, ensuring a more accurate and effective approach to this complex process.

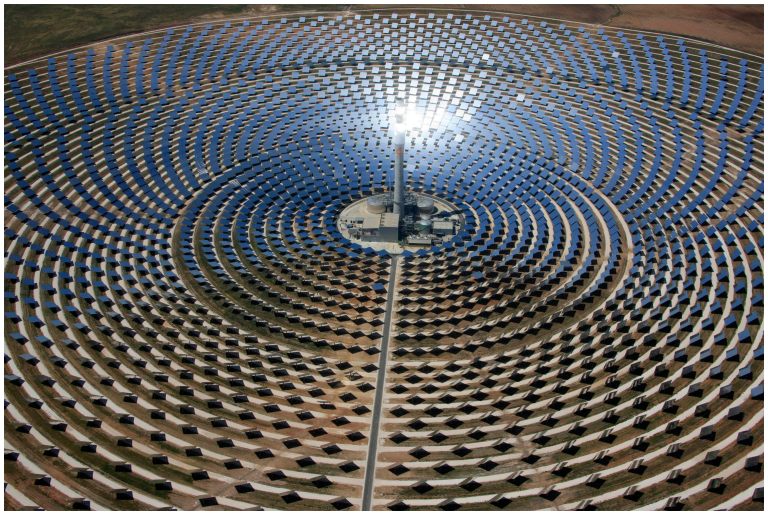


Figure 1: Central Receiver System "Gemasolar" in Seville, Spain. The power plant uses a single cylindric external receiver and 2650 heliostats. Taken from [6].

[1] https://www.bmwk.de/Redaktion/DE/Dossier/erneuerbare-energien.html

1.1 Motivation

The requirements for central receiver systems (CRS) have changed in recent years[2]. Most CRS have been built to last for decades in the same space, producing renewable energy during their lifetime. This requires extensive landscaping and high initial development costs. For temporary installations (e.g. a mine) that require large amounts of heat or electricity, transporting this power to the site can be challenging. A new approach is a mobile central receiver system (mCRS). This kind of system also consists of receivers and heliostats like conventional CRS, but focuses on flexibility. Power plants can be relocated without much effort to meet the local energy demand that is only present for a limited time. These heliostats are called **Tilt-and-Roll heliostats (T&R)** and are no longer big structures mounted into the ground with huge filaments made of concrete, but rather small mirrors that are placed on the surface. This allows mCRS to be redeployed to different locations where a renewable source of heat is needed.
The heliostats used by mCRS are much smaller and lighter compared to conventional heliostats. They also incorporate different shapes and mounting mechanisms, resulting in new rotational axes. This changes the behavior of tracking algorithms used for these heliostats. Tracking algorithms determine the angle at which the heliostat is aligned with the sun and receiver to properly reflect sun rays. The mirrors are smaller and inexpensive compared to heliostats in CRS.
Existing simulation tools only consider CRS and do not allow the simulation of Tilt-and-Roll heliostats. Simulation tools can significantly reduce the cost and time it takes to develop mCRS. We need to extend the existing heliostat model to be able to create a digital twin of the Tilt-and-Roll heliostat used in mCRS. Rather than implementing the exact heliostat shape, we opt for a flexible approach by allowing to define heliostat facets as irregular polygons. This will enable easy testing of new shapes without a need for additional programming. Another feature required for the simulation of Tilt-and-Roll heliostats is the option to configure blind facets. Such facets do not reflect any sunlight as they are not made of a reflective surface, but rather miscellaneous objects such as servo motors or other mounting equipment. We opt to implement a configurable reflectivity for each facet that can be set to zero to disable ray generation for that facet.

1.2 Related Work

The original *SunFlower* ray tracer was introduced by Richter et al. in 2007 [19]. It implements simulation and optimization tooling for the development of CRS. Simulation considers the annual optical output of the complete power plant. There is also the option to configure non-optical aspects of the CRS like electricity or heat production. An in-depth overview is given by Hövelmann [13]. We will focus on the optical model of the simulation. Optimization of CRS was implemented by Fischer [10] and overhauled by Hövelmann [13]. The simulation of *SunFlower* uses bi-directional ray

[2] `https://www.heliogen.com/technology/#heliostats-pg`

tracing with an analytic or Monte Carlo approach.
There exist other Monte Carlo ray tracers for CRS. We differentiate between forward tracing and bi-directional tracing. Forward tracing generates rays on a plane above the power plant site mimicking the sun. These rays are then traced and checked if they hit a heliostat. Bi-directional tracing generates rays on the mirror surface and traces them towards the sun and towards the receiver. *Tonatiuh* [4], *MIRVAL* [16] and *SolTrace* [23] are forward tracing Monte Carlo ray tracers. *STRAL* [2] and *TieSol* [14] are successors to the former and utilize bi-directional ray tracing. Due to the nature of Monte Carlo simulations, the run time of such ray tracers is much higher than those of analytic approaches. To improve the run time of Monte Carlo ray tracers, GPU implementations were introduced. The initial GPU realization for *SunFlower* was done by Aldenhoff [1] and improved by Hövelmann [13]. *TieSol* [14] also uses GPUs to speed up the simulation. Other GPU-based ray tracers include *sbpRAY* [11] and *QMCRT* [7]. All of the existing CRS simulation tools can only simulate azimuth-elevation tracking heliostats, but not Tilt-and-Roll heliostats.

1.3 Contribution

The development of CRS and mCRS is a time and cost-intensive process. To facilitate this process, digital simulation can be used. In this thesis, we introduce new functionality to enable the simulation and optimization of mCRS using Tilt-and-Roll heliostats. To do so, the option to use irregular polygons is implemented. This is a flexible approach to enable fast-paced development of mCRS without the need for new programming. Furthermore, we introduce a novel algorithm that solves the additional requirements by Tilt-and-Roll heliostats. The AABB tree, used extensively during simulation, is adjusted to support both heliostat types. The measurement-driven approach further increases the accuracy of simulations by using a scan of an actual heliostat.

1.4 Outline of this Work

In this thesis, we mainly consider the optical model of (m)CRS. The aspects of the optical model without heliostats will be described in Section 2. We will touch on the different receiver types, environmental influences and sun shapes. In Section 3, we follow up by introducing two heliostat models, an azimuth-elevation tracking and a Tilt-and-Roll heliostat. For these models, we show different aiming strategies and tracking algorithms. An explanation of relevant parts of the ray tracers is given in Section 4. This includes the analytic and Monte Carlo ray tracer, AABB trees and measurement-driven simulation. To validate our findings, we discuss multiple case studies in Section 5. This includes the correctness and performance of our implementation. A final summary of our findings is given in Section 6.

2 Optical Model

The optical model consists of various aspects and allocates the most computation time for a simulation of central receiver systems. It is comprised of a predefined area in which all heliostats are located, a receiver, on which all heliostats project solar radiation and heliostats which consist of large mirror areas to project sunlight onto the receiver. The location and environment of the CRS play an important role as it determines the position and intensity of the sun. Molecules in the air and shading or blocking by other heliostats can reduce the efficiency of one heliostat and introduce optical losses. Multiple sun shapes are used to approximate the sun. We will discuss the heliostat model in detail in Section 3.

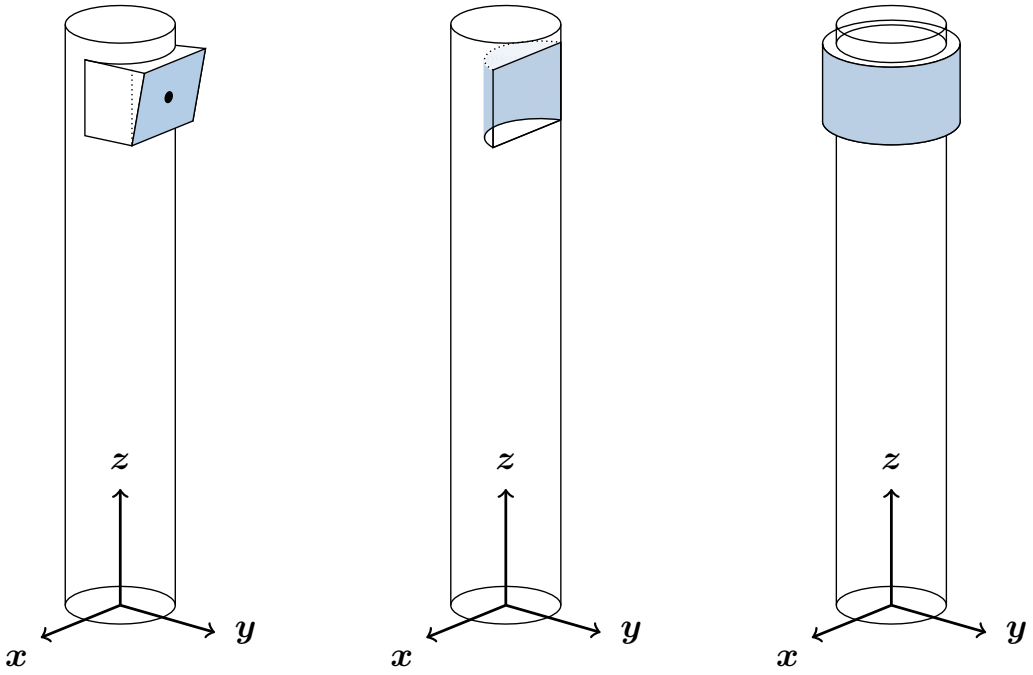
Environmental Influences The location of the CRS can be defined by multiple parameters. These include the boundaries of the site and restricted areas within this site. This is especially useful for the optimization of heliostat layouts.
All objects are placed according to a Cartesian coordinate system with positive x pointing toward the east, positive y pointing towards the north and positive z pointing towards the sky. The topography can be generated using data from the Space Shuttle Radar Topography Mission [9]. The sun vector needed for simulation can be computed using the sun's azimuth γ_{solar} and altitude angle θ_{solar} as follows:

$$\vec{\tau}_{solar} = \begin{pmatrix} \sin\left(-\gamma_{\text{solar}}\right) \cdot \left(-\cos\left(\theta_{\text{solar}}\right)\right) \\ \cos\left(-\gamma_{\text{solar}}\right) \cdot \cos\left(\theta_{\text{solar}}\right) \\ \sin\left(\theta_{\text{solar}}\right) \end{pmatrix}. \tag{1}$$

In addition, direct solar irradiation I_{DNI} [3] can be calculated directly using the Meteorological Radiation Model (MRM) [15].

Receiver Receivers are towers with receiver panels mounted at the top to collect the projected solar radiation (flux) by the heliostat and convert it into heat. To reduce the shading and blocking effect of heliostats with other heliostats, the towers have a predefined height. There are different types of receivers used in CRS, we differentiate between the following three. A flat tilted receiver (Figure 2a) is defined by a tilted rectangular receiver panel mounted at the top of the tower. A cylindric cavity receiver (Figure 2b) is recessed into the tower. This optimizes the incoming angle of sun rays. Cylindric external receivers (Figure 2c), like the one used in Gemasolar [6], wrap around the top of the tower and can collect solar radiation from all directions.

Optical Losses There are different sources for the loss of solar radiation. We will discuss some of them in the following.
The **cosine effect** is a result of the alignment of the heliostat to reflect sun rays onto the receiver panel. As the heliostat is rotated and tilted, the area perpendicular to the sun vector is reduced by the cosine efficiency. $\vec{\tau}_{\text{solar}}$ describes the sun vector and $\vec{n}$



(a) Flat tilted receiver (b) Cylindric cavity receiver (c) Cylindric external receiver

Figure 2: Common receiver types available in central receiver systems. Taken from [13].

the normal vector of the heliostat or respective facet. All vectors are normalized. The calculation is as follows:

$$\eta_{\cos} = \vec{\tau}_{\text{solar}} \cdot \vec{n}. \tag{2}$$

A major impact on the efficiency of a CRS can be attributed to shading and blocking. Each heliostat can throw shade on other heliostats or block a reflected ray of another heliostat. In addition, the tower can also throw shade onto heliostats depending on the sun's position. For cavity receivers, rays can be blocked by the ledge of the indentation. The calculation of shading and blocking is the most computationally intensive part of simulating a CRS in *SunFlower*.
Each heliostat that reflects solar radiation will also absorb a part of the radiation and is subject to diffuse reflection that does not hit the intended receiver panel. In literature, this is often modeled by a constant value [18]. We use a configurable reflectivity value for each facet.
Another loss can be attributed to **atmospheric attenuation**. This is the result of sun rays interacting with molecules in the air and thus a reduction in solar radiation. We derive the **atmospheric attenuation** using a formula from Schmitz et al. [21] in which d describes the distance between heliostat and receiver. The formula reads as

follows:

$$\eta_{\text{aa}} = \begin{cases} 0.99321 - 1.176 \cdot 10^{-4} \cdot d + 1.97 \cdot 10^{-8} \cdot d^2 & , d \leq 1000\,\text{m} \\ \exp(-1.106 \cdot 10^{-4} \cdot d) & , d > 1000\,\text{m} \end{cases}. \tag{3}$$

Determining the ideal reflected ray during a simulation is not a difficult task. This will be shown in Algorithm 1. In practice, however, there are several possible errors involved in aligning the heliostat according to the actual sun position and receiver. The larger the distance between the receiver and the heliostat, the bigger the impact of the alignment error. We differentiate between two errors: the horizontal and vertical tracking errors. Both errors are modeled using Gaussian distributions [13, 20].

