Fast ray tracing in phase space for optical design

Portegies, J.M.

Award date:
2014
Fast ray tracing in phase space for optical design

Jorg Portegies

Eindhoven University of Technology,
Department of Mathematics and Computer Science,

Philips Lighting

September 2013
Publishing date: 27-01-2015
Abstract

Philips Lighting is interested in designing optical systems that produce desired light characteristics, such as luminous intensity, on a target. Ray tracing is one way to relate the photometric properties of the light source to the characteristics of the light at the target. Monte Carlo methods are commonly used for illumination ray tracing, but these are computationally expensive.

When light travels through an optical system, the flux and étendue are conserved and hence the luminance is constant along a ray. The goal of ray tracing methods is to find how the luminance, defined as a function on the optical phase space of the source, behaves as a function on the optical phase space of the target. From this, other light characteristics can be derived. Even if the luminance of the source is a well-behaved function, the luminance on the target has discontinuities. These discontinuities occur at the boundaries of regions of rays that have followed a similar path through the optical system. An iterative ray trace process is designed to approximate these boundaries. Then a multilevel B-spline interpolation algorithm is used to interpolate the luminance that is known only on a finite data set. This method creates a function that can be integrated analytically, to obtain the luminous intensity or the illuminance. Results show that the phase space ray trace method can be a significant improvement to the classical Monte Carlo ray tracer.
Acknowledgements

The present report serves as a Master Thesis for the master Industrial and Applied Mathematics from the Eindhoven University of Technology. The project was carried out as an internship at Philips Lighting.

First of all I would like to thank my supervisor at the university, Jan ten Thije Boonkkamp, not only for bringing me in contact with Philips, but mostly for all the support during the project. By showing his interest in my problems, asking me critical questions and proofreading my report numerous times, he has had a great influence on the project and finally the report.

I wish to express my great appreciation to my supervisors at Philips, Wilbert IJzerman and Teus Tukker, for giving me the opportunity to carry out my thesis in the form of an internship. I have learned a lot from them since day 1 at Philips and I really value all the useful and creative input they had during our weekly meetings. They also let me present my work at the Nederlands Mathematisch Congres in Nijmegen halfway through the project, which was a very nice opportunity for me.

I wish to thank the rest of my colleagues at Philips for making my internship an enjoyable experience by joining me for coffee, lunch, ice cream on the hot summer days and of course by sharing their optical expertise with me.

Finally, I am my friends, girlfriend and family infinitely grateful for giving me the right amount of distraction from and encouragement for my entire study and in particular this thesis.

Jorg Portegies
Contents

1 Introduction 4
  1.1 Context ........................................ 4
  1.2 Ray tracing in a nutshell ......................... 5
  1.3 Research goals ................................ 5
  1.4 Contents ...................................... 7

2 Physical background for ray tracing 8
  2.1 A concise overview of basic optics ............... 8
    2.1.1 Light rays ................................ 8
    2.1.2 Fermat’s principle ......................... 9
    2.1.3 Reflection and refraction ................ 10
  2.2 Photometric quantities ........................ 11
    2.2.1 Definitions in 3D ........................ 11
    2.2.2 Analogues in 2D .......................... 13
  2.3 2D ray tracer .................................. 14
    2.3.1 Linear segments ........................... 15
    2.3.2 Splines .................................... 16
  2.4 Other fundamental information ................... 17
    2.4.1 Constant luminance along a ray ............ 17
    2.4.2 Lambert’s cosine law ...................... 18
    2.4.3 Monte Carlo ray tracing .................... 19

3 Phase space 21
  3.1 Mathematical formulation ....................... 21
  3.2 Exact approach ................................ 23
    3.2.1 Two-faceted cup ........................... 23
    3.2.2 CPC ...................................... 25
    3.2.3 Non-constant luminance ................... 28

4 Hamiltonian optics 29
  4.1 Fermat’s principle and the Lagrangian .......... 29
  4.2 Derivation of the Hamiltonian .................. 31
  4.3 Hamiltonian on the screen ...................... 32
  4.4 Symplectic integration of the Hamiltonian ...... 33
  4.5 Numerical results for the symplectic integration of a light ray 34
5 Numerical phase space ray tracing

5.1 From source to target in giant steps ...................................................... 38
5.2 Finding the boundaries of the $M$-regions ........................................... 39
  5.2.1 Grid refining algorithm ................................................................. 39
  5.2.2 Triangulation .................................................................................. 42
  5.2.3 $\alpha$-shapes .................................................................................. 43
5.3 Interpolation for scattered data ............................................................... 45
  5.3.1 Shepard’s interpolation ................................................................. 45
  5.3.2 B-spline interpolation ................................................................. 46
  5.3.3 Accuracy of B-spline interpolation .................................................. 51
5.4 Integration of the luminance ................................................................. 54

6 Results ............................................................................................................. 58

6.1 Faceted cups ............................................................................................. 58
6.2 CPC ............................................................................................................ 61
6.3 TIR Collimator .......................................................................................... 63
6.4 Calculation times ...................................................................................... 65

7 Conclusions and recommendations ................................................................ 68

7.1 From two to three dimensions ................................................................. 69

A List of frequently used symbols .................................................................. 71

B Phase space pictures ................................................................................... 72
Chapter 1

Introduction

1.1 Context

Over the years, many different ways to produce electric light have been discovered. Examples are the light bulb, an incandescent lamp that produces light as a current passes through a thin wire causing its temperature to rise until it glows, and the fluorescent lamp, that contains a gas that produces UV-light as a current passes through that is converted to visible light with use of phosphors. Nowadays, increasing interest goes out to the Light Emitting Diode (LED), a semiconducting solid that illuminates when a current flows through [12]. Whereas a light bulb produces (warm) white light, the first LEDs produced light of a certain color. The first visible spectrum LEDs produced a red light and were commercially used as indicator lamps and in seven-segment displays already in the early 60s, later also green and blue LEDs have been discovered and much later white LEDs. Meanwhile their efficiency has increased exponentially, so much that they have now become popular for use in general lighting. LEDs now offer far more possibilities than simple displays only: they can be combined in such a way that they are powerful enough for car headlights or to illuminate New York’s Empire State Building, Figure 1.1. They can be used as a simple replacement of a light bulb, but LEDs with different colors can also be combined in one lamp or light fixture (luminaire), that can be controlled by a mobile phone, see Figure 1.2.

Each of the applications has its optical challenge: the car headlight should be powerful, but not blinding for oncoming traffic, when replacing a simple light bulb we want to have a lamp that fits the original fitting, and if a consumer wants the Hue to turn yellow, the colors of the LEDs should be mixed in such a way that the lamp indeed gives a yellow color. In particular this mixing of colors is challenging. If in one lamp two red, two green and two blue LEDs are combined with the purpose to create a white light source, it is difficult to create a beam of light with a uniform color for all positions and angles at which the light leaves the lamp.

These properties play an important role when designing an optical system and software packages are very capable of simulating light sources with LEDs. These so-called ray tracers are used to determine the light properties of an LED system by tracing a large number of rays through a model of the lamp or luminaire. However, as the complexity of an optical design increases, more calculation time is needed to obtain accurate results. The functionality of a ray tracer is briefly explained in the next section.
1.2 Ray tracing in a nutshell

The concept of ray tracing is not difficult to understand, but since it is the central topic of this report, it is good to describe as early as possible the basic principles of a ray tracer. Figure 1.3 shows a two-dimensional optical system, containing a light source and a target screen, both normal to the optical axis, and in between a gray area that may contain lenses, reflective surfaces or other objects that influence the propagation of light. A ray tracer simply calculates how light rays emitted from the source propagate through the optical system and at what position and with what angle they hit the target screen, while keeping track of, for example, the energy flux of every ray. The more rays are traced, the more accurately the output of the simulation reflects the actual light source.

Current commercial ray tracers have an impressive list of features and have put a lot of effort in increasing their speed, but still a simulation can take days if an optical designer desires a reliable output, for example to obtain a color analysis of a system.

1.3 Research goals

There are numerous ways to speed up the process of ray tracing itself, i.e. trace as many rays as possible in a fixed amount of time. More on this field of research can be found in, e.g., [15]. Instead, we aim to improve the post processing of the acquired ray trace data. We look for an approach that makes use of the optical phase space, which explains the name phase space ray tracing. It is a completely new way of ray tracing and it is yet unclear what steps
need to be taken to actually get a meaningful output from such a ray tracer.