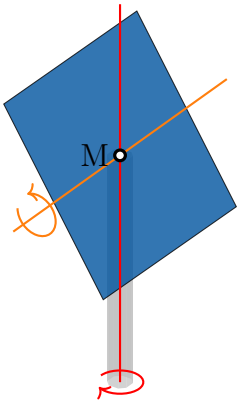
Sun Shapes The sun can be represented using different sun shapes. *SunFlower* currently supports four distributions [13]: Gaussian, Buie, Pillbox and custom. All sun shapes modify the sun vector which is static otherwise. An extensive overview of the different implementations is given by Hövelmann [13].
All aspects described until now will influence the result of the ray tracer. In the next Section, we will describe the last part of the ray tracer, the heliostat model.

3 Heliostat Model

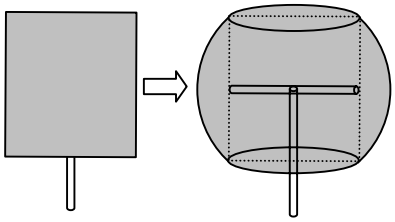
In the previous section, we introduced many aspects of the optical model for the simulation of CRS. In this section, the most important part of the optical model, heliostats, is discussed. First, we describe conventional, azimuth-elevation tracking heliostats and their rotational properties. We continue with the Tilt-and-Roll heliostat, describing its similarities and differences with azimuth-elevation tracking heliostats. After that, we explain aiming strategies for the different receiver types. This will be important when we introduce the tracking algorithm for both heliostat types.

3.1 Azimuth-Elevation Tracking Heliostats

Azimuth-elevation tracking heliostat models consist of several facets. Each facet is a reflective area with the form of a rectangle like in the Gemasolar Power Plant [6]. These facets can be mounted in different patterns. Each heliostat includes (servo) motors to control each axis and align the heliostat with the sun and the receiver according to the aiming strategy (see Section 3.4). [17]
In Gemasolar [6] each heliostat consists of a grid with 5 rows and 7 columns of rectangular facets with a gap of 0.04 m. Each facet has a width of 1.36 m and a height of 2.43 m. The heliostat is mounted on a horizontal bar which itself is mounted centered on a pedestal. In Figure 3a, we can see a heliostat using a single large facet for easier perception. Rotating along the orange axis enables the facet to be aligned with the elevation of the sun, while the red axis is used to track the azimuth.



(a) Rotational axes of an azimuth-elevation tracking heliostat. The red axis allows tracking of the azimuth. The orange axis enables alignment with the elevation of the sun.

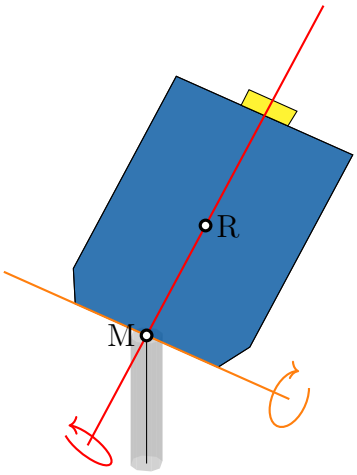


(b) Maximum expansion of an azimuth-elevation tracking heliostat in both rotational axes. Taken from [22].

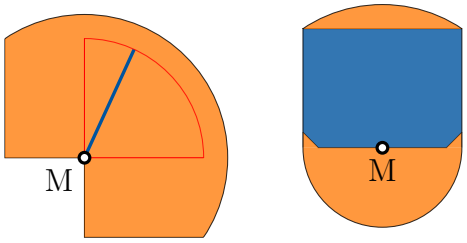
Figure 3: Model of an azimuth-elevation tracking heliostat with one facet (marked in dark blue) and a pedestal (marked in gray).

3.2 Tilt-and-Roll Heliostats

In contrast to conventional azimuth-elevation tracking heliostats, Tilt-and-Roll heliostats consist of only one reflective facet [22]. This facet is relatively small and uses an inexpensive mirror to reduce the initial cost. The rotational axes of these heliostats differ from those found in traditional azimuth-elevation tracking systems. In Figure 4a we can see the told axis colored in orange and the roll axis in red. The mounting point of the heliostat is at M of Figure 4a. The center of the heliostat facet which should be perfectly aligned with the receiver is denoted with R. We will discuss the tracking algorithm required for Tilt-and-Roll heliostats in Section 3.5. The bounding box is shown in Figure 4b. There we can see a side view and a front view.
In the side view, the orange area denotes the bounding box. The dark blue line shows an example position of the facet. The rotational range of the tilt axis is marked in red. The extension of the bounding box results from turning the facet along the roll axis. The front view shows the facet in the center as it is aligned with the horizon. If we tilt back the facet, it becomes ever smaller until it cannot be seen from the front. The bottom orange part of the bounding box is a result of the facet being tilted back completely and then rotating the facet along the roll axis.
In Figure 5b, the top-down view of a potential heliostat field with 16 heliostats is shown.
Depending on the location of the CRS, the availability of space can be a limiting factor. In most CRS, the location is chosen to provide enough space for the desired amount of heliostats. In most cases, this is far away from urban centers. [22]
With mCRS the aspect of space plays a much more important role as the location is not

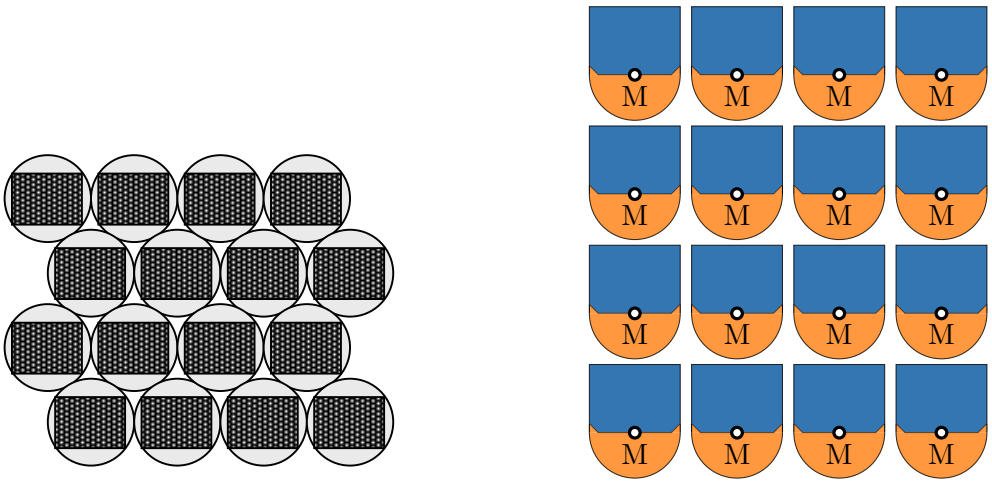


(a) Rotational axes of a Tilt-and-Roll heliostat. The red axis is used to roll the facet. The orange axis is used to tilt back the facet. The yellow box represents a miscellaneous object.



(b) Bounding box of a Tilt-and-Roll heliostat. On the left, we can see the side view with the bounding box marked in orange and an example position of the facet in dark blue. On the right, the front view is depicted. In the center, we can see the facet marked in dark blue. Above and below the facet, the bounding box is denoted in orange. In both views, the mounting point is labeled M.

Figure 4: Model of Tilt-and-Roll heliostat and the corresponding bounding box with one facet (marked in dark blue) and a pedestal (marked in gray).



(a) Ground coverage of azimuth-elevation tracking heliostats. This is a top-down view of closely packed heliostats. Each heliostat is aligned parallel to the ground with the bounding box drawn around it.

(b) Ground coverage of Tilt-and-Roll heliostats. This is a top-down view of closely packed heliostats. Each heliostat is aligned parallel to the ground. Point M marks the mounting point of the facet and pedestal. The facet is colored dark blue and the additional bounding box area is marked with orange.

Figure 5: Comparison of ground coverage of azimuth-elevation tracking and Tilt-and-Roll heliostats.

specified by the best location in terms of available space and sunlight, but rather by the deployment which it is meant to supply with power or heat (e.g. a mine). The ground coverage of a heliostat model describes its space efficiency concerning the maximum expansion in all rotational axes. We can determine the expansion by rotating the heliostat in both rotational axes, creating a bounding box. For an azimuth-elevation tracking heliostat, this can be seen in Figure 3b. The bounding box of azimuth-elevation tracking heliostats requires that all heliostats be placed at a significant distance from each other. This results in a low ground coverage of azimuth-elevation tracking heliostats. Tilt-and-Roll heliostats improve upon this problem by using different rotation axes, thus increasing the ground coverage. This makes Tilt-and-Roll heliostats well-suited for usage in mCRS.

3.3 Coordinate Systems

We use multiple coordinate systems with different alignments and origins to model all aspects of a heliostat. All coordinate systems use meters as the unit of length (one unit in the coordinate system is equal to one meter in the real world). First, we have the global coordinate system that is used to define the position of the receiver tower and each heliostat. Then, each heliostat has a coordinate system that is used to determine

the position of each facet in relation to the heliostat. Finally, each facet also has its own coordinate system to determine the position of each piece of the facet. Pieces are described in Section 4.4. To allow the projection of points from one coordinate system to another, we use two vectors that show the direction of the relative x axis $\vec{a}_x$ and relative y axis $\vec{a}_y$ respectively. The normal vector $\vec{n}$ can be calculated using the cross product of both vectors.

$$\vec{n} = \vec{a}_x \times \vec{a}_y \tag{4}$$

All vectors need to be normalized to properly project coordinates.

$$\vec{n}' = \vec{n} \,/\, \|n\| \tag{5}$$
$$\vec{a}'_x = \vec{a}_x \,/\, \|a_x\| \tag{6}$$
$$\vec{a}'_y = \vec{a}_y \,/\, \|a_y\| \tag{7}$$

x, y, z are the coordinates to be projected. The variable $\vec{g}$ will contain the projected point.

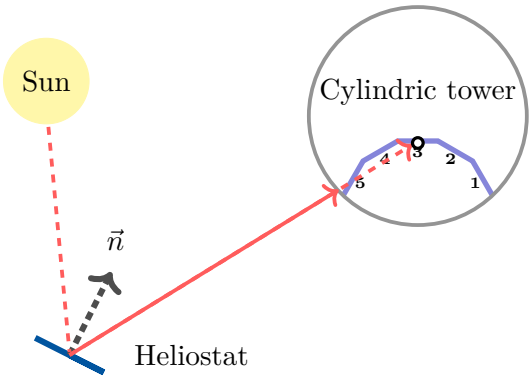
$$\vec{g} = \vec{a}'_x \cdot x + \vec{a}'_y \cdot y + \vec{n}' \cdot z \tag{8}$$

Using these coordinate systems, we can project the coordinates of a piece to global coordinates by first projecting it to the heliostat coordinate system using the axes of the facet and then further projecting it to the global coordinate system using the axes of the heliostat. These coordinate systems will be important for the implementation of measurement-driven simulation in Section 4.6.3.

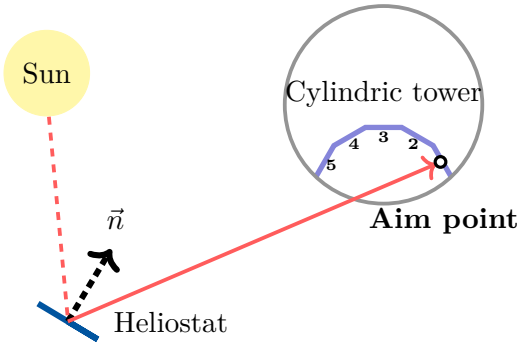
3.4 Aiming Strategy

The aiming strategy is part of the tracking algorithm and determines to which point the heliostat tries to reflect sun rays. The target differs depending on the type of receiver that is used in the power plant. It is beneficial for a receiver to distribute the received power evenly across the receiver surface to more easily transfer the heat to the fluid running through the tower. In Figure 2, we can see three different receiver types that are used by CRS. The flat tilted receiver in Figure 2a consists of a circular tower and a flat receiver piece mounted on the outside of the tower at a specified height. In Figure 2b, we can see a cavity receiver. It consists of a cylindric tower with a circular cavity. For this type of receiver, the aiming strategy is not as simple as using the center of the receiver because different heliostats that reflect sun rays onto the receiver can either hit the body of the tower or hit the receiver at a suboptimal angle. The cylindric external receiver consists of a tower and a receiver panel that spans around the body of the tower. This can be seen in Figure 2c. For this type, we also do not have a fixed center that can be used for heliostat tracking. For optimal absorption efficiency, the ray should hit the receiver perpendicularly.

Flat Tilted Receiver With this type of receiver, we align the center of the heliostat with the center of the flat receiver panel. The angle of impact will depend on the position of the heliostat.



(a) Suboptimal alignment of a heliostat with a cavity receiver. The rays, reflected by the heliostat, are blocked by the body of the tower. Adapted from [1].



(b) Optimal alignment of the heliostat with the receiver, by choosing a different piece of the receiver panel, as the aim point. Always the farthest piece from the heliostat is chosen. Adapted from [1].

Figure 6: Comparison of aim points for a heliostat using a cavity receiver.

Cylindric Cavity Receiver In this case, we need to consider the body of the tower as it can block incoming sun rays if the heliostat is not aligned properly, like trying to hit the center of the panel. In Figure 6, we can see two figures that each show a top-down view of a tower with a cavity receiver, a heliostat and a sun. The circular receiver panel is discretized into smaller pieces. These pieces are also used for generating the flux map that we see in Figure 18. In Figure 6a, the heliostat is aligned to track the center of the receiver panel. In this situation, the reflected ray will be blocked by the outer body of the receiver. In Figure 6b, the heliostat is set up to track the piece that is farthest from its position, resulting in an improved aim point. Rays reflected by the heliostat are much less likely to be blocked by the body of the tower. These figures only show a top-down view. The vertical axis can be considered analogically.

Cylindric External Receiver The cylindric external receiver has a circular tower body. The receiver panel is wrapped completely around the top of the tower as can be seen in Figure 2c. The receiver panels will be discretized into pieces that are rectangles. In Figure 7, we can see a circular tower body and an externally wrapped receiver panel that is discretized into 12 pieces. In addition, we have two heliostats that are each aligned to track piece 6 and 7 respectively.
For this receiver type, the alignment strategy also plays a vital role in correctly projecting sun rays onto the receiver panel. We cannot use the center of the top of the tower structure as it would lead to sun rays being reflected to the underside of the receiver panels. This is a result of heliostats usually being placed much lower than the receiver panel. We once again use the different pieces that were created during discretization. For each heliostat, we determine the closest receiver piece and use the center of said piece as a tracking point. This will result in a close to optimal angle and collection of sun rays.

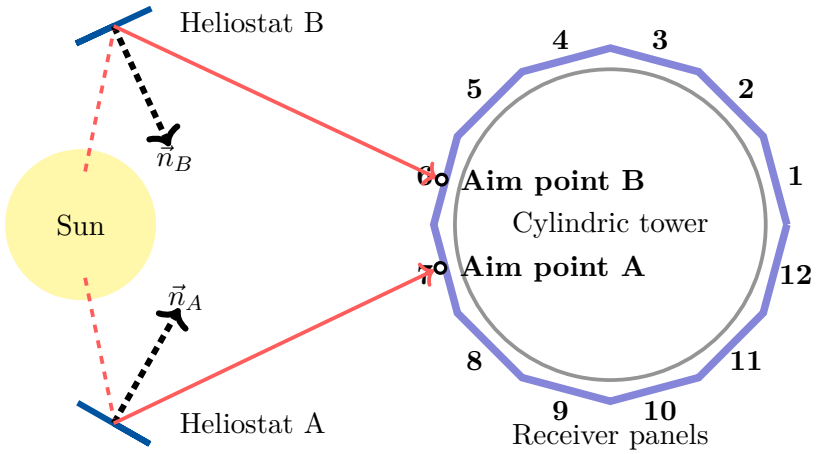


Figure 7: Aim point definition for heliostats when using a cylindric external receiver. The piece closest to the respective heliostat is chosen instead of the center. This results in an improved angle of the ray at the receiver panel facilitating the absorption of flux. Taken from [1].

One limitation of this implementation is the restricted flexibility for configuring the discretization. For all other receivers, we can increase the amount of pieces that are generated during discretization without any consideration for correctness. This leads to a flux map with a greater resolution which proves to be useful for the development of CRS. If we increase the number of vertical pieces, the tracking point for each heliostat will move further to the bottom edge of the receiver panel as we always choose the closest piece and heliostats are usually placed lower than the receiver panel. In the worst case, all heliostats will track the lower edge of the receiver panel and roughly half of the reflected sun rays will hit the tower body instead of being collected by the receiver panel.
Thus, for the cylindric external receiver, the configuration parameters need to be chosen carefully as they can influence the correctness of the simulation.
Next, we discuss tracking algorithms for heliostats. There, we use the aim point defined by the aiming strategy to align the heliostat with the sun and the receiver.

3.5 Tracking Algorithm

In this section, we introduce a new tracking algorithm required to work with the new rotational axes that are part of the Tilt-and-Roll heliostat model (discussed in Section 3). For the simulation and thus the generation of rays, we need to know the center of the heliostat and the aligned normal vector. The aim point of each heliostat is defined by the aiming strategy.

Azimuth-Elevation Tracking With azimuth-elevation tracking heliostats, the center of the mirror area is fixed and does not change with the rotation of any axis. To obtain the normal vector, we first calculate the vector $\vec{v}_{aim}$ from p_{center} to aim point

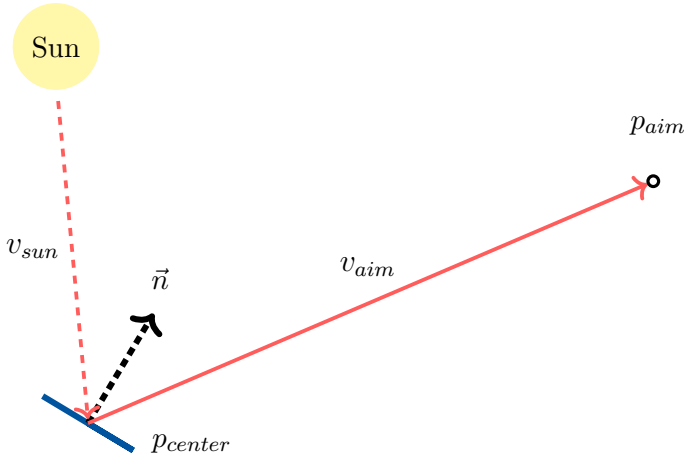


Figure 8: Visualization of the perfectly reflected sun ray based on a normal vector $\vec{n}$, the heliostat's center p_{center} and an aim point p_{aim}.

p_{aim} determined by the aiming strategy.

$$\vec{v}_{aim} = p_{aim} - p_{center} \tag{9}$$

We obtain the normal vector $\vec{n}$ by adding this vector to the reversed sun vector. All vectors $\vec{v}_{aim}$ and $\vec{v}_{sun}$ need to be normalized.

$$\vec{n} = \vec{v}_{aim} - \vec{v}_{sun} \tag{10}$$

In Figure 8 we can see a 2D representation of this algorithm. The labels for each vector and point correspond to the names of the variables in the equations. A pseudo-code implementation is given in Algorithm 1 with procedure ALIGN.

Tilt-and-Roll Tracking With Tilt-and-Roll heliostats, the center of the heliostat depends on the tilt angle. Thus, the computation is not as trivial as with azimuth-elevation tracking heliostats. We introduce an iterative procedure in Algorithm 1. The mounting point M is used as an initial center. Next, we iteratively calculate the center of the heliostat based on the current normal vector and then recalculate the new normal vector for this new center. The calculation of the center based on the normal vector is done in GET_CENTER_MIRROR. We define two possible epsilons (break-conditions of the while-loop) that are based on the distance between the previous and current center point ϵ_p or the angle between the current and previous normal vector ϵ_a respectively. If this epsilon reaches a specified limit, we break out of the while loop and use the last calculated normal and center as return values. The complete algorithm using the distance between the last two centers can be found in Algorithm 1.

Center based on Normal Vector In the method GET_CENTER_MIRROR we calculate the center of the mirror based on a targeted normal vector $\vec{n}_{\text{hel}}$. We need to

Algorithm 1 Iterative tracking algorithm for heliostats.

```
1: procedure ALIGN(center, sun, aim_point)
2:     dir_receiver = (aim_point - center).toNormalized();
3:     normal = (dir_receiver - sun).toNormalized();
4:     return normal;
5: end procedure
6: procedure GET_CENTER_MIRROR(hel, normal)
7:     sky = Vector(0,0,1);
8:     plane_normal = cross_product(hel.tilt_axis, sky).toNormalized();
9:     in_plane = normal - (dot_product(plane_normal, normal)) * plane_normal;
10:    rot = rotational matrix that turns π/2 along plane_normal;
11:    res = (rot * in_plane).toNormalized();
12:    return hel.mounting_point + ((hel.mirror_height / 2) * res);
13: end procedure
14: procedure ALIGN_TNR(sun, hel, rec)
15:    center = hel.mounting_point;
16:    previous_center = center + Vector(0,0,EPSILON);
17:    normal = Vector(0,0,1);
18:    while (previous_center - center).norm() >= EPSILON do
19:        previous_center = center;
20:        normal = ALIGN(center, sun, rec);
21:        center = GET_CENTER_MIRROR(hel, normal);
22:    end while
23:    return center, normal;
24: end procedure
```

tilt back the heliostat as such that the normal vector is perpendicular to the mirror. For this, we first define the vector $\vec{v}_{\text{tilt}}$ which points in the direction where the heliostat tilts back. The plane $plane_{\text{hel}}$ is based on the vector of the mounting point M and is spanned by $\vec{v}_{\text{tilt}}$ and the vector that points towards the sky $\vec{v}_{\text{sky}} = (0, 0, 1)^T$. The scalar value M_h denotes the total height of the mirror facet.

$$\vec{n}_{\text{plane}} = \vec{v}_{\text{tilt}} \times \vec{v}_{\text{sky}} \tag{11}$$
$$\vec{n}'_{\text{plane}} = \vec{n}_{\text{plane}} \,/\, \|\vec{n}_{\text{plane}}\| \tag{12}$$

After this setup, we can project $\vec{n}_{\text{hel}}$ onto $plane_{\text{hel}}$ and obtain $\vec{n}_{\text{inplane}}$.

$$\vec{n}_{\text{inplane}} = (x, y, z)^T = \vec{n}_{\text{hel}} - (\vec{n}'_{\text{plane}} \cdot n_{\text{hel}}) \tag{13}$$

Using the rotational matrix r with angle $\alpha = \frac{\pi}{2}\,\text{rad}$,

$$r = \begin{pmatrix} x^2(1-\cos\alpha) + \cos\alpha & xy(1-\cos\alpha) - z\sin\alpha & xz(1-\cos\alpha) + y\sin\alpha \\ yx(1-\cos\alpha) + z\sin\alpha & y^2(1-\cos\alpha) + \cos\alpha & yz(1-\cos\alpha) - x\sin\alpha \\ zx(1-\cos\alpha) - y\sin\alpha & zy(1\cos\alpha) + x\sin\alpha & z^2(1-\cos\alpha) + \cos\alpha \end{pmatrix} \tag{14}$$

we then rotate $\vec{n}_{\text{inplane}}$ by $90° \equiv \frac{\pi}{2}\,\text{rad}$ and normalize it.

$$\vec{\text{res}} = r \cdot \vec{n}_{\text{inplane}} \tag{15}$$
$$\vec{\text{res}}' = \vec{\text{res}} \,/\, \|\vec{\text{res}}\| \tag{16}$$

This vector points along the mirror facet. To calculate the center of the mirror, we stretch this vector by half of the height of the mirror and add it to M.

$$p_{\text{center}} = M + (M_h \,/\, 2) \cdot \vec{\text{res}}' \tag{17}$$

Finally, we obtain the center p_{center} corresponding to the given normal vector.

Calculation of Epsilons To calculate the distance between the current $p_{\text{center}} = (x, y, z)^T$ and previous center $p'_{\text{center}} = (x', y', z')^T$, we can use the euclidean distance:

$$\epsilon_p = \sqrt{(x-x')^2 + (y-y')^2 + (z-z')^2}. \tag{18}$$

The angle between the current $\vec{n}_{\text{hel}}$ and previous $\vec{n}'_{\text{hel}}$ normal vector is given by

$$\epsilon_a = \text{acos}\left(\frac{\vec{n}_{\text{hel}} \cdot \vec{n}'_{\text{hel}}}{\sqrt{\text{norm}(\vec{n}_{\text{hel}}) \cdot \text{norm}(\vec{n}'_{\text{hel}})}}\right). \tag{19}$$

In this section, we described two heliostat models and their properties. We explained fundamental concepts for tracking the sun and reflecting sun rays onto a receiver. In the next section, we discuss our ray tracers and illustrate the concepts used to make efficient and accurate ray tracing possible.

4 Ray Tracer

SunFlower offers multiple ray tracers with different accuracy and performance characteristics. These include analytic, Monte Carlo, convolution and integrated convolution ray tracers. A complete overview of these is given by Hövelmann [13]. All ray tracers are available as an implementation on the CPU and GPU. In Section 4.3, the CUDA implementation of the ray tracers is discussed. We will focus on the analytic and Monte Carlo ray tracer. The overall purpose of a ray tracer is to determine the intercept power of each heliostat. All ray tracers consist of multiple parts: object discretization, ray generation, shading, blocking and statistic creation. For discretization, all ray tracers use pieces as the foundational shape. In the existing implementations, pieces are either rectangles or triangles. Some parts of the ray tracers, especially in the CUDA implementation, rely only on rectangles as the underlying shape. With the introduction of the irregular polygon as a facet shape, we need to have ray tracers that rely solely on triangles. The shading aspect of a ray tracer checks if a ray should be generated at the specific point in the first place by tracing towards the sun and checking for collisions. Blocking is done similarly, but by tracing the ray towards the receiver and checking for blocking by other heliostats. This is called bi-directional ray tracing. To reduce the number of shading and blocking checks we first generate a list of candidates for each heliostat and later for each facet. Each ray tracer also generates statistics for each heliostat. At the end of a simulation, each heliostat's performance is analyzed. In addition to the new shape, we also introduce a new positioning system for facets and the ability to create blind facets that do not generate rays as they are not reflective.
As each heliostat model is an abstract and not a perfect representation of the actual heliostat, we also created the option to use a scan of a heliostat facet for ray generation instead of the model.
First, we explain the intercept power of a heliostat's piece and continue with the analytic and Monte Carlo ray tracers, then we discuss CUDA acceleration of said ray tracers. After that, discretization and the concept of AABB trees are introduced. Furthermore, we illustrate our implementation of measurement-driven simulation.