Despite the unclarities there is good hope that such a ray tracer will work, so the main goal of this study is to build one and to overcome the difficulties that might come with that process. We simplify the problem by focussing solely on 2D symmetric optical systems, but the methods used should be chosen such that they allow for a generalization to three dimensions.

The next important step is to see how much time can be gained compared to standard Monte Carlo ray tracing. For comparison, we use a very basic Monte Carlo tracer that has
no improved performance. We will investigate the number of traced rays necessary for a certain accuracy, the computational time this saves us and how much time is taken by the post processing of the data.

1.4 Contents

The structure of this report is as follows: in Chapter 2 an overview is provided of the physics (optics) that is required to understand the rest of the report and, in more detail, the process of ray tracing is explained. Chapter 3 puts ray tracing in a mathematical framework and examples of optical systems are shown for which the light properties can be calculated with an analytical method. Chapter 4 explains some of the theory of Hamiltonian optics that might be used later in a ray trace application, but for now it stands a bit on its own.

Chapters 5 and 6 are the core of this report. Here the entire numerical ray trace protocol is explained in detail and the performance is checked with various examples. Conclusions and recommendations can be found in Chapter 7.
Chapter 2

Physical background for ray tracing

This chapter provides the necessary physical background to understand the concept of ray tracing, starting with some properties of light propagation, followed by an explanation of the physical quantities that play a role. Then in more detail is explained how and why a ray tracer works.

2.1 A concise overview of basic optics

Light is an interesting physical phenomenon and many pages can be written about optics. In this report, the focus lies on the propagation of light. It is well known that light behaves both as particle and as wave. Light photons propagate in vacuum with speed \( c = 3.00 \cdot 10^8 \) m/s, but when light is transmitted through a medium, the apparent velocity \( v \) can differ. The refractive index \( n \) is a material property that characterizes this shift in velocity: \( n = c/v \).

If the light has a wavelength between 400 nm (violet) and 700 nm (red) it is in the visible spectrum.

2.1.1 Light rays

A useful model for the visualization of light is the light ray, a line in space with the same direction as the flow of radiant energy [7]. In a homogeneous and isotropic medium, rays are straight lines and perpendicular to the wavefront of the light. Also photometric quantities can be assigned to these light rays, such as energy flux (amount of energy per time unit) or luminance (a slightly more involved photometric quantity that will turn out to be very useful, see Section 3.1). Light rays can be parameterized by the arc length. A Cartesian or cylindrical coordinate system is used to define the position of a ray. Light sources are often placed in the \( xy \)-plane, normal to the \( z \)-axis, the optical axis, around which there can be a rotational symmetry. A meridional ray is a ray that is traced in a plane containing the optical axis. Rays that do not have this property are called skew rays. A two-dimensional ray tracer traces only the meridional rays of a three-dimensional ray tracer, but does not give any information about the skew rays. Therefore, the output of a 2D ray tracer cannot be translated to 3D. Another special type of rays are the edge rays, in this report used to denote the rays that are emitted from the edges of the light source.
2.1.2 Fermat’s principle

Light emitted from a source can encounter several objects before hitting a target. The question of how a beam of light is influenced by these obstacles can be answered with Fermat’s principle. When light traverses from point \(A\) to point \(B\) along a curve \(C\), the optical path length is given by

\[
S = \int_C n ds. \tag{2.1}
\]

Fermat’s principle states that an actual light ray joining \(A\) and \(B\) is a curve connecting the two points that has a stationary optical path length compared to neighbouring curves connecting \(A\) and \(B\) [1]. In two dimensions, Equation (2.1) can be rewritten in terms of the parameter along the optical axis \(z\) and the function \(f(z)\), such that \(x = f(z)\) defines the ray path:

\[
\int_C n(z, x) ds = \int_{z_1}^{z_2} n(z, f(z)) \sqrt{1 + f'(z)^2} dz. \tag{2.2}
\]

The function \(f\) should then be such that \(S\) is stationary compared to neighbouring functions. This means that the first variation \(\delta S\) should vanish [10], i.e., for \(f = f_0 + \varepsilon f_1\) with \(f_1(z_1) = f_1(z_2) = 0\) we should have

\[
\delta S = \frac{d}{d\varepsilon} \int_{z_1}^{z_2} n(z, f_0 + \varepsilon f_1) \sqrt{1 + (f_0' + \varepsilon f_1')^2} dz |_{\varepsilon=0} = 0. \tag{2.3}
\]

This last expression can be expanded by interchanging integration and differentiation:

\[
\delta S = \int_{z_1}^{z_2} \frac{\partial}{\partial \varepsilon} n(z, f_0 + \varepsilon f_1) \sqrt{1 + (f_0' + \varepsilon f_1')^2} dz \bigg|_{\varepsilon=0} = \int_{z_1}^{z_2} \partial_x n(z, f_0 + \varepsilon f_1) \sqrt{1 + (f_0' + \varepsilon f_1')^2} f_1 + \frac{n(z, f_0 + \varepsilon f_1)(f_0' + \varepsilon f_1') f_1'}{\sqrt{1 + (f_0' + \varepsilon f_1')^2}} dz \bigg|_{\varepsilon=0}
\]

\[
= \int_{z_1}^{z_2} \partial_x n(z, f_0) \sqrt{1 + (f_0')^2} f_1 + \frac{n(z, f_0) f_0' f_1'}{\sqrt{1 + (f_0')^2}} dz. \tag{2.4}
\]

With use of integration by parts of the second term and the fact that \(f_1\) vanishes at the endpoints, we get

\[
\delta S = \int_{z_1}^{z_2} f_1 \partial_x n(z, f_0) \sqrt{1 + (f_0')^2} - f_1 \frac{d}{dz} \left( \frac{n(z, f_0) f_0'}{\sqrt{1 + (f_0')^2}} \right) dz = 0. \tag{2.5}
\]

This should hold for any (continuous) function \(f_1\), so we have

\[
\frac{d}{dz} \left( \frac{n(z, f_0) f_0'}{\sqrt{1 + (f_0')^2}} \right) = \partial_x n(z, f_0) \sqrt{1 + (f_0')^2}. \tag{2.6}
\]

Now if the refractive index is constant, the derivative on the right hand side vanishes and we find that \(f' = f_0' = \text{constant}\), and therefore \(f\) is a straight line. In a similar fashion, one can derive the laws of reflection and refraction.
2.1.3 Reflection and refraction

Figure 2.1: Refraction (a) and reflection (b).

Figure 2.1 illustrates the law of reflection and the law of refraction, also known as Snell’s law. We now derive a proof from Fermat’s principle, starting with refractions. Suppose we have two materials with refractive indices \( n_1 \) and \( n_2 \) that have a common boundary at \( z = 0 \). Inside both media, the light ray is straight, as we have shown, but this does not hold at the boundary. We applied integration by parts to obtain Equation (2.5), but since in this case \( n \) is not differentiable at \( z = 0 \), we can only apply partial integration to the separate regions \( z > 0 \) and \( z < 0 \). Again we assume that \( f_1(z_1) = f_1(z_2) = 0 \). This leads to

\[
\delta S = \int_{z_1}^{0} f_1 \dot{\theta}_2 n_1 \sqrt{1 + (\dot{f}_0)^2} - f_1 \frac{d}{dz} \left( \frac{n_1 f_0'}{\sqrt{1 + (f_0')^2}} \right) \right) dz \\
+ \int_{0}^{z_2} f_1 \dot{\theta}_2 n_2 \sqrt{1 + (\dot{f}_0)^2} - f_1 \frac{d}{dz} \left( \frac{n_2 f_0'}{\sqrt{1 + (f_0')^2}} \right) \right) \right) dz \\
+ f_1(0) \left( \frac{n_1 f_0'(0^+)}{\sqrt{1 + (f_0'(0^+))^2}} - \frac{n_2 f_0'(0^-)}{\sqrt{1 + (f_0'(0^-))^2}} \right) = 0. \tag{2.7}
\]

Since this should hold for all \( f_1 \), all three terms should vanish individually. From the first two terms we obtain as before that the rays are straight in the regions \( z < 0 \) and \( z > 0 \). The last term gives us

\[
\frac{n_1 f_0'(0^+)}{\sqrt{1 + (f_0'(0^+))^2}} = \frac{n_2 f_0'(0^-)}{\sqrt{1 + (f_0'(0^-))^2}}. \tag{2.8}
\]

We can rewrite the left term as

\[
\frac{n_1 f_0'(0^+)}{\sqrt{1 + (f_0'(0^+))^2}} = \frac{n_1 \frac{dx}{ds}}{\sqrt{1 + (\frac{dz}{ds})^2}} = \frac{n_1 \frac{dx}{ds}}{\sqrt{1 + (\frac{dz}{ds})^2 + (\frac{dx}{ds})^2}} = n_1 \frac{dx}{ds} = n_1 \sin \theta_1 \tag{2.9}
\]
and similar for the right term. The law of refraction then follows:

\[ n_1 \sin \theta_1 = n_2 \sin \theta_2. \tag{2.10} \]

A ray that travels from a point \( A \) to a point \( B \) via a reflective surface as in Figure 2.1b has the same optical path length as a ray that travels from \( A \) to \( B' \) with constant index of refraction. It follows from the previous calculations that the ray is a straight line, so that

\[ \theta_1 = \theta_2. \tag{2.11} \]

Other types of reflection are Fresnel reflection and diffuse reflection, see Figure 2.2. In some cases when light travels between two media with a different index of refraction, the light is partly reflected and partly refracted. This is called Fresnel reflection. Diffuse reflection is when light falls onto a surface and is scattered, i.e., reflected in every direction. These two types are disregarded in this report, we assume that only specular reflection occurs.

![Figure 2.2: Fresnel reflection (a) and diffuse reflection (b).](image)

### 2.2 Photometric quantities

In this section the photometric quantities luminous flux, luminous intensity and luminance are defined. They differ from radiometric quantities in the sense that they are adapted to the sensitivity of the human eye. The normalized sensitivity of the human eye as a function of the wavelength \( \lambda \) of light is given by the luminosity function \( \bar{y}(\lambda) \), see Figure 2.3. With use of this function, the luminous flux, luminous intensity and luminance can be related to radiant flux, radiant intensity and radiance respectively. Below, these quantities are defined in three dimensions and subsequently the 2D analogues are given. Throughout this report we will encounter these quantities defined on a light source, indicated with a subscript \( s \), or on a target screen, indicated with \( t \).

#### 2.2.1 Definitions in 3D

- The radiant energy flux \( \Phi_r \) is the total energy emitted from a source or accepted by a target per time unit [2]. The luminous flux \( \Phi \) is a measure for the perceived power of light. The total emitted (weighted) radiation is defined as

\[ \Phi = 683 \int_0^\infty \Phi_r(\lambda)\bar{y}(\lambda)\,d\lambda, \tag{2.12} \]
where $\bar{y}$ is the luminosity function. $\Phi$ has unit lumen [lm]. In a closed, ideal optical system, the flux emitted from the source is the same as the flux that hits the target.

- The luminous flux per solid angle $d\Omega$ is called \textit{luminous intensity}, given by

$$I = \frac{d\Phi}{d\Omega} \quad [\text{cd} = \frac{\text{lm}}{\text{sr}}].$$

The intensity of a light source can give information about the beam width: a broader intensity profile means a wider beam.

- The luminous flux falling on a surface area $dA$ is called \textit{illuminance} and is defined as

$$E = \frac{d\Phi}{dA} \quad [\text{lm} m^{-2}].$$

Equivalently we can define the \textit{luminous emittance} (or exitance) as the luminous flux per unit area emitted from a surface.

- \textit{Etendue} is a measure for how much area and angular space light requires as it travels through an optical system. In three dimensions, étendue can be defined in its differential form as

$$dU = n^2 dA \cos t d\Omega \quad [\text{m}^2 \text{sr}],$$

in which $dA$ is an infinitesimal area, $d\Omega$ a solid angle and $n$ the index of refraction of the material in which the light is located. $t$ is the angle that the centre line of the solid angle makes with the optical axis. Etendue is an important concept in nonimaging optics, since it is conserved in an ideal optical system.

- The flux that is emitted from a source with area $dA$ at an angle $t$ to the normal of this area, contained in a solid angle $d\Omega$ is called the \textit{luminance} $L$ and is defined as
In other words, it indicates how much of the power emitted from the source is received by a viewer looking at the system at a certain angle. This also explains the \( \cos t \) in the denominator, since the apparent surface is given by \( dA \cos t \).

### 2.2.2 Analogues in 2D

In this report, we reduce the complexity of the problem by considering two dimensional optical systems only. We need to find two dimensional analogues to the definitions above.

- The luminous flux is defined exactly the same.
- The luminous intensity is the luminous flux per angle \( dt \) and becomes:

\[
I = \frac{d\Phi}{dt} \left[ \frac{\text{lm}}{\text{rad}} \right].
\]

(2.17)

- The illuminance now denotes the luminous flux falling on a line segment \( da \):

\[
E = \frac{d\Phi}{da} \left[ \frac{\text{lm}}{\text{m}} \right].
\]

(2.18)

- In two dimensions, the étendue can be defined in its differential form as

\[
dU = nda \cos t dt,
\]

(2.19)
in which \( da \) is an infinitesimal length and \( t \) the angle with the \( z \)-axis as in Figure 2.4.

![Figure 2.4: Etendue in two dimensions.](image)

- Since the étendue has changed, the luminance changes accordingly:

\[
L = \frac{d^2\Phi}{da \cos t dt}.
\]

(2.20)
2.3 2D ray tracer

In principle, a ray tracer needs two inputs to do its job properly: it needs a set of rays, described by their starting point and starting angle at the source, and it needs to know the optical system, i.e., it needs to know where the reflective and refractive surfaces are located. There are various ways in which the ray tracer can be provided with the necessary input about the optics, but in our implementation is chosen for the following setup, where the input consists of:

- a set of points that lie on the reflective or refractive surfaces.
- a set of segments, that describe which points are connected to each other and in what way. A straight segment is created by connecting two points, a curved segment is created by making a spline of an arbitrary set of points on the segment.
- a vector that describes for every segment if it is reflective or refractive.

![Figure 2.5: The coordinate system as used throughout the report.](image)

The ray with starting point \( x_0 \) and initial angle \( t \) with the optical axis, see Figure 2.5, is described by

\[
x = x_0 + sx_d, \quad s \geq 0,
\]

(2.21)

where \( s \) is the arc length of the ray and \( x_d \) the direction vector

\[
x_d = \begin{pmatrix} - \sin t \\ \cos t \end{pmatrix}.
\]

(2.22)

The ray tracer then runs the following process:

for 'all rays in the ray set'  
create position and direction vector  
while 'target ray has not hit target'  
for 'all segments'  
calculate ray intersection  
end
choose the segment for which s is the smallest
if ‘ray hits the target’
store position and direction on target
else
calculate reflection or refraction
update position and direction vector
end
end

How exactly the intersections and reflections/refractions are calculated is different for lines and splines. The following two sections show how it is done.

2.3.1 Linear segments
A parametric representation of a linear segment between points \( x_1 \) and \( x_2 \) is given by

\[
\mathbf{x} = \mathbf{x}_1 + \xi (\mathbf{x}_2 - \mathbf{x}_1), \quad 0 \leq \xi \leq 1.\tag{2.23}
\]

We can find \( s \) and \( \xi \) for which the ray and the segment intersect by solving

\[
(\mathbf{x}_1 - \mathbf{x}_2) \begin{pmatrix} \xi \\ s \end{pmatrix} = \mathbf{x}_1 - \mathbf{x}_0.\tag{2.24}
\]

It has a solution if the vectors are not parallel, but solutions are only admissible if \( s > 0 \) and \( 0 \leq \xi \leq 1 \).