Intercept Power Each piece of a heliostat obtained by discretization (see Section 4.4) has a certain intercept power. This describes the energy (or flux) $P_{\text{int,piece}}$ that is reflected onto the receiver by that piece. It depends on multiple factors: the Direct Normal Irradiation I_{DNI}, representative piece area A_{piece}, cosine efficiency $\eta_{\cos}$, reflectivity η_{ref}, atmospheric attenuation η_{aa}, intercept efficiency η_{int} and whether the ray was shaded or blocked. η_{sb} is either one or zero depending on shading and blocking of the ray that was generated by the respective piece. The calculation is as follows:

$$P_{\text{int,piece}} = \underbrace{I_{\text{DNI}}\, A_{\text{piece}}\, \eta_{\cos}\, \eta_{\text{ref}}\, \eta_{\text{aa}}}_{=:P_{\text{reflected}}}\, \eta_{\text{sb}}\, \eta_{\text{int}}. \tag{20}$$

4.1 Analytic Ray Tracer

The analytic ray tracer is the simplest ray tracer delivering a balance of performance and accuracy. It generates rays on top of each facet and traces them in two directions (bi-directional ray tracer). One ray is traced towards the sun to check for shading and the other ray is traced towards the receiver to check for blocking. Each ray is traced ideally, without any artificially added fluctuations that try to mimic real world deviations (e.g. diffuse reflection). Thus, this ray tracer produces deterministic results.

Convolution Method The previously mentioned ray tracer uses representative rays that are traced to determine the intercept efficiency of the receiver.
The convolution method uses a different approach. It projects the receiver onto the image plane using a perspective projection [13]. Every projection only requires one matrix multiplication and computes the local coordinates of the image plane. There is also an integrated convolution method available. Both are described in detail by Hövelmann [13].

Shading & Blocking Calculations The configuration of shading & blocking is essential to achieve accurate and fast results. Using one ray for each heliostat facet would produce invalid results in most cases as it is a very simple approximation for a large area. To improve accuracy, facets can be discretized into a configurable amount of pieces. Each piece then generates a ray and has a smaller representative area. This dial can be used to set the trade-off between lower run time and higher accuracy.
To reduce the overall run time of the simulation, we pre-calculate potential shading and blocking candidates for each sun position. This significantly reduces the amount of shading and blocking checks required for each ray.
Next, we explain how we mimic real world deviations in the simulation by using a Monte Carlo-based ray tracer.

4.2 Monte Carlo Ray Tracer

The analytic ray tracer does not consider real-world deviations when it comes to reflecting rays off a heliostat's surface. The Monte Carlo ray tracer extends the analytic implementation by introducing small fluctuations to each ray. An example using a Gaussian normal distribution can be seen in Figure 9. There, we perturb a ray (marked in red). The heliostat is colored in dark blue and the receiver in light blue. Monte Carlo simulations require the law of large numbers to counter the introduced fluctuations and still obtain an accurate result. Thus, each ray is perturbed multiple times and each perturbed ray is independently checked. This increases the processing time of the ray tracer compared to other solutions. Another bottleneck for performance is the generation of pseudorandom numbers for ray perturbation. As we generate millions of rays per simulation, we might need to generate billions of pseudorandom numbers depending on the configuration.

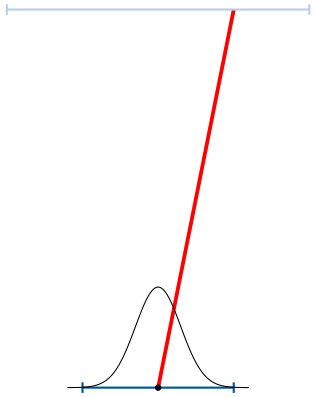


Figure 9: Perturbation of a ray using Monte Carlo simulations. The heliostat is shown in dark blue, the Gaussian distribution in black and the receiver in light blue. The red line represents a possible perturbed ray. Adapted from [12].

4.3 CUDA Acceleration

Aside from a few setup tasks, the computation of shading and blocking for each ray (and in extension heliostat) is independent from all other rays (and heliostats).
A CPU consists of a few high-performing cores with large caches. Each core works independently. GPUs in contrast contain many slower cores with smaller caches for each core. This makes GPUs beneficial for highly parallel computations [24].
CUDA-based GPUs use threads. A group of 32 threads is called a warp. Each warp has shared caches for all threads and a small amount of registers for each thread[3]. Threads in one warp cannot branch as they share one common program counter. This makes CUDA programming quite different compared to programming on a CPU [1].
The initial implementations of ray tracers in CUDA was done by Aldenhoff [1] and improved on by Hövelmann [13].

4.4 Discretization for Accelerated Shading & Blocking

The existing ray tracer implementations are not complete and some require rectangles as the foundational piece for part of the ray tracing process. We will change these implementations to only use triangles for heliostats in all aspects of the ray tracer. Triangles are suited well as a foundation shape, as all polygons including triangles and rectangles, can be cut into a list of triangles. This process is called *discretization*.
Triangles are also well suited for collision detection as a strict intersection test is possible in constant time [1]. These triangles will be discretized further to achieve a specific ratio. These smaller pieces can then be used for ray generation. Given the existing shapes of facets (rectangular, regular polygon, triangle), discretization into triangles is not complex. For rectangular facets, we can cut the rectangle in half and retrieve two triangles (see Figure 10a). We do not need to further process triangular

[3] `https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/`

(a) Discretization of a rectangle facet into two triangles.

(b) Discretization of a regular polygonal facet with 8 corners into 8 triangles.

Figure 10: Discretization of simple shapes.

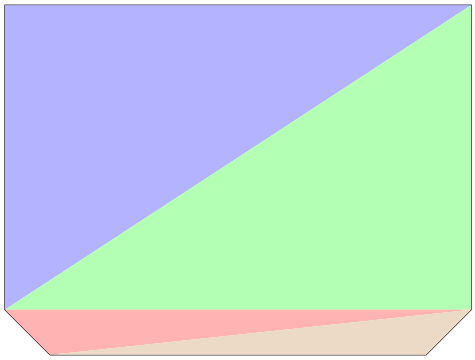
facets. For regular polygons, we split the polygon by drawing a line from every corner to the center. This creates n triangles for a regular polygon with n corners. This can be seen in Figure 10b.

With **irregular polygons** this process is more involved as the algorithm needs to consider every possible layout of the points that make up the polygon. There are many different approaches that we will not discuss further. Due to its simplicity and support for polygonal holes, we chose the ear-cut (also called ear-clipping) method [8]. This method is based on the concept of *ears*. An ear is a triangle formed by three points of the polygon if two edges between the three points exist as part of the polygon and the third potential edge is contained within the overall polygon [8]. Considering Figure 11b, we can identify two ears: $\{1, 2, 4\}, \{1, 3, 4\}$. If we choose the points $\{2, 3, 4\}$ we can still span a triangle but the edge $(2, 3)$ (marked with a dotted line) would not be contained inside the polygon and thus this triangle does not qualify as an ear.

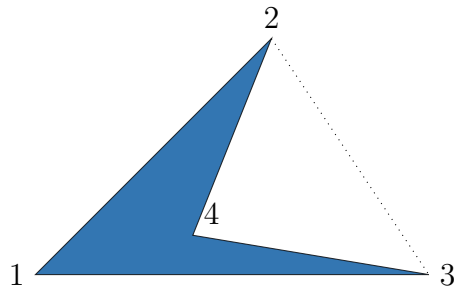
To discretize an irregular polygon, we first determine all ears of the given polygon and iteratively *cut* them from the polygon by removing a corner. There exist different heuristics for choosing which ear to cut. For a more detailed overview of the ear-clipping method, refer to Eder et al. [8]. An example discretization of a Tilt-and-Roll heliostat facet is show in Figure 11a. Now, we have discretized each facet into a list of triangles. These triangles can then be used to construct an AABB Tree. This is described in the next section.

For ray generation, we need to further discretize each triangle into multiple smaller triangles to fulfill a configurable precision level. These triangles are then used to generate a ray in the center of each piece. Each triangle needs to be discretized evenly, otherwise the result will no longer be representative. We use a naïve algorithm that divides the triangle in half, creating two new triangles in the process. This is repeated until the configured density is reached.

A disadvantage of this algorithm is the fact that it only allows to discretize triangles into 2^i ($i \in \mathbb{N}_0$) smaller triangles and does not allow an exact number of sub triangles. The advantage is a perfect distribution of triangles across the area. In Figure 12, we can see a triangle that is discretized into $2^3 = 8$ smaller triangles with each center marked by a dot.



(a) Possible discretization of a facet of a Tilt-and-Roll heliostat using an irregular polygon as a shape. We receive four triangles (each marked with a different color) that, when combined, make up the whole facet.



(b) Irregular polygon with four corners. The triangles spanned by $\{1, 2, 4\}, \{1, 3, 4\}$ are *ear*s. The triangle spanned by $\{2, 3, 4\}$ is not an *ear*.

Figure 11: Discretization of an irregular polygon using the ear-cut method.

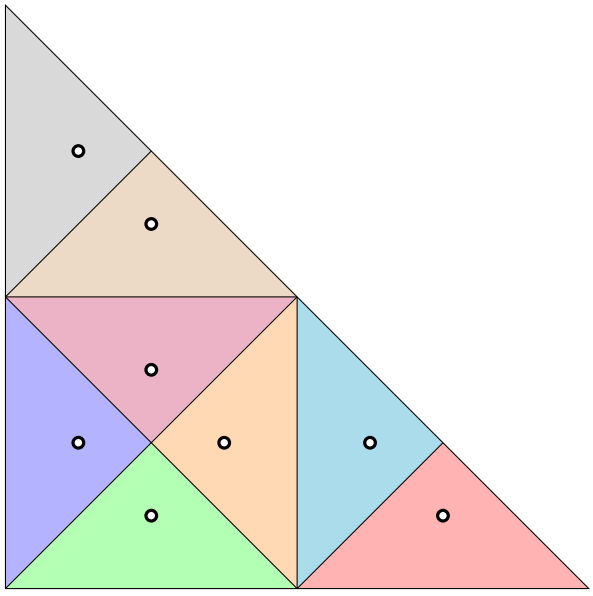


Figure 12: Even discretization of a triangle into 8 smaller triangles. The center of every triangle is marked.

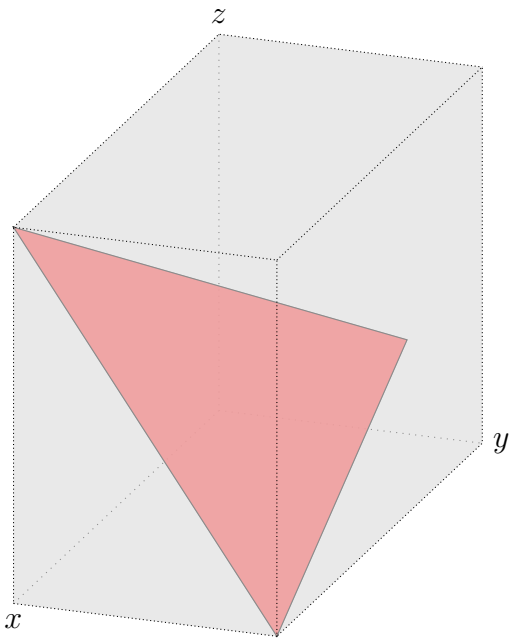
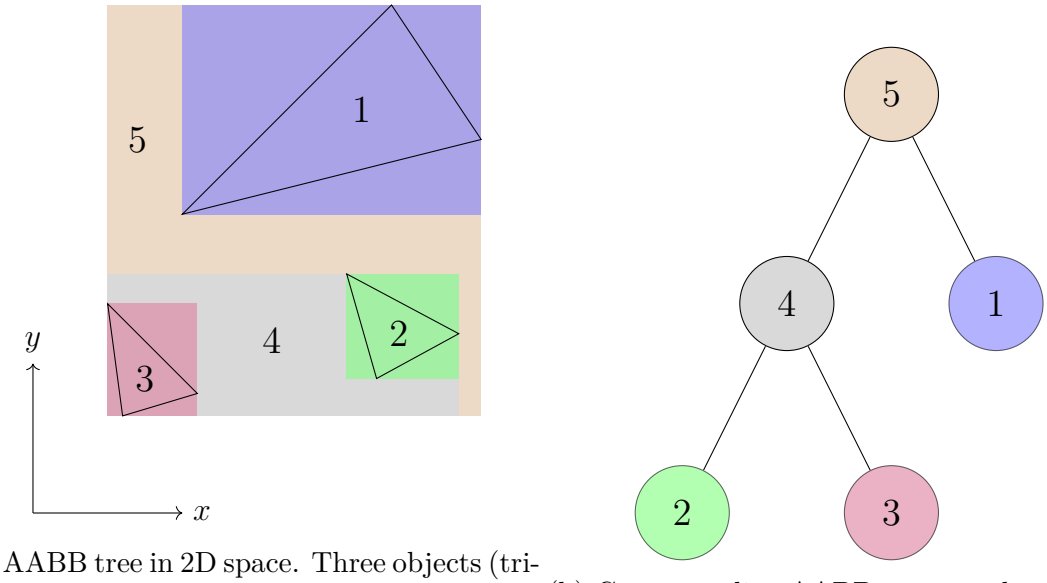


Figure 13: Axis-aligned bounding box of a triangle in 3D space. The triangle is colored in red. The bounding box is denoted in gray.

4.5 AABB Tree

Now that we have seen how a facet's shape is discretized into triangles, we explain how we use these triangles for efficient collision detection.
An **axis-aligned bounding box (AABB)** describes an over-approximation of a complex object (triangles in our case) by drawing a box around the maximum expansion of the object in each direction (positive x, negative x, ...) as can be seen in Figure 13. We can use this AABB as a stand-in for the triangle to check if the box is intersected by a ray. This check can be done in constant time [1] whereas directly checking the triangle would be more computationally expensive. In combination with a tree structure, multiple triangles can be grouped into an AABB tree.

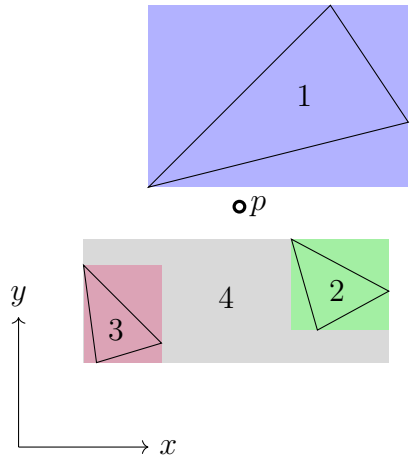
AABB Tree An **AABB tree** consists of nodes and leafs. Each node has an AABB property and up to two child leaves. A leaf represents a single triangle. For easier comprehension, Figure 14a only shows two dimensions. The concept is analogous for three or more dimensions.
In Figure 14a, we can see three triangles that represent areas we want to test for collision with a ray. The AABB of each node contains all AABBs of all of its (recursive) child nodes. The corresponding AABB tree is shown in Figure 14b. Nodes 1, 2 and 3 are leafs. Node 5 is the root. Testing each triangle individually does not scale well with an increasing number of triangles. This tree structure of AABBs allows us to generate a list of collision candidates which can then be used for strict collision testing.



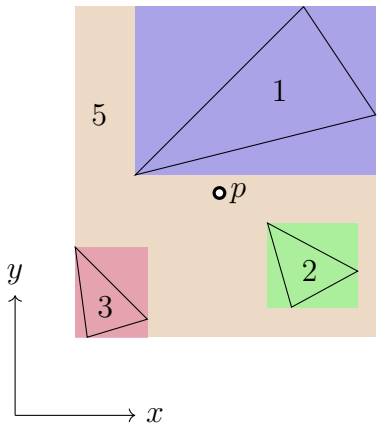
(a) AABB tree in 2D space. Three objects (triangles) of interest are contained in the area (labeled with 1, 2 and 3). The gray area (labeled with a 4) is a result of the combination of triangle 2 and triangle 3. The brown area (labeled with a 5) stems from combining triangle 1 and area 4.

(b) Corresponding AABB tree as a data structure. Node 5 is the root node. Nodes 4 and 1 are its child nodes. Node 4 consists of nodes 2 and 3. The leafs 1, 2 and 3 each contain a triangle.

Figure 14: Example AABB Tree using three triangles as objects. The tree consists of 5 nodes.



(a) Combining nodes 2 and 3 to node 4. Point p does not collide with any tree node and the collision detection can abort.



(b) Combining nodes 1 and 3 to node 5. It seems like point p might collide with one triangle, even though it does not. Thus the tree is traversed further, introducing computational overhead.

Figure 15: Comparison of AABB tree construction. The construction and thus layout of the tree is important for efficient tree traversal.

Tree Construction The layout of the tree is essential for efficient collision testing as it has to be done for each generated ray with all shading & blocking candidates. In the worst case, each node has at most one child which results in linear traversal run time in comparison to logarithmic run time in a flat tree where nodes are evenly distributed. In general, we can differentiate between two approaches for tree construction, bottom-up and top-down. In the bottom-up approach, we generate a leaf for each triangle and iteratively combine two nodes into an upper node until exactly one node is left. This node will become the root of the tree. The top-down approach constructs the tree by iteratively adding leaf nodes into the tree and traversing the tree to find the best location for the newly added node. The advantage of the bottom-up approach is the performance of construction, but this method requires a given strict structure of leaf nodes such that nodes that are close to each other can be combined. Such a fixed structure is not given for all configurable heliostat shapes. In Figure 15, we can see two possible combinations. In Figure 15a, we combine node 2 and node 3 to node 4. In Figure 15b, we combine node 1 and node 3 to node 5. By comparing the two possibilities, we can see that in Figure 15a the area that is *not* actually covered by any triangles (marked in gray) is much smaller than the area in Figure 15b (marked in brown).
If we now test a point like p, the AABB of the combined node 4 will not be a hit and thus, this path of a potential tree will not be traversed further. In comparison, p will be counted as a hit of the AABB of node 5. As a result, this path will be traversed further. This problem extrapolates itself the more nodes are added to the tree. Hence, the construction of the tree must combine nodes that are placed close to each other.

The top-down method inserts each node iteratively and tries to find the best location for each node in the tree. This is done with a heuristic considering the area of an AABB that is not covered by any of the possible child nodes. An implementation inspired by an existing realization[4] is given in Algorithm 2.

Collision Detection and Tree Traversal The most computationally intensive part of CRS simulation is the tracing of sun rays. Thus, an efficient method for collision detection is essential for fast simulation. In the previous paragraph, we discussed the construction of an AABB Tree for efficient collision detection. In the following, we will show an iterative algorithm that is used to traverse the AABB tree and generate a list of collision candidates. The algorithm can be seen in Algorithm 3 and requires two input parameters: the tree we want to traverse and the ray that should be checked for collision. The method will return a list of collision candidates (triangles in this case) that need to be checked for strict intersection. We initialize a stack with the root node of the tree as the only element. Then we loop over the stack's elements until the stack is empty. For each iteration, we check if the AABB of the current node at the top of the stack intersects with the ray. If it does, we need to consider it further for traversal. We then check if that node is a leaf. In this case, we have a leaf node whose AABB intersects with the ray. This implies that we can add its triangle to the list of collision candidates to check for strict collision later. If the node is not a leaf, this indicates that at least the left child is populated and thus needs to be considered. We add the left child node to the stack. The right child node does not have to be populated. As a result, we first have to check if it is populated and then add it to the stack if necessary.

4.6 Measurement-Driven Simulation

In Section 4.2, we presented the Monte Carlo ray tracer, an extension of the analytic ray tracer that incorporates realistic imperfections. To optimize the focus of heliostats, two widely employed techniques have been developed: facet canting and mirror curvature. Facet canting modifies the positioning of facets on the heliostat to improve the reflection direction. Mirror curvature alters the shape of the mirror to a curved configuration, thereby refining the focal point. For both methods, mathematical models are used. This results in a perfect representation of curvatures. In real world heliostats, these curvatures cannot be produced with such high accuracy. Especially when using inexpensive mirrors like in Tilt-and-Roll heliostats, this discrepancy between the digital model and the real world heliostat can lead to inaccurate simulations. In this section, we first describe facet canting and mirror curvature. Next, we explain the concept of measurement-driven simulation based on a scan of the mirror's surface.

[4] https://github.com/JamesRandall/SimpleVoxelEngine

Algorithm 2 Inserting a new leaf into an AABB tree by traversing it to find the best location for the new leaf.

```
1: procedure INSERTLEAF(leaf)
2:     current = rootNode;
3:     while !current.isLeaf() do
4:         left = node.left;
5:         right = node.right;
6:         combinedAABB = current.aabb.merge(leaf.aabb);
7:         newParentCost = 2 * combinedAABB.area;
8:         minPushDownCost = 2 * (combinedAABB.area - current.aabb.area);
9:         costLeft = 0, costRight = 0;
10:        if left.isLeaf() then
11:            costLeft = leaf.aabb.merge(left.aabb).area + minPushDownCost;
12:        else
13:            newAABB = leaf.aabb.merge(left.aabb);
14:            costLeft = newAABB - left.aabb.area + minPushDownCost;
15:        end if
16:        if right.isLeaf() then
17:            costRight = leaf.aabb.merge(right.aabb).area + minPushDownCost;
18:        else
19:            newAABB = leaf.aabb.merge(right.aabb);
20:            costRight = newAABB - right.aabb.area + minPushDownCost;
21:        end if
22:        if newParentCost < costLeft && newParentCost < costRight then
23:            break;
24:        end if
25:        if costLeft < costRight then
26:            treeNodeIndex = leftNodeIndex;
27:        else
28:            treeNodeIndex = rightNodeIndex;
29:        end if
30:    end while
31:    treeNodeIndex = leafNode.parentNodeIndex;
32:    fixUpwardsTree();
33: end procedure
```

Algorithm 3 Iterative tree traversal that generates a list of collision candidates.

```
1: procedure COLLISION(tree,ray)
2:     stack = new Stack();
3:     result = new Vector();
4:     stack.push(tree.rootNode);
5:     while !stack.empty() do
6:         node = stack.pop();
7:         if node.aabb.intersectsWith(ray) then
8:             if node.isLeaf() then
9:                 result.append(node.object);
10:            else
11:                stack.push(node.left);
12:                if node.right != null then
13:                    stack.push(node.right);
14:                end if
15:            end if
16:        end if
17:    end while
18:    return result;
19: end procedure
```

4.6.1 Facet Canting

Canting is the process of adjusting the position of mirror facets to optimize the focal point of the reflective area [17]. We usually differentiate between on-axis and off-axis canting.
For on-axis canting, we assume that the sun vector and target vector are perpendicular to the facet. From this point on, we adjust the position of each facet to optimize in such a way that all facets individually hit the target on the receiver panel [5].
Off-axis canting also incorporates the position of the sun in the calculation of the facet positions and we no longer assume that both vectors are perpendicular to the facet. As a result, the position of each facet needs to be recalculated for each sun position. The calculation itself is the same as for on-axis canting.
For Tilt-and-Roll heliostats with one facet, canting is not applicable. Large heliostats with many facets can use canting to obtain improved energy output of the CRS [5].

4.6.2 Mirror Curvature

With increasing mirror size and distance between the heliostat and receiver, the misalignment of the outer parts of a mirror results in optical losses. The rays reflected by the outer part of a mirror miss the receiver, even though they are not shaded or blocked by other heliostats. To reduce this unwanted effect, mirrors (or facets) can be curved to have a focal point that correlates with the distance from the heliostat to the receiver. [17]

X	Y	Z	Xn	Yn	Zn	A
⋮	⋮	⋮	⋮	⋮	⋮	⋮
-0.812856	-0.606723	0.0057868	0.0101644	0.0058244	0.999931	10.7708
-0.812586	-0.269545	0.0046388	0.0096117	0.0022993	0.999951	4.37469
-0.812385	-0.276235	0.0046487	0.0096087	0.0024443	0.999950	15.3070
-0.812073	-0.286640	0.0046649	0.0096264	0.0025026	0.999950	15.3000
-0.811762	-0.297042	0.0046821	0.0096509	0.0025707	0.999950	15.2929
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 1: Heliostat scan using a point cloud with normal vectors and representative area for the respective point. The columns X, Y and Z contain the coordinates of the respective point from where the ray should be generated. Xn, Yn and Zn are the components of the normal vectors. Column A contains the representative area in mm^2.

Approximation for Simulation Mirror curvature can be configured in *SunFlower* by defining the focal point(s) of mirrors in meters. The algorithm then chooses the focal point that is closest to the actual distance between the heliostat and the receiver.
The process of ray generation (see Section 4) is modified to move each ray origin (the center of the piece for which the ray is generated) to the corresponding position in the circle that is drawn by the curvature.
We also need to adjust the normal vector of the piece as the reflective behavior of the piece is changed by the curvature. In the end, we have a new origin and normal (thus, also the reflection vector) of the ray and can begin to trace it as we would normally.
When configured correctly, this approximation uses a perfect curvature which is only applicable to expensive, high-quality mirrors. The mirrors used by Tilt-and-Roll heliostats are more affordable and thus not built to the same quality standards. Using curvature for these mirrors would result in inaccurate results. We address this problem in the following section.

4.6.3 Measurement-Driven Simulation

As curvature cannot be applied well to mirrors of lower quality and we want to represent the actual mirror surface as best as possible to enable accurate simulations, we introduce *measurement-driven simulation*.
We parse a scan of a mirror and use this scan to generate rays instead of generating rays based on the shape of the facets. In Table 1 an excerpt of such a scan for the facet of a Tilt-and-Roll heliostat is shown. The first three columns describe the coordinates of the point in the heliostat coordinate system. The center of this coordinate system is the mounting point of the heliostat. Columns Xn, Yn and Zn contain the three components of the normal vector. The representative area of a ray is given in column A. The unit of column A is square millimeters.
The existing implementation generates rays by discretizing each heliostat facet into

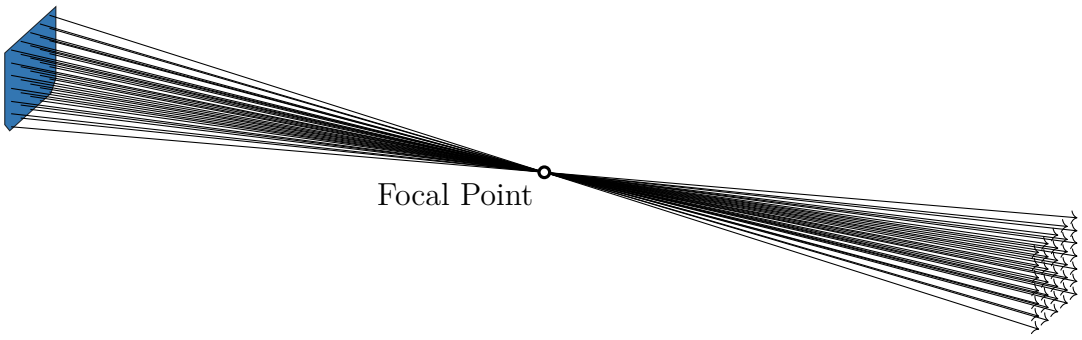


Figure 16: Use case of a heliostat scan. The facet modeled using an irregular polygon is shown in dark blue on the left. The focal point is given in the center of the figure. We can now emulate the actual surface of the facet with the data generated from a scan.