The normal to the segment is given by

\[
\mathbf{n} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} (\mathbf{x}_2 - \mathbf{x}_1).\tag{2.25}
\]

Call \( \mathbf{\hat{n}} \) the unit normal. Then the reflection of the ray in the segment causes a change in the direction vector given by

\[
\mathbf{x}_{d, \text{new}} = \mathbf{x}_d - 2(\mathbf{x}_d, \mathbf{\hat{n}})\mathbf{\hat{n}}.\tag{2.26}
\]

Note that although the normal in (2.25) can have two signs, both choices have the same result when calculating the reflection.

Refraction of a ray that propagates from a material with refractive index \( n_1 \) to a material with refractive index \( n_2 \) is given by Snell’s law. In practice it is most convenient to first rotate over the angle \( \alpha \), the angle the segment makes with the horizontal axis. Then the refraction is given by

\[
\mathbf{x}_{d, \text{new}} = \left( \beta \sqrt{1 - \beta^2} \right), \quad \beta = \frac{n_1}{n_2} x_{d,x,\alpha},\tag{2.27}
\]

in which \( x_{d,x,\alpha} \) is the \( x \)-direction of the rotated \( \mathbf{x}_d \) vector. When \( n_1 > n_2 \), internal reflection occurs when \( \beta > 1 \).
2.3.2 Splines

A curved segment is a spline \( x(z) \) constructed between points \((x_1, z_1), \ldots, (x_n, z_n)\). We interpolate the points with cubic splines; piecewise cubic polynomials in the intervals \([z_j, z_{j+1}], \ j = 1, \ldots, n-1\). The resulting function \( x \) is two times continuously differentiable. We follow the approach of [9]. The points can be interpolated with linear Lagrange polynomials

\[
x(z) = A(z)x_j + B(z)x_{j+1}, \quad A(z) = \frac{z_{j+1} - z}{z_{j+1} - z_j}, \quad B(z) = 1 - A(z),
\]

so that the resulting function is piecewise linear. Suppose that also the second derivatives \( x_j'' \) are known, we can add a function that linearly interpolates the second derivatives and evaluates to 0 in \( z_j \), i.e.,

\[
\hat{x}(z) = C(z)x_j'' + D(z)x_{j+1}''.
\]

The spline is then given by

\[
x(z) = A(z)x_j + B(z)x_{j+1} + C(z)x_j'' + D(z)x_{j+1}'', \quad z \in [z_j, z_{j+1}]
\]

(2.30)

where

\[
A(z) = \frac{z_{j+1} - z}{z_{j+1} - z_j}, \quad B(z) = 1 - A(z),
\]

\[
C(z) = \frac{1}{6}(A(z)^3 - A(z))(z_{j+1} - z_j)^2,
\]

\[
D(z) = \frac{1}{6}(B(z)^3 - B(z))(z_{j+1} - z_j)^2.
\]

Although the second derivatives are unknown in general, they can be determined with use of the fact that the first derivatives at the endpoints of the intervals should be equal. The first derivative is given by

\[
\frac{dx}{dz} = \frac{x_{j+1} - x_j}{z_{j+1} - z_j} - \frac{3A(z)^2 - 1}{6}(z_{j+1} - z_j)x_j'' + \frac{3B(z)^2 - 1}{6}(z_{j+1} - z_j)x_{j+1}'', \quad z \in [z_j, z_{j+1}]
\]

(2.32)

We demand for the first derivative that

\[
\frac{dx}{dz}(z_j^+) = \frac{dx}{dz}(z_j^-), \quad j = 2, \ldots, n-1.
\]

(2.33)

We use this with (2.32) to get

\[
\frac{z_j - z_{j-1}}{6}x_{j-1}'' + \frac{z_{j+1} - z_j}{3}x_j'' + \frac{z_{j+1} - z_j}{6}x_{j+1}'' = \frac{x_{j+1} - x_j}{z_{j+1} - z_j} - \frac{x_j - x_{j-1}}{z_j - z_{j-1}}, \quad j = 2, \ldots, n-1,
\]

(2.34)

in which the \( x_j'' \) are the unknowns. By setting \( x_n'' = 0 \) we have a system of \( n-2 \) equations in the \( n-2 \) unknowns (there are boundary conditions possible, but these seem natural). The spline (2.30) is then completely determined.

The above only states how to construct the spline, but does not give an expression to find the intersection between a ray and the spline. Instead, a method related to Newton’s method
is used. For an initial value $z$ we calculate the derivative $\frac{dx}{dz}$ with (2.32) and calculate the intersection between the tangent and the ray. This gives a new value $z$ for which we can do the same, until the desired accuracy is reached (in our simulations $10^{-12}$).

There is no need to carry out this algorithm if the ray and spline do not intersect. Therefore, we first check if the ray hits the bounding box of the spline, the smallest rectangle that completely covers the curve.

If the ray hits the bounding box, we still need an initial value $z$. We obtain this by taking a piecewise linear approximation of the spline. Once the point of intersection is found up to the required accuracy, the tangent can be calculated and reflection and refraction go the same as before.

### 2.4 Other fundamental information

#### 2.4.1 Constant luminance along a ray

The luminance, as defined in Section 2.2 is constant along a ray that propagates through a non-absorbing, uniform medium. Suppose a light ray emitted from a line segment $\text{da}_1$ hits another line segment $\text{da}_2$, located at a distance $r$, see Figure 2.6.

![Figure 2.6](image)

**Figure 2.6:** (a) Two line elements $\text{da}_1$ and $\text{da}_2$ with normals $\text{n}_1$ and $\text{n}_2$, respectively, that make an angle of $\theta_1$ and $\theta_2$ with the central ray with length $r$. (b) Viewing angles $d\phi_1$ and $d\phi_2$

The viewing angle $d\phi_1$ that $\text{da}_2$ defines on $\text{da}_1$ is

$$d\phi_1 = \frac{\cos \theta_2 \text{da}_2}{r}$$

(2.35)
and similarly

$$d\phi_2 = \frac{\cos \theta_1 da_1}{r}. \quad (2.36)$$

From this we obtain that

$$dU_1 := n \cos \theta_1 da_1 d\phi_1 = n \frac{\cos \theta_1 \cos \theta_2 da_1 da_2}{r} = n \cos \theta_2 da_2 d\phi_2 =: dU_2. \quad (2.37)$$

Since we consider here an elementary light beam, the étendue of the beam \(dU\) equals the étendue of the light emitted from \(da_1\) passing through \(da_2\): \(dU = dU_1\). On the other hand, all the light that passes through \(da_2\) is emitted from \(da_1\), and the étendue of the light beam \(dU = dU_2\). We conclude that the étendue \(dU\) is conserved and since also the flux through areas \(da_1\) and \(da_2\) is the same, we find that the luminance \(L\) is conserved along a ray in a homogeneous medium:

$$L = n \frac{d\Phi}{dU} = \text{constant}. \quad (2.38)$$

This also holds when the ray reflects or refracts. We will see in Chapter 3 that the luminance plays a central role in ray tracing. A small side note: a similar reasoning holds when light propagates between two media with different indices of refraction. In that case not the luminance but the basic luminance defined as \(L^* := L/n\) is constant. When we state in the rest of this report that luminance is constant, one should read basic luminance instead. However, in our examples both the source and the target are both located in media with refractive index \(n = 1\), so the luminance and basic luminance are equal.

### 2.4.2 Lambert’s cosine law

Lambert’s cosine law states that the luminous intensity of a perfectly diffusely emitting source is proportional to \(\cos t\), where \(t\) is again the angle between the normal of the surface and the direction of the emitted light, i.e.

$$I(\theta) = I_0 \cos t, \quad (2.39)$$

see Figure 2.7.

Definitions (2.17) and (2.20) together give that the luminous intensity emitted in direction \(t\) by the source \(S\) is given by

$$I = \int_S L(x, t) \cos t dx = \cos t \int_S L(x, t) dx. \quad (2.40)$$

In the specific case that \(L(x, t) = L_0\) is a constant, then the luminous intensity becomes

$$I = aL_0 \cos t = I_0 \cos t, \quad I_0 = aL_0, \quad (2.41)$$

so we recover (2.39). A source that has this property is called Lambertian, and the fact that they have a constant luminance makes them easy to use in calculations, as we will see in Section 3.2. Some LEDs have an intensity profile that is close to Lambertian, so in those cases it makes sense to use the Lambertian profile as a model.
Figure 2.8 shows a simple optical system with a ray set. On some parts of the target, rays are more concentrated than on other parts. The illuminance on the target screen can be calculated by dividing the line into bins and adding the flux of rays that fall into each bin and in a similar way the intensity can be calculated.