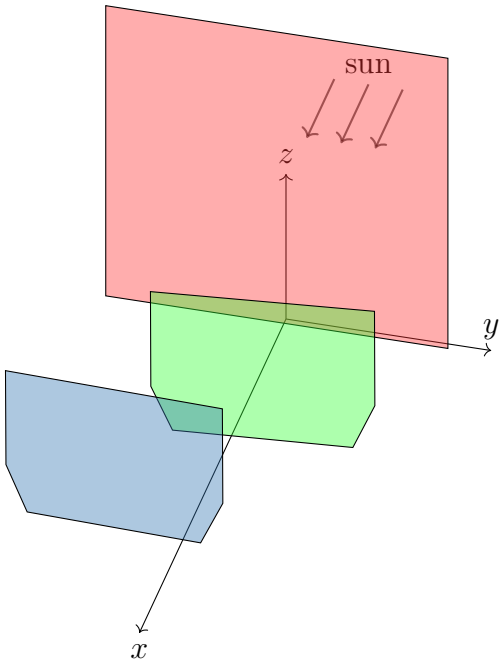
many smaller triangles which each generate a ray in the center of a piece (see Section 4.4). To this end, we first project the coordinates of the piece (which is in the facet coordinate system) to the heliostat coordinate system using the method described in Section 3.3.
To also properly depict facet canting and not only mirror curvature, the scan data is interpreted as rays for the entire heliostat and not just for one facet. To project each ray's origin to the global coordinate system, we need to project it from the heliostat system.
Checks for shading & blocking remain based on the original heliostat shape as we cannot make a direct connection from rays back to the heliostat shape. Thus, to use measurement-driven simulation, a list of rays and a heliostat shape is required. A validation of ray positions and directions is not possible as it is incompatible with the idea of accurately modeling real-world deviations. This check would also be computationally expensive.
Until this point, we have introduced the optical model, two heliostat models and their properties and described the challenges and implementations of ray tracing. In the next section, we validate the correctness and run time of our implementation.

5 Validation

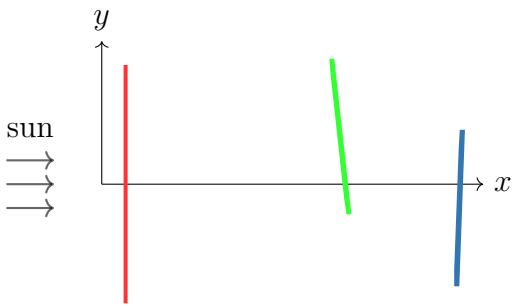
To validate our implementations, we consider the run time, system resources and correctness. First, we use a small simulation consisting of two heliostats to verify the model of a Tilt-and-Roll heliostat. Next, we inspect the amount of loop iterations needed for accurate alignment using the new iterative tracking algorithm based on two different break-conditions. Furthermore, we analyze the run time and RAM usage of the analytic and Monte Carlo ray tracer. The properties of the power plants used in Section 5.3, Section 5.4 and Section 5.5 are described in Table 2. Finally, we check the result of these simulations for correctness.

Parameter	PS20	PS10	Gemasolar	abengoaCRS
Number of heliostats	1255	624	2650	8600
Heliostat width	12.96 m	12.84 m	11 m	10.71 m
Heliostat height	9.69 m	9.45 m	10 m	12.95 m
Heliostat reflectance	88 %	88 %	93 %	91.96 %
Receiver shape	Cylindric cavity	Cylindric cavity	External cylindrical	External cylindrical
Receiver panels	4	4	18	16
Receiver panel width	3.445 m	3.445 m	1.476 m	3.158 m
Receiver height	12 m	12 m	16 m	18.5 m
Receiver top height	165 m	115 m	126.5 m	229.5 m
Sun error (Gaussian)	2.35 mrad	2.35 mrad	2.35 mrad	2.35 mrad
Slope error	1 mrad	2.6 mrad	2.6 mrad	2.6 mrad
Tracking error	1 mrad	1.3 mrad	1.3 mrad	1.3 mrad

Table 2: Properties of power plants used for validation. Adapted from [1].



(a) 3D representation of the validation test using one receiver and two heliostats.



(b) Top-down view of the setup. The shading effect of the green heliostat is visible.

Figure 17: Setup of the simulation used for validation test of ray tracer using one receiver (red) and two heliostats (green and blue). The green heliostat shades about half of the blue heliostat's reflective area. The sun originates from behind the receiver. Tower shading is disabled.

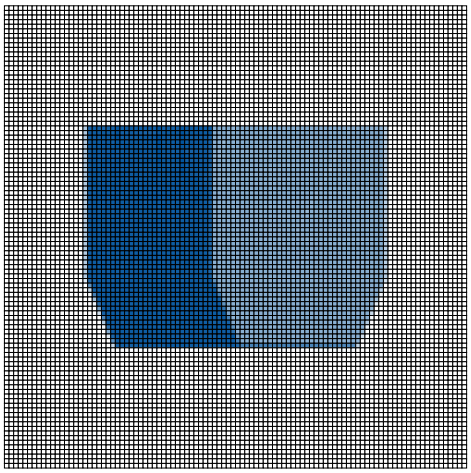


Figure 18: Receiver flux map with a resolution of 100 × 100 pieces. The colors indicate the amount of flux per piece that was reflected by the heliostats. The lighter the color, the less flux was captured at the respective piece.

Parameter	Value
Ray tracer	Analytic
Accuracy	10000 rays / m^2
Rec. dimensions	10 m × 10 m
Rec. discretization	100 × 100
Heliostat polygon	(2.580324 m, −2.4 m), (3.2 m, −0.89016 m), (3.2 m, 2.4 m), (−3.2 m, 2.4 m), (−3.2 m, −0.89016 m), (−2.580324 m, −2.4 m)
Positions	(10.0 m, 2.0 m, 0 m) (15.0 m, −2.0 m, 0 m)

Table 3: Configuration parameters for the blocking and shading validation. Receiver is abbreviated to Rec.

5.1 Blocking and Shading Validation using a Flux Map

To validate the correctness of the discretization of irregular polygons of the ray tracer, we want to verify that each heliostat correctly reflects sun rays and that the shape of the heliostat is correctly represented. In addition, we need to make sure that heliostats provide shading and blocking for other heliostats. For this, we use a small power plant consisting of one big receiver and two heliostats. The layout can be seen in Figure 17. The red rectangle represents a flat receiver with dimensions of 10 m by 10 m and a discretization of 100 by 100 pieces. The blue and green polygons are Tilt-and-Roll heliostats. The total width of each heliostat is 6.4 m and the height is 4.8 m. Both heliostats track the center of the receiver panel and have the same elevation. An overview of configuration parameters can be seen in Table 3.
To facilitate a clearer understanding of the reflective behavior and shading and blocking effects, the simulation is configured to position the sun behind the receiver and to disable tower shading. The number of rays was configured to 10000 per square meter. Such a high precision is only possible due to the low amount of heliostats as we would otherwise be limited by the simulation's memory footprint.
Because about half of the dark blue heliostat is shaded (and blocked) by the green heliostat, we expect that about half of the rays reflected by the dark blue heliostat to not reach the receiver panel. In Figure 18, we can see the flux map of the receiver.

Rather than calculating the total flux that is captured by the receiver panel, the flux map uses the pieces retrieved from discretization to calculate the flux that is absorbed by each individual piece. As a result, we can see which parts of the receiver capture the most flux. The expected effect can be seen in Figure 18 as the right half of the flux map has received less flux than the left half. The shape of the polygons of the facets is also apparent. The darker the color of a piece on the flux map, the more flux is received.
If we now move one heliostat three meters further away from the other heliostat, the flux map will be uniform and not show any reduced flux collection as no shading and blocking effects will occur.
Next, we will analyze the run time of the iterative tracking algorithm for Tilt-and-Roll heliostats.

5.2 Run Time of Tilt-and-Roll Heliostat Tracking Algorithm

The tracking algorithm for Tilt-and-Roll heliostats is iterative and terminates if the epsilon is small enough. Thus, we need to validate the amount of iterations needed to reach such a small epsilon. In this section, we will test the epsilon definitions introduced in Section 3.5. The first epsilon ϵ_p describes the distance between center points of two iterations. The second epsilon ϵ_a represents the angle between the normal vectors of two iterations.

Test Layout For this test, we use the ranges $[0°, 1°, \ldots, 90°]$ for the altitude and $[0°, 1°, \ldots, 180°]$ for the azimuth of the sun. The algorithm is tested on each pair of (azimuth, altitude) resulting in $91 \cdot 181 = 16471$ different sun positions. For each position, we use the new tracking algorithm to determine ϵ_p and ϵ_a. Figure 19 and Figure 20 show the development of ϵ_p and ϵ_a respectively. The absolute values and their standard deviation are shown in Table 4.

Results for Distance of Centers In Figure 19, we can see a plot with the iterations and corresponding epsilon ϵ_p in relation. The x-axis denotes the iteration starting with 0 up to 10. The y-axis shows the mean value of the ϵ_p (of all sun positions) for each iteration step. The scale of the y-axis is logarithmic with base 10. There are no values shown for iterations 7, 8, 9 and 10 as they are all zero and cannot be represented on a logarithmic scale. For each point in the plot, the lines above and below the point show the standard deviation of the respective iteration.
The algorithm starts with the mounting point as the current center. In the first iteration, we calculate the actual center of the mirror. The height of the mirror is 1.2 m, thus the center is at half of the height from the mounting point resulting in a fixed distance (and epsilon) of $1.2\,\text{m} / 2 = 0.6\,\text{m}$ for the first iteration, independent of the sun position. This can be seen in Figure 19 at iteration = 1. After the second iteration, this distance has dropped to a value of about 2 cm with a standard deviation of approximately 0.41 cm. With the third iteration, the mean has dropped to 0.22 mm

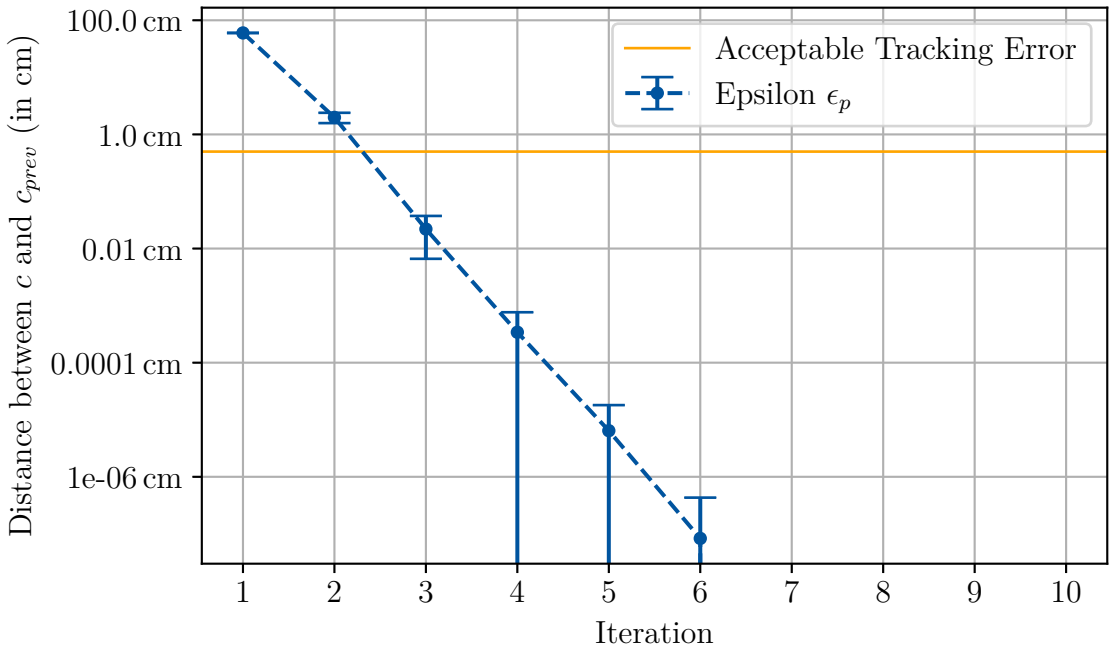


Figure 19: Effect of multiple iterations on the size of ϵ_p. The epsilon is based on the distance between the previous and current center points between iterations. After the third iteration, the improvement is negligible and below the acceptable tracking error. Iterations 7 through 10 show no improvement whatsoever and are thus absent from the figure. The scale of the y-axis is logarithmic with base 10.