At this point we can see two downsides of this method:

• Although we know exactly the position and angle of a ray that hits the target, when calculating the illuminance or intensity some of this information is lost, because the flux of a ray is averaged over a bin.

• Some parts of the target screen receive no rays or only a few, but one can only confidently say that no flux falls on those parts if the number of rays is extremely high. Hence a large number of rays is traced to parts where e.g. the illuminance can be determined easily, while only a few go there where the illuminance is low and hard to calculate.

The big advantage of the method is that it is robust and requires no difficult calculations. In the next chapters we will try to view things in a different light and provide a new approach to perform fast ray tracing.
Figure 2.8: Simple optical system with a ray set.
Chapter 3

Phase space

This Section explains how the concepts introduced in Section 2 can be put to good use, so that the light properties of the source can be related to the desired properties of the target. First we describe the mathematical environment in which this process is carried out, followed by an exact approach to solve the problem for a few simplified cases.

3.1 Mathematical formulation

Throughout this report only 2D optical systems are considered. A light source in 2D is a point source (not considered) or a line segment, say $S$, see Figure 3.1. Every ray emitted from the source can be represented by its position of emittance $x$ and the angle of emittance $t$. The optical phase space for the light source is defined as

$$P_s = S \times [-n_s, n_s],$$

so that every ray $(x, t)$ is related to a unique point $(x, \tau) \in P_s$, where $\tau = n_s \sin t$ (the transformation $\tau = n \sin t$ will be explained shortly), and $n_s$ the index of refraction of the medium in which the source located.

We define the (basic) luminance as

$$L_s : P_s \mapsto \mathbb{R}, \quad L_s(x, \tau).$$

From the definitions for intensity (2.17) and luminance (2.20) it follows that

$$I_s : [-n_s, n_s] \mapsto \mathbb{R}, \quad I_s(\tau) = n_s \cos \tau \int_S L_s(x, \tau)dx,$$

and similarly, the definition for illuminance (or emittance for a source) reads

$$E_s : S \mapsto \mathbb{R}, \quad E_s(x) = \int_{-n_s}^{n_s} L_s(x, \tau)d\tau.$$

Note that although these definitions contain $\tau$ and not $t$, eventually the results are given in terms of $t$, since this is preferred by optical designers.

The above gives a mathematical model for the properties of the source. The properties of the target are now defined in a very similar fashion. Let $T$ be the target, then the optical phase space for the target is defined as
Figure 3.1: A light ray travels from position $x$ on the source to $q$ on the target, as the angle changes from $t$ to $\theta$. If the ray is drawn as a vector with length $n_s$ or $n_t$, then $\tau$ and $\eta$ respectively, are the projections of the vector on the plane normal to the optical axis.

\[ \mathcal{P}_t = \mathcal{T} \times [-n_t, n_t], \]

so light rays that hit the target at point $q$ with angle $\theta$ correspond to a point $(q, \eta) \in \mathcal{P}_t$, with $\eta = n_t \sin \theta$. On the target, we define the luminance $L_t = L_t(q, \eta)$, intensity $I_t(\eta)$ and illuminance $E_t(q)$.

In Section 2.4.1 we have already seen that the étendue was conserved. We can rewrite this as

\[
dU = n_s dx \cos t \, dt = n_s dx \cos t \frac{dt}{d\tau} \, d\tau = dx d\tau.
\]

Since the left hand side is conserved, the right hand side must be conserved as well, so volume is conserved in phase space.

In between the source and the target lies the optical system. Rays emitted from the source reflect and refract along the way and eventually hit the target. The map that describes how the optical system changes the rays is defined as

\[ \mathcal{M} : \mathcal{P}_s \rightarrow \mathcal{P}_t, \quad (x, \tau) \mapsto (q, \eta). \]

We can give an expression for the luminance on the target with help of this mapping $\mathcal{M}$. The luminance is constant along a ray, therefore $L_t$ is simply
\[ L_t = L_s(\mathcal{M}^{-1}(q, \eta)). \] (3.8)

\( \mathcal{M} \) is injective since we assume that no Fresnel reflection occurs and therefore a light ray cannot split up into two parts, but it is not surjective, so we need to restrict the inverse \( \mathcal{M}^{-1} \) to the range of \( \mathcal{M} \). To give meaning to this expression for the luminance, the mapping \( \mathcal{M} \) should be determined. That will be the goal of the next section.

### 3.2 Exact approach

Although we defined \( \mathcal{M} \) to be a map from \( \mathcal{P}_s \) to \( \mathcal{P}_t \), for now it is easier to think of \( \mathcal{M} \) in terms of the original coordinates \((x, t)\) and \((q, \theta)\).

For general optical systems it is not straightforward to find an analytical expression for the map \( \mathcal{M} \). Therefore we start by considering one of the simplest systems, the two-faceted cup. After that some words are spent on the Compound Parabolic Concentrator.

#### 3.2.1 Two-faceted cup

The two-faceted cup in Figure 3.2 is one of the simplest optical systems one can image. It consists of a straight source with length \(2a\) and two straight reflecting surfaces on each side that make an angle of \(\gamma\) with respect to the normal of the source. The straight target lies at height \(h\) at the end of the cup.

For every individual ray it is not difficult to describe how it propagates through the system and where exactly it hits the target. One could, for example, calculate at what height \(h_1\) the ray hits one of the reflectors, then calculate how the ray reflects and continue until the ray intersects the target, see Figure 3.2. It is however not so easy to find a closed expression for \( \mathcal{M} \). The difficulty lies in the fact that the expression for \( \mathcal{M} \) is different on different parts of \( \mathcal{P}_s \): some rays do not hit the reflectors, others are reflected once or multiple times. We define \( M_k \) as

\[ M_k := \{(x, t) | \text{corresponding ray reflects } k \text{ times}\}. \] (3.9)

\(k\) positive when the first reflection is on the left reflector, \(k\) negative when the first reflection is on the right reflector. We use \( M_k \) as a shorthand notation for the image of \( M_k \) under \( \mathcal{M} \). The next part is devoted to determining the maximum \( r \) such that \( M_r \neq \emptyset \) and determining the sets \( M_k, k \leq r \).

There is an elegant method to determine the maximum number of times that a ray can reflect. Figure 3.3 shows that mirroring a ray in the reflector has the same effect as mirroring the entire cup in its own reflector and letting the ray propagate in a straight line. The points \( B_k \) are obtained by rotating the point \((0, B)\) around the point \((0, -\frac{a}{\tan \gamma})\). A straightforward calculation shows that

\[ B = \frac{h}{\cos \gamma} + \frac{a}{\sin \gamma} - \frac{a}{\tan \gamma}, \] (3.10)

so the exact expression for \( B_k \) is
Figure 3.2: An approach to find \((q, \theta)\) for a ray \((x, t)\) is to calculate successively the heights \(h_i\) where the ray is reflected.

\[
B_k = \begin{pmatrix} B_{kx} \\ B_{kz} \end{pmatrix} = \begin{pmatrix} \cos((2k+1)\gamma) & -\sin((2k+1)\gamma) \\ \sin((2k+1)\gamma) & \cos((2k+1)\gamma) \end{pmatrix} \begin{pmatrix} 0 \\ B + \frac{a}{\tan\gamma} \end{pmatrix} - \begin{pmatrix} 0 \\ \frac{a}{\tan\gamma} \end{pmatrix}. \tag{3.11}
\]

The maximum number of reflections is then simply

\[
r = \max \{ k \in \mathbb{N} | B_{k-1,z} \geq 0 \}, \tag{3.12}
\]

obtained by rays that leave the source at a large angle.

The method of reflecting the cup instead of the ray also gives insight in how to find the regions \(M_k\). The set of rays that form the boundary of these regions can only consist of rays that either leave the source at its boundary, or hit one of the \(B_k\)'s. Only then, a small change in position or angle can cause a difference in the number of reflections. Figure 3.4 illustrates this for \(M_0\). The rays that hit \(B_k\), as in Figures 3.4a for \(B_0\) and 3.4b for \(B_{-1}\), are given by the relation

\[
\tan t = \frac{x - B_{kx}}{B_{kz}}. \tag{3.13}
\]

The rays that are emitted from the boundary of the source, as in Figures 3.4c and 3.4d, form vertical lines in \(P_s\).

It takes some good bookkeeping to calculate for a ray where it exits the cup, but also here the mirroring principle offers a nice solution. For a ray \((x, t) \in M_k\), the point of intersection \((u, v)\) with the line segment \(B_{k-1}B_k\), the \(k\) times mirrored target, is calculated, see Figure 3.3. The point of intersection with the actual target can be found by rotating this point back as much as we rotated the cup forward, and possibly a sign change, depending on \(k\) being even or odd:

\[
\begin{pmatrix} q \\ h \end{pmatrix} = \begin{pmatrix} (-1)^k & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \cos(-2k\gamma) & -\sin(-2k\gamma) \\ \sin(-2k\gamma) & \cos(-2k\gamma) \end{pmatrix} \begin{pmatrix} u + \frac{a}{\tan\gamma} \\ v \end{pmatrix} - \begin{pmatrix} 0 \\ \frac{a}{\tan\gamma} \end{pmatrix}. \tag{3.14}
\]

The angle of exitance is also simply an addition of an angle and possibly a sign change:
\[ \theta = (-1)^k (t - 2k\gamma). \] (3.15)

For every \( k \), the mapping \( (x, t) \mapsto (q, \theta) \) is now completely determined. For the two-faceted cup, the regions \( M_k \) and corresponding regions \( \mathcal{M}_k \) are shown in Figure 3.5.

### 3.2.2 CPC

We leave the two-faceted cup for what it is and consider a compound parabolic concentrator or CPC [2]. It consists of two parabolically shaped reflectors, one on each side of the source, see Figure 3.6. AC is part of a parabola with axis parallel to AD and focus B, BD is part of a parabola with axis parallel to BC and focus A. A convenient way of finding a description for these reflectors is to find the right expression for a parabola with focus \((0, 0)\) and symmetric in the vertical axis, then tilt it by an angle \( \Gamma \) and apply a translation. We do this for the left reflector with \( A = (-a, 0) \) and \( B = (a, 0) \). Let \( B \) be the origin in a tilted coordinate system with the horizontal \( u \)-axis perpendicular to \( AD \) and the vertical \( v \)-axis parallel to \( AD \). The
Figure 3.5: (a) $P_s$ and (b) $P_t$ for the two-faceted cup, with $a = 5$, $b = 20$, $h = 40$ and $S = [-5, 5]$. Regions $M_k$ of rays that reflect $k$ times are mapped to $M_k$ in the target phase space.
point $A$ can be expressed in the new coordinate system by translating and rotating as follows:

$$
\tilde{A} = \left( \tilde{A}_u \quad \tilde{A}_v \right) = \begin{pmatrix} \cos \Gamma & -\sin \Gamma \\ \sin \Gamma & \cos \Gamma \end{pmatrix} (A - B). \quad (3.16)
$$

The parabola is then described by

$$
v = p(u) = \frac{\tilde{A}_u + \sqrt{\tilde{A}_u^2 + \tilde{A}_v^2}}{2A_u} u^2 - \frac{8\tilde{A}_u}{\tilde{A}_v + \sqrt{\tilde{A}_u^2 + \tilde{A}_v^2}}. \quad (3.17)
$$

It is convenient to use vector notation to apply a rotation and translation:

$$
\begin{pmatrix} x \\ z \end{pmatrix} = \begin{pmatrix} \cos(-\Gamma) & -\sin(-\Gamma) \\ \sin(-\Gamma) & \cos(-\Gamma) \end{pmatrix} \begin{pmatrix} u \\ p(u) \end{pmatrix} + B \quad (3.18)
$$

This gives us the desired reflector.

![Diagram of CPC with source AB and acceptance angle $\Gamma$.](image)

Figure 3.6: Example of a CPC with source AB and acceptance angle $\Gamma$.

A special property of the CPC is that all rays leave the exit aperture $CD$ with an angle $\theta$ such that $-\Gamma \leq \theta \leq \Gamma$ and all rays that have such an angle can be traced back to the source. When $L_s(x,t) = L_0$, the domain in $P_t$ where $L_t(q,\theta) > 0$ is then simply the area $CD \times [-\Gamma, \Gamma]$ and the intensity is given by

$$
I_t(\theta) = \begin{cases} 
\int_{CD} L_0 \cos \theta dq = L_0 |CD| \cos \theta & |\theta| \leq \Gamma \\
0 & |\theta| > \Gamma
\end{cases} \quad (3.19)
$$
Another interesting case to consider is when the luminance is not constant, but an arbitrary function of \(x\) and \(t\). Recall that expression (3.8) gives the relation between \(L_t\) and \(L_s\). Up to this point, we only considered the case where \(L_s(x,t) = \text{constant}\), which made it unnecessary to use this relation. Now that we allow \(L_s\) to depend on \(x\) and \(t\), the relation is useful and stresses the importance of finding \(M^{-1}\). For the two-faceted cup we can still obtain exact results. Mapping the angle \(\theta\) back to the original angle \(t\) is done easily by inverting expression (3.15):

\[
t = (-1)^k \theta + 2k\gamma.
\]  

(3.20)

How to find \(x\) for a given \(q\) can be derived with the same logic as with which we derived (3.14): we rotate the point \((q,h)\) back and then calculate where the ray should have been emitted. Suppose the ray hits the target in \((q,h)\) with angle \(\theta\), such that \((q,\theta) \in M_k\). The point \((u,v)\) as in Figure 3.3 can be calculated with

\[
\begin{pmatrix}
u \\
v
\end{pmatrix} = \begin{pmatrix}
cos(2k\gamma) & -\sin(2k\gamma) \\
\sin(2k\gamma) & \cos(2k\gamma)
\end{pmatrix} \begin{pmatrix}
(-1)^k q \\
h + \frac{a}{\tan\gamma}
\end{pmatrix} - \begin{pmatrix}
0 \\
\frac{a}{\tan\gamma}
\end{pmatrix}
\]  

(3.21)

The position \(x\) on \(S\) where the ray is originated is then given by

\[
x = u + \tan((-1)^k \theta + 2k\gamma)v.
\]  

(3.22)

With this closed expression for \(M^{-1}\) we can use (3.8) to the fullest and later it will turn out to be very useful to verify numerical results.
Chapter 4

Hamiltonian optics

The treatment of phase space in the previous chapter points strongly in the direction of a Hamiltonian formulation of light propagation, known as Hamiltonian optics. This chapter has an exploratory nature and aims to form a connection between Fermat’s principle, phase space, a Lagrangian formulation and a Hamiltonian formulation. In Section 4.4 a symplectic numerical scheme in considered to integrate a Hamiltonian system. Results are given in Section 4.5.

4.1 Fermat’s principle and the Lagrangian

A light ray propagating through a medium can be represented by a vector $\vec{q} \in \mathbb{R}^3$ in the Cartesian coordinate system, parameterized by the arc length $s \in \mathbb{R}$ of the ray:

$$\vec{q}(s) = \begin{pmatrix} q_1(s) \\ q_2(s) \\ q_3(s) \end{pmatrix}. \tag{4.1}$$

Rays are continuous and piecewise differentiable [16]. Suppose that the refractive index $n$ depends on the position of the ray in the material, i.e., $n = n(q_1, q_2, q_3)$. Recall from Chapter 2 that the optical path length between two points $A$ and $B$ along the curve $C$ is then given by

$$S = \int_C n(\vec{q}(s)) ds, \tag{4.2}$$

where $ds = \sqrt{dq_1^2 + dq_2^2 + dq_3^2}$ is the infinitesimal curve length. Fermat’s principle then states that the optical path length of any ray between two points is an extremum, so

$$\delta S = \delta \int_A^B n ds = 0. \tag{4.3}$$

The trajectory of the ray can be represented by $\vec{q}(\sigma) = (q_1(\sigma), q_2(\sigma), q_3(\sigma))$ with $\sigma$ an arbitrary parameter. If $A = \vec{q}(\sigma_1)$ and $B = \vec{q}(\sigma_2)$, then $S$ becomes

$$S = \int_{\sigma_1}^{\sigma_2} n(q_1, q_2, q_3) \sqrt{\left(\frac{dq_1}{d\sigma}\right)^2 + \left(\frac{dq_2}{d\sigma}\right)^2 + \left(\frac{dq_3}{d\sigma}\right)^2} d\sigma. \tag{4.4}$$
Let $q'_i = dq_1/d\sigma$, we can now define the Lagrangian $\mathcal{L}$ as

$$\mathcal{L} := n(q_1, q_2, q_3) \sqrt{\left(\frac{dq_1}{d\sigma}\right)^2 + \left(\frac{dq_2}{d\sigma}\right)^2 + \left(\frac{dq_3}{d\sigma}\right)^2} = \mathcal{L}(q_1, q_2, q_3, q'_1, q'_2, q'_3).$$ \hspace{1cm} (4.5)