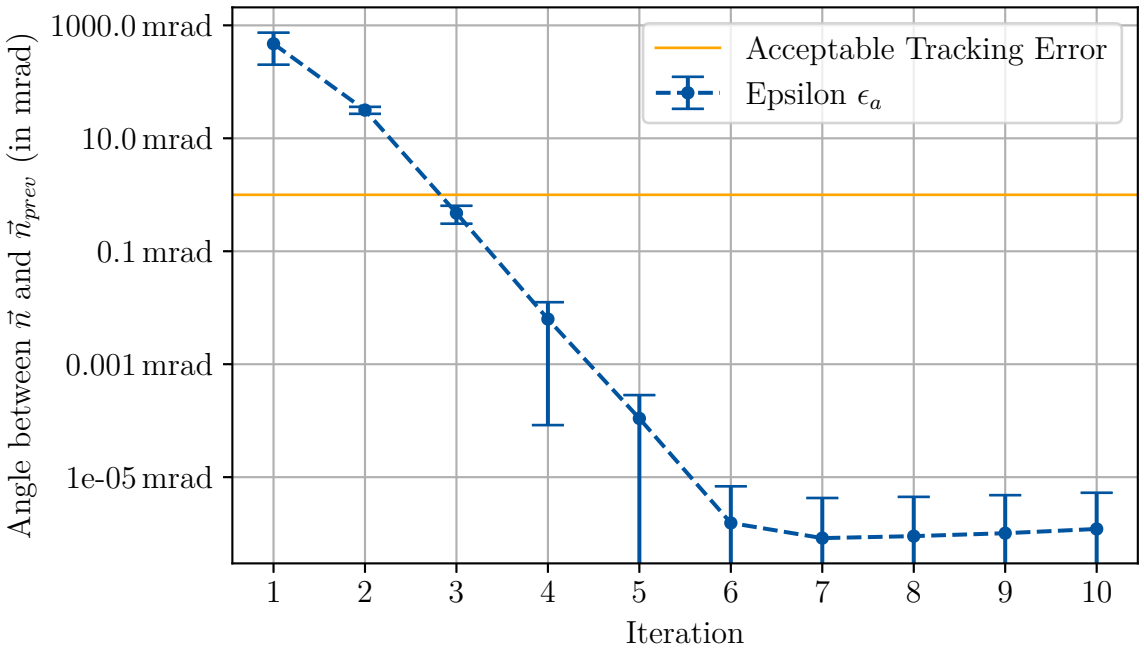


Figure 20: Effect of multiple iterations on the size of ϵ_a. The epsilon ϵ_a is based on the angle of the current and previous normal vector between iterations. After the third iteration, the improvement is negligible and the acceptable tracking error is reached. The scale of the y-axis is logarithmic with base 10.

and the standard deviation has reached 0.15 mm. This value continues to drop until iteration 6 after that the mean is so close to zero that it can no longer be represented by a 64-bit floating point number. The standard deviation also continuously shrinks with each iteration.

Results for Angle of Normal Vectors In Figure 20, a plot of iterations considering the angle between the current and previous normal vector is presented. The x-axis shows the iteration count. The y-axis denotes mean values of ϵ_a (of all sun positions) in mrad. The scale of the y-axis is logarithmic with base 10. The line below and above each point describes the standard deviation over all tested sun positions.
After the first iteration, the angle is at about 470 mrad which is well above the usual tracking error of a regular heliostat. The standard deviation of this iteration is approximately 270 mrad. The angle in the second iteration shrinks to about 31 mrad with a standard deviation of 4.45 mrad. This is still not good enough for accurate tracking. With the third iteration, the angle is at 0.47 mrad and has a standard deviation of 0.166 mrad. A regular heliostat has a tracking error of about 1 mrad. This makes three iterations good enough to achieve an accurate normal vector for ray tracing.

Iteration	Value ϵ_p (cm)	SD of ϵ_p (cm)	Value ϵ_a (mrad)	SD of ϵ_a (mrad)
1	60.00000	0.000000	472.331526	270.951350
2	1.985698	0.407506	31.614425	4.457986
3	0.021949	0.015337	0.473576	0.165788
4	0.000006	0.000423	0.006308	0.006224
5	0.000000	0.000012	0.000110	0.000174
6	0.000000	0.000000	0.000002	0.000005
7	0.000000	0.000000	0.000001	0.000003
8	0.000000	0.000000	0.000001	0.000004
9	0.000000	0.000000	0.000001	0.000004
10	0.000000	0.000000	0.000001	0.000004

Table 4: Values of ϵ_p and ϵ_a for each iteration rounded to the sixth decimal place. After the third iteration, the improvement is negligible. The unit of ϵ_p is cm and the unit of ϵ_a is mrad. SD denotes the standard deviation for ϵ_p and ϵ_a respectively.

5.3 Run Time of Simulation

In this section, we validate the run time of the simulation. The GPU implementation is not finished in time for this thesis, thus it is excluded from the comparison.
Our test is based on five power plants. We sample 10 iterations per test to rule out any fluctuations. All tests are run on a Debian 12 system with an AMD 5900x (12 cores, 24 threads at 3.7 GHz) CPU and 32 GiB of DDR4 3200 MHz RAM. The properties of the power plants being tested are shown in Table 2.

Analytic Ray Tracer During the development of the new features introduced in this thesis, we were able to speed up the calculation of shading & blocking candidates and improve the performance of ray generation and tracing. The discretization and AABB tree construction have become more complex and compute-intensive. This is reflected in the run time of the simulation. A summary can be seen in Figure 21. There we can observe that the performance of the new implementation is worse in most cases. Only the run time of the large abengoaCRS power plant is improved. Especially smaller power plants suffer from the more complex AABB tree creation overhead. In these power plants, fewer blocking & shading checks occur which are thus outweighed by the increased run time of the AABB tree creation.

Monte Carlo Ray Tracer The Monte Carlo ray tracer suffers from the same performance degradation as the analytic ray tracer. This can be seen in Figure 22. Power plants with large amounts of heliostats like abengoaCRS and Gemasolar only see a small increase in run time. The run time of other, smaller power plants, can double in some cases.

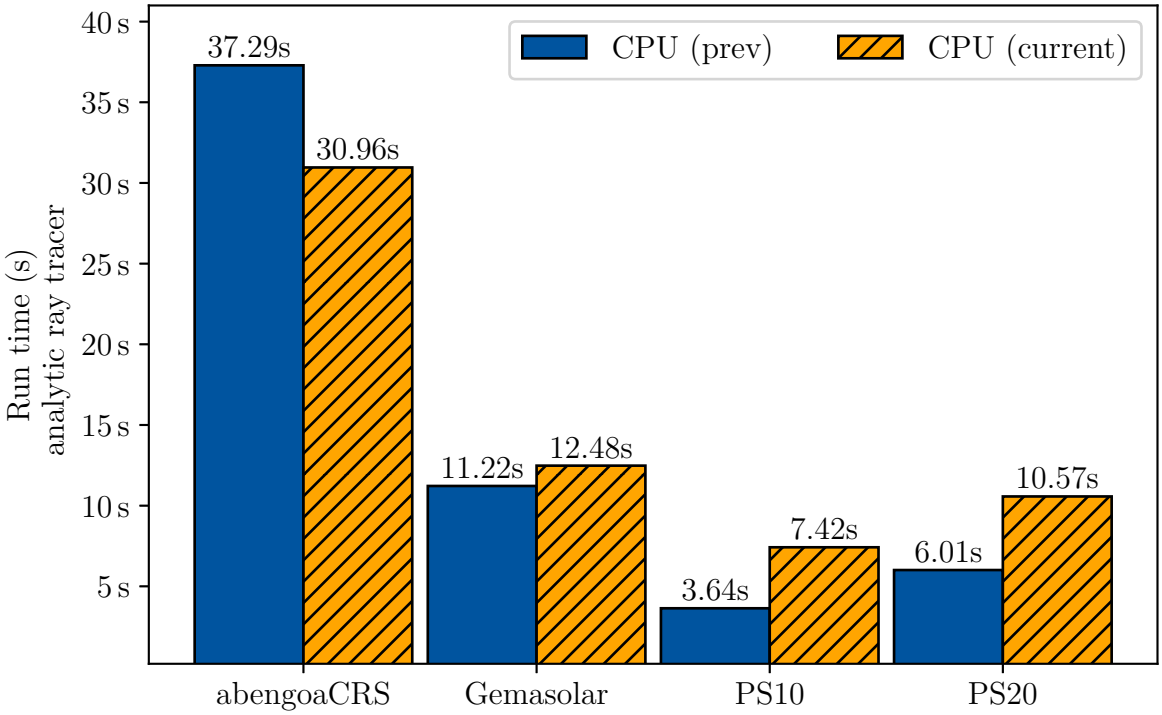


Figure 21: Before and after comparison of the analytic CPU ray tracer using 5 power plants and 10 rays/m^2. The before state is marked in blue. The after state is hatched and colored in orange. In most cases, the run time is increased. The run time of large CRS like abengoaCRS and Gemasolar see less impact of the more compute-intensive AABB tree creation as they contain more heliostats. Thus, AABB tree creations are less represented and more shading & blocking checks have to take place.

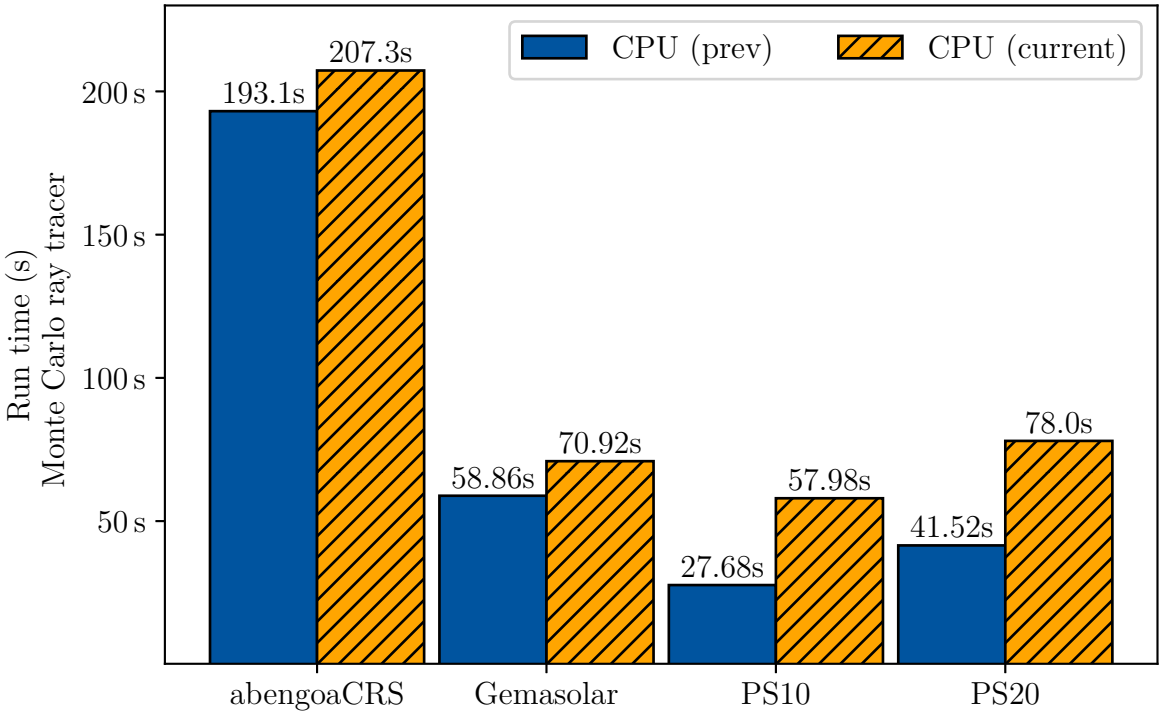


Figure 22: Before and after comparison of the Monte Carlo CPU ray tracer using 5 power plants, 10 rays/m^2 and a ray multiplier of 10. The before state is marked in blue. The after state is hatched and colored in orange. The run time of large CRS like abengoaCRS and Gemasolar see less impact of the more compute-intensive AABB tree creation as they contain more heliostats. Thus, AABB tree creations are less represented and more shading & blocking checks have to take place.

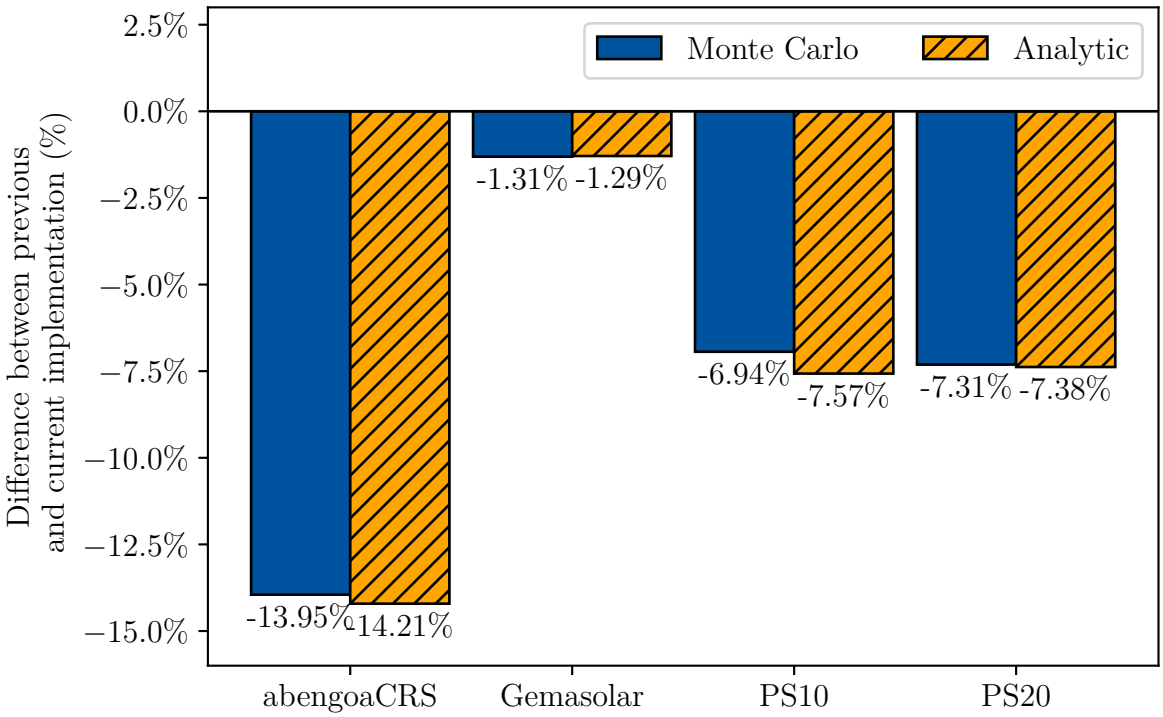


Figure 23: Maximum RAM usage of simulations on different power plants using the analytic (marked in orange) and Monte Carlo (denoted in dark blue) ray tracer with 10 rays/m^2 and a ray multiplier of 10. Numbers were obtained using the GNU `time` utility.

5.4 RAM Usage of Simulation

As the accuracy of simulations increases and the number of heliostats being modeled grows, so does the memory requirement. Consequently, reducing RAM utilization becomes increasingly crucial. For this test, we measure the maximum RAM usage of the analytic and Monte Carlo ray tracer. Measurements were taken using the GNU `time` utility[5]. In Figure 23, we can see a reduction of RAM usage for all power plants on all ray tracers. The dark blue bars show the reduction using the Monte Carlo ray tracer while the hatched orange bars represent the reduction using the analytic ray tracer. Table 5 shows the absolute values and the difference for each power plant and ray tracer. Especially for the big abgengoaCRS power plant, we were able to save almost 1.5 GB in both ray tracers. This equates to savings of around 14 %. PS10 and PS20 also see a reduction of around 7 %. With Gemasolar the saving is less at around 1.3 %. In general, the memory usage is reduced across both ray tracers.

Power Plant	RAM Usage (prev)	RAM Usage (current)	Savings
Monte Carlo			
abengoaCRS	10629.64 MB	9146.61 MB	1483.03 MB
Gemasolar	2947.86 MB	2909.2 MB	38.66 MB
PS10	743.2 MB	691.62 MB	51.58 MB
PS20	1480.84 MB	1372.56 MB	108.28 MB
Analytic			
abengoaCRS	10452.48 MB	8966.92 MB	1485.56 MB
Gemasolar	2872.34 MB	2835.24 MB	37.1 MB
PS10	728.19 MB	673.04 MB	55.15 MB
PS20	1445.6 MB	1338.96 MB	106.64 MB

Table 5: RAM usage of different power plants using the analytic and Monte Carlo ray tracer. The usage is decreased in all cases. Lower usage enables the simulation of bigger power plants with higher accuracy.

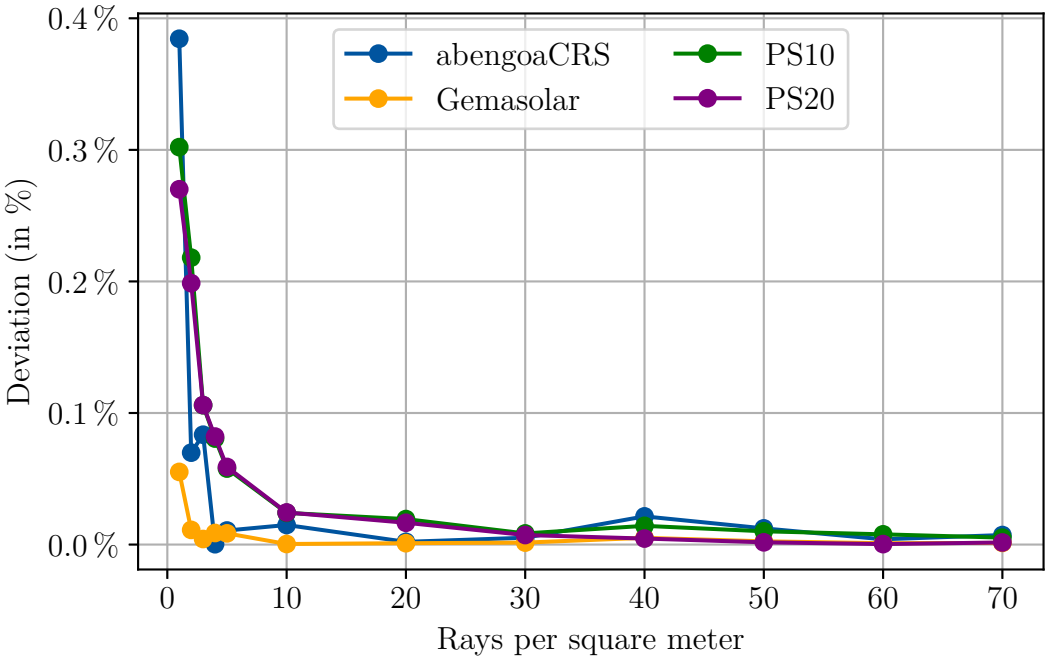


Figure 24: Difference of optical power for each power plant in per mille. Simulations were run using the analytic ray tracer with different accuracy levels. All values are in relation to the ideal optical power determined using 100 rays per square meter.

5.5 Correctness of Simulation

To verify the correctness of the new implementation of the simulation, we consider the total optical power of select power plants. For each power plant, we run multiple annual simulations and obtain the total optical power collected by the receiver. As these simulations incorporate a lot of variables, the result can be used well to compare the accuracy and validity of our implementation. The simulations differ in the configured accuracy. We consider 1, 2, 3, 4, 5, 10, 20, 30, 40, 50, 60 and 70 rays per square meter for each individual power plant. An overview of the deviation of all power plants in relation to the ideal result can be seen in Figure 24. The ideal result is obtained by an annual simulation with 100 rays per square meter. With an increasing number of rays, the total optical power converges exponentially towards the ideal value. After 50 rays per square meter, the improvement is negligible. All tested power plants reliably improve the accuracy with more rays. This result validates the correctness and accuracy of our implementation.

[5] `https://www.gnu.org/software/time/`

6 Conclusion and Outlook

In this thesis, we extended the existing modeling capabilities to enable the simulation of Tilt-and-Roll heliostats. A facet of a heliostat can now be defined as an irregular polygon, delivering immense flexibility in the development of new facet shapes even beyond Tilt-and-Roll heliostats. Miscellaneous objects like (servo) motors or mounting equipment as part of the heliostat's model can also now be considered during simulation. Due to the different rotational properties of Tilt-and-Roll heliostats, the previously used tracking algorithm for azimuth-elevation tracking heliostats cannot be used for Tilt-and-Roll heliostats. We defined an iterative approach for a novel tracking algorithm which can properly align Tilt-and-Roll heliostats with the sun and receiver. Its run time and accuracy were validated. The existing AABB tree implementation was overhauled to work with the new facet shapes and layouts of Tilt-and-Roll heliostats and also to increase flexibility in development of future heliostat models. To improve the simulation of inexpensive mirror material and complex canting concepts, we introduced measurement-driven simulation. There, we use a scan of a heliostat surface to accurately model real world production deficiencies.
With these features implemented, mCRS using Tilt-and-Roll heliostat can now be simulated without limiting the ability to simulate existing azimuth-elevation tracking heliostats.
The correctness and accuracy of the newly implemented features was validated. We were able to achieve a noticeable decrease in RAM usage. The run time only slightly increased for large power plants while up to doubling for smaller power plants. The novel tracking algorithm enables accurate alignment using only three iterations.
To improve the run time of the new features, the AABB tree needs to be optimized further. This can potentially be done by using different construction methods depending on the situation. The receiver has a fixed discretization structure and thus a bottom-up construction approach could be beneficial.
Other requirements for mCRS power plants also need to be considered for accurate simulation of such power plants. This includes the simulation of miscellaneous objects in the scene that result in additional shading and blocking effects. These objects could be buildings or trees.

References

[1] L. Aldenhoff. Ray tracer for central receiver systems using gpu. Master's thesis, RWTH Aachen University, 2021.

[2] B. Belhomme, R. Pitz-Paal, P. Schwarzbözl, and S. Ulmer. A new fast ray tracing tool for high-precision simulation of heliostat fields. *Journal of Solar Energy Engineering*, 131(3), June 2009. ISSN 1528-8986. doi: 10.1115/1.3139139.

[3] P. Blanc, B. Espinar, N. Geuder, C. Gueymard, R. Meyer, R. Pitz-Paal, B. Reinhardt, D. Renné, M. Sengupta, L. Wald, and S. Wilbert. Direct normal irradiance related definitions and applications: The circumsolar issue. *Solar Energy*, 110: 561577, Dec. 2014. ISSN 0038-092X. doi: 10.1016/j.solener.2014.10.001.

[4] M. J. Blanco, J. M. Amieva, and A. Mancillas. The tonatiuh software development project: An open source approach to the simulation of solar concentrating systems. In *Computers and Information in Engineering*, IMECE2005. ASMEDC, Jan. 2005. doi: 10.1115/imece2005-81859.

[5] R. Buck and E. Teufel. Comparison and optimization of heliostat canting methods. *Journal of Solar Energy Engineering*, 131(1), Jan. 2009. ISSN 1528-8986. doi: 10.1115/1.3027500.

[6] J. I. Burgaleta, S. Arias, and D. Ramirez. Gemasolar, the first tower thermosolar commercial plant with molten salt storage. *SolarPACES, Granada, Spain*, pages 20–23, 2011.

[7] X. Duan, C. He, X. Lin, Y. Zhao, and J. Feng. Quasi-Monte Carlo ray tracing algorithm for radiative flux distribution simulation. *Solar Energy*, 211:167–182, 2020. doi: 10.1016/j.solener.2020.09.061.

[8] G. Eder, M. Held, and P. Palfrader. Parallelized ear clipping for the triangulation and constrained delaunay triangulation of polygons. *Computational Geometry*, 73:1523, Aug. 2018. ISSN 0925-7721. doi: 10.1016/j.comgeo.2018.01.004.

[9] T. Farr, P. Rosen, E. Caro, R. Crippen, R. Duren, S. Hensley, M. Kobrick, M. Paller, E. Rodriguez, L. Roth, et al. The shuttle radar topography mission. *Reviews of geophysics*, 45(2), 2007.

[10] L. Fischer. Multi-step layout-optimization of heliostat fields in central receiver systems. Master's thesis, RWTH Aachen University, 2021.

[11] D. Gebreiter, G. Weinrebe, M. Wöhrbach, F. Arbes, F. Gross, and W. Landman. sbpray–a fast and versatile tool for the simulation of large scale csp plants. In *AIP Conference Proceedings*, volume 2126, page 170004. AIP Publishing LLC, 2019. doi: 10.1063/1.5117674.

[12] F. Hövelmann. Accelerated raytracer for solar tower power plants. Bachelor thesis, RWTH Aachen University, 2019.

[13] F. Hövelmann. Heuristic layout optimization of central receiver systems. Master's thesis, RWTH Aachen University, 2022.

[14] M. Izygon, P. Armstrong, C. Nilsson, and N. Vu. Tiesol–a gpu-based suite of software for central receiver solar power plants. *Proceedings of SolarPACES*, 2011.

[15] H. Kambezidis, B. Psiloglou, D. Karagiannis, U. Dumka, and D. Kaskaoutis. Meteorological radiation model (mrm v6.1): Improvements in diffuse radiation estimates and a new approach for implementation of cloud products. *Renewable and Sustainable Energy Reviews*, 74:616637, July 2017. ISSN 1364-0321. doi: 10.1016/j.rser.2017.02.058.

[16] P. L. Leary and J. D. Hankins. *Users guide for MIRVAL: a computer code for comparing designs of heliostat-receiver optics for central receiver solar power plants.* Feb. 1979. doi: 10.2172/6371450.

[17] L. Li, J. Coventry, R. Bader, J. Pye, and W. Lipiski. Optics of solar central receiver systems: a review. *Optics Express*, 24(14):A985, May 2016. ISSN 1094-4087. doi: 10.1364/oe.24.00a985.

[18] R. Pitz-Paal, N. B. Botero, and A. Steinfeld. Heliostat field layout optimization for high-temperature solar thermochemical processing. *Solar Energy*, 85(2):334343, Feb. 2011. ISSN 0038-092X. doi: 10.1016/j.solener.2010.11.018.

[19] P. Richter. *Simulation and optimization of solar thermal power plants.* Dissertation, RWTH Aachen University, Aachen, 2017. Veröffentlicht auf dem Publikationsserver der RWTH Aachen University; Dissertation, RWTH Aachen University, 2017.

[20] P. Richter and F. Hövelmann. Computationally fast analytical ray-tracer for central receiver systems. In *SOLARPACES 2020: 26th International Conference on Concentrating Solar Power and Chemical Energy Systems.* AIP Publishing, 2022. doi: 10.1063/5.0085714.

[21] M. Schmitz, P. Schwarzbözl, R. Buck, and R. Pitz-Paal. Assessment of the potential improvement due to multiple apertures in central receiver systems with secondary concentrators. *Solar Energy*, 80(1):111120, Jan. 2006. ISSN 0038-092X. doi: 10.1016/j.solener.2005.02.012.

[22] P. Schramek and D. R. Mills. Heliostats for maximum ground coverage. *Energy*, 29(56):701713, Apr. 2004. ISSN 0360-5442. doi: 10.1016/s0360-5442(03)00178-6.

[23] T. Wendelin. Soltrace: A new optical modeling tool for concentrating solar optics. In *Solar Energy*, ISEC2003. ASMEDC, Jan. 2003. doi: 10.1115/isec2003-44090.

[24] K. Yoshida, S. Miwa, H. Yamaki, and H. Honda. Analyzing the impact of cuda versions on gpu applications. *Parallel Computing*, 120:103081, June 2024. ISSN 0167-8191. doi: 10.1016/j.parco.2024.103081.