Together with (4.3) and (4.4) this gives a Lagrangian system for which the corresponding Euler-Lagrange equations read [10]:

$$\frac{d}{d\sigma} \left( \frac{\partial \mathcal{L}}{\partial q'_i} \right) = \frac{\partial \mathcal{L}}{\partial q_i}, \quad i = 1, 2, 3.$$ \hspace{1cm} (4.6)

The term $\frac{\partial \mathcal{L}}{\partial q'_i}$ is in this context referred to as the momentum $p_i$ of the ray:

$$p_i := \frac{\partial \mathcal{L}}{\partial q'_i} = n \frac{q'_i}{\sqrt{(q'_1)^2 + (q'_2)^2 + (q'_3)^2}} = n \frac{dq_i}{ds}, \quad i = 1, 2, 3.$$ \hspace{1cm} (4.7)

Hence the momentum vector

$$\vec{p}(s) = \begin{pmatrix} p_1(s) \\ p_2(s) \\ p_3(s) \end{pmatrix}$$ \hspace{1cm} (4.8)

is parallel to the tangent $\frac{dq(s)}{ds}$ of the ray. It follows using (4.6) that

$$\frac{\partial S}{\partial q_i} = \frac{\partial}{\partial q_i} \int_{\sigma_1}^{\sigma_2} \mathcal{L} d\sigma = \int_{\sigma_1}^{\sigma_2} \frac{\partial \mathcal{L}}{\partial q_i} d\sigma = \int_{\sigma_1}^{\sigma_2} \frac{dp_i}{d\sigma} d\sigma = p_i, \quad i = 1, 2, 3.$$ \hspace{1cm} (4.9)

The last equality in Equation (4.7) together with Equation (4.9) combine to:

$$\vec{p} = n \frac{d\vec{q}}{ds} = \nabla S.$$ \hspace{1cm} (4.10)

We use this relation to find how the momentum changes with the arc length:

$$\frac{d}{ds} \left( n \frac{dq_i}{ds} \right) = \frac{d}{ds} \frac{\partial S}{\partial q_i} = \frac{d}{ds} \frac{\partial}{\partial q_i} \int_C n ds = \frac{d}{ds} \int_C \frac{\partial n}{\partial q_i} ds = \frac{\partial n}{\partial q_i}, \quad i = 1, 2, 3,$$ \hspace{1cm} (4.11)

or in vector notation:

$$\frac{d}{ds} \left( n \frac{d\vec{q}}{ds} \right) = \frac{\partial n}{\partial \vec{q}}.$$ \hspace{1cm} (4.12)

We conclude that

$$\frac{d\vec{p}}{ds} = \frac{\partial n}{\partial \vec{q}}.$$ \hspace{1cm} (4.13)
4.2 Derivation of the Hamiltonian

In Equation (4.6) we have the Euler-Lagrange equations that describes how light travels through the medium. Recall that in Chapter 2 we actually already used this in a 2D case with \( \sigma = z \), the position along the optical axis. We now wish to continue from the Lagrangian formulation to derive a Hamiltonian formulation for the problem.

The Lagrangian \( L \) is a function of \( \bar{q} \) and \( \dot{\bar{q}} \), both in the real space. Therefore solving the Lagrangian formulation gives the ray trajectory in the real space. The Hamiltonian function \( H \) is a function defined on phase space. \( H \) is called a Hamiltonian function if

\[
H = H(q_1, q_2, q_3, p_1, p_2, p_3)\text{ is such that}
\]

\[
\frac{dq_i}{ds} = \frac{\partial H}{\partial p_i}, \quad \frac{dp_i}{ds} = -\frac{\partial H}{\partial q_i}, \quad i = 1, 2, 3. \tag{4.14}
\]

We take a closer look at (4.7) to find

\[
p_1^2 + p_2^2 + p_3^2 = n^2, \tag{4.15}
\]

and therefore

\[
|\bar{p}| = n. \tag{4.16}
\]

From the first set of equations in (4.14) and (4.10) we get

\[
\frac{\partial H}{\partial p_i} = \frac{dq_i}{ds} = \frac{p_i}{n} = \frac{p_i}{|\bar{p}|}. \tag{4.17}
\]

Integration immediately gives that

\[
H = |\bar{p}| + c_1(\bar{q}), \tag{4.18}
\]

with \( c_1 \) an arbitrary function depending only on \( \bar{q} \). The second set of equations in (4.14) together with (4.13) gives

\[
\frac{\partial H}{\partial q_i} = -\frac{dp_i}{ds} = -\frac{\partial n}{\partial q_i}. \tag{4.19}
\]

Again we can integrate to obtain

\[
H = -n + c_2(\bar{p}). \tag{4.20}
\]

We combine these two results two find that

\[
H = |p(\bar{s})| - n(\bar{q}(\bar{s})) + c. \tag{4.21}
\]

We have the freedom to choose the constant \( c \), so we choose \( c = 0 \) such that, combined with (4.16) we find \( H = 0 \). It is constant along any ray, since it is does not depend (explicitly) on the arc length \( s \).
4.3 Hamiltonian on the screen

In the previous section, we derived a Hamiltonian defined on a six-dimensional space, but we can reduce this by using again (4.16). The momentum vector is restricted to a sphere with radius \( n \), so we can eliminate \( p_3 \) by writing

\[
\vec{p} = \sqrt{n} (\vec{q}^2 - (p_1^2 + p_2^2)).
\]

We can further reduce the number of dimensions by parameterizing the rays with \( q_3 = z \), the position on the optical axis, instead of the arc length \( s \). Then the manifold of rays (oriented lines) in \( \mathbb{R}^3 \) has 4 dimensions and is called optical phase space \( P \). We write

\[
\vec{q} = \begin{pmatrix} q_1 \\ q_2 \\ z \end{pmatrix}, \quad \vec{q} := \begin{pmatrix} q_1 \\ q_2 \end{pmatrix}, \quad \vec{p} = \begin{pmatrix} p_3 \\ p_1 \end{pmatrix}, \quad p := \begin{pmatrix} p_1 \\ p_2 \end{pmatrix}.
\]

(4.22)

The 2-dimensional submanifold \( z = 0 \) is called the standard screen with optical center at \( \vec{q} = 0 \) and optical axis \( q_3 (p = 0) \). Now \( p_3 = \sqrt{n(q, z)^2 - |p|} \). We define the optical phase space \( P \) as a manifold with coordinates \((p, \sigma, q)\), where \( q \subset \mathbb{R}^D \) denotes the position of the intersection of the ray with the screen at \( z = 0 \), \( p \in D_n^D \in \mathbb{R}^D \) is the momentum projected on the screen, restricted to the disk \( D_n^D \) with radius \( n \) and \( \sigma \in \{-1, 0, 1\} \) distinguishes backward rays \((\sigma = -1)\), forward rays \((\sigma = 1)\) and rays parallel to the screen \( \sigma = 0 \). For now we only consider forward rays.

We can transform the Hamilton equations (4.14) related to the arc length \( s \) into Hamilton equations on a screen at \( q_3 = z \). Figure 4.1 illustrates how the transformation works: since the momentum vector has the same direction as the tangent to the ray, the ratio of \( dz \) and \( dq \) has to equal the ratio of \( p_3 \) and \( p \).

\[
\begin{align*}
\frac{dz}{ds} & = \frac{dq}{ds} \quad \frac{dp}{dz} = \frac{d\sigma}{ds} \quad \frac{dh}{dp} = \frac{\partial h}{\partial q}.
\end{align*}
\]

(4.23)

The screen Hamiltonian function \( h \) is then given by

\[
h(p, q, z) = -p_3.
\]

(4.24)

We can define a positive (energy) density of rays \( \rho(p, q) \geq 0 \) on phase space, corresponding to the luminance. A single ray corresponds to the 2D-dimensional Dirac delta function on phase space. The total energy flux of the beam is given by
\[ \Phi = \int_{\mathcal{P}} d\rho(p, q). \] (4.25)

Similarly, the screen illuminance and the screen intensity can be defined.

Define the Poisson bracket of two functions \( f(p, q) \) and \( g(p, q) \) on phase space as

\[ \{ f, g \}(p, q) := \frac{\partial f(p, q)}{\partial q} \frac{\partial g(p, q)}{\partial p} - \frac{\partial f(p, q)}{\partial p} \frac{\partial g(p, q)}{\partial q}. \] (4.26)

A map of phase space \( \mathcal{M} : (p, q) \mapsto (P(p, q), Q(p, q)) \) that preserves the Hamiltonian structure is a canonical transformation, which is the case when

\[ \{ P_i, P_j \} = 0, \quad \{ Q_i, Q_j \} = 0, \quad \{ Q_i, P_j \} = \delta_{ij}. \] (4.27)

The evolution of \( \rho \) in \( z \) is governed by the Hamilton equations and can be described by

\[ \frac{d\rho(p, q)}{dz} = \frac{\partial \rho}{\partial p} \frac{dp}{dz} + \frac{\partial \rho}{\partial q} \frac{dq}{dz} = -\frac{\partial h(p, q)}{\partial q} \frac{\partial \rho(p, q)}{\partial p} + \frac{\partial h(p, q)}{\partial p} \frac{\partial \rho(p, q)}{\partial q} = -\{ h(z), \rho \}(p, q). \] (4.28)

### 4.4 Symplectic integration of the Hamiltonian

As we have seen, the propagation of every light ray is determined by the Hamiltonian structure. This section tries to answer the question of how to find the ray path of a ray with certain initial position and initial momentum, given the refractive index in every point in the material.

First we briefly recap what we have seen so far: we have phase space \( \mathcal{P} \) in \( \mathbb{R}^4 \), a function \( H \in C^1(\Omega) \) and the relations

\[ \dot{q}_i = \frac{\partial H}{\partial p_i}, \quad \dot{p}_i = -\frac{\partial H}{\partial q_i}, \quad i = 1, 2. \] (4.29)

The refractive index of the material may depend on the position vector and we assume that it is known. A special property of the Hamiltonian structure is that when the system evolves, a volume in phase space is conserved. A numerical integration method that preserves this property is called a symplectic integrator, which we will discuss next.

We will take a step back and then slowly work up to a point where we have found such a symplectic integrator. Suppose we have a differential equation of the form

\[ y' = f(t, y), \quad y(0) = y_0, \quad t \geq 0. \] (4.30)

An \( s \)-stage Runge-Kutta method gives the solution of the differential equation \( y_{n+1} \) at time \( t_{n+1} \), given the solution \( y_n \) at time \( t_n \). It makes use of a Butcher tableau [6] with constants \( a_{ij}, b_j, c_i \) with \( i, j = 1, \ldots, s \), arranged as in (4.31).

\[
\begin{array}{c|ccc}
c_1 & a_{11} & \ldots & a_{1s} \\
\vdots & \vdots & \ddots & \vdots \\
c_s & a_{s1} & \ldots & a_{ss} \\
\hline & b_1 & \ldots & b_s
\end{array}
\] (4.31)
The constants are such that
\[ c_j \geq 0, \quad \sum_{j=1}^s c_j = 1, \quad \sum_{j=1}^s a_{ij} = c_i, \quad \sum_{j=1}^n b_j = 1. \] (4.32)

The solution is calculated as follows:
\[ y_{n+1} = y_n + h \sum_{j=1}^s b_j f(t_n + c_j h, \xi_j), \] (4.33)
\[ \xi_i = y_n + h \sum_{j=1}^s a_{ij} f(t_n + c_i h, \xi_j), \] with \( h \) the time step, so \( t_n = nh \). The last equation of (4.33) is an implicit equation for the \( \xi_i \)'s that can be solved with Newton’s method.

In general, the Runge-Kutta methods are not symplectic integrators, but they can be, depending on the choice of the tableau. Define constants \( m_{ij} \) as
\[ m_{ij} := b_i a_{ij} + b_j a_{ji} - b_i b_j, \] (4.34)
then the Runge-Kutta method is symplectic if the matrix \( M = (m_{ij})_{i,j=1}^s = 0 \) [11]. This is the case for the fourth order Gauss-Legendre Runge-Kutta method with tableau (4.35).

\[
\begin{array}{ccc}
\frac{1}{2} - \frac{\sqrt{3}}{6} & \frac{1}{4} & \frac{1}{4} - \frac{\sqrt{3}}{6} \\
\frac{1}{2} + \frac{\sqrt{3}}{6} & \frac{1}{4} + \frac{\sqrt{3}}{6} & \frac{1}{4} \\
\frac{1}{2} & \frac{1}{2} & \frac{1}{2}
\end{array}
\] (4.35)

We rewrite the system (4.29) in the form of (4.30):
\[
\begin{pmatrix}
\dot{q} \\
\dot{p}
\end{pmatrix} = \begin{pmatrix}
\frac{\partial H}{\partial p} \\
-\frac{\partial H}{\partial q}
\end{pmatrix}.
\] (4.36)

The right hand side does not explicitly depend on the arc length parameter \( s \). Now given initial conditions \((q_0, p_0)\), we are able to numerically compute the ray path through the medium. The next section gives some results for this numerical integration method.

### 4.5 Numerical results for the symplectic integration of a light ray

In this section we show some examples of the propagation of light rays through a material with (possibly) variable index of refraction. We use the numerical scheme of the previous section to solve the Hamiltonian system.

We start with a trivial example with \( n = 1 \), so that it is possible to solve the Hamilton equations (4.14) by hand. The solution is given by
\[ \bar{p} = \bar{p}(0) \]
\[ \bar{q} = \bar{q}(0) + s \frac{\bar{p}}{|ar{p}|}, \] (4.37)
so there is no change in momentum and the ray is a straight line. If we specify the initial conditions in two dimensions

\[ \bar{q}(0) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \bar{p}(0) = \begin{pmatrix} \frac{1}{2} \sqrt{2} \\ \frac{1}{2} \sqrt{2} \end{pmatrix}, \] (4.38)

the system can be integrated numerically. The result is shown in Figure 4.2.

![Figure 4.2: Numerical integration of a ray in a medium with constant index of refraction results in a straight line.](image)

Another choice for \( n \) could be

\[ n(\bar{q}) = 1 + \kappa q_2, \] (4.39)

with \( \kappa > 0 \) a constant. Figure 4.3 gives the trajectory of the ray and the change in momentum as the ray propagates, for \( \kappa = 1/10 \) and the same initial conditions as in the previous example. As expected, the ray bends towards the higher refractive index material. Since \( n \) depends on \( q_2 \) only, \( p_1 \) remains unchanged, while \( p_2 \) increases.

In the last example we consider a Hamiltonian system with the screen Hamiltonian \( h \). An elliptic index-profile guide is a medium with refractive index

\[ n(q) = \sqrt{n_0^2 - \kappa^2 |q|^2}, \] (4.40)

with \( n_0 \) and \( \kappa \) constant [16]. Ray trajectories through such a medium spiral around the \( z \)-axis, see Figure 4.4.
Figure 4.3: Numerical integration of a ray in a medium with $n = 1 + q_2/10$. 
Figure 4.4: Result of numerical integration of the screen Hamiltonian for a ray in an index profile as in (4.40).
Chapter 5

Numerical phase space ray tracing

In Chapter 3 we have seen a mathematical formulation of ray tracing and Section 3.2 gives an exact approach to relate the input luminance $L_s$ to the luminance $L_t$ on the target. For more complicated optical systems this method becomes too involved or requires too much insight. The numerical approach explained in this chapter offers a solution. It can be subdivided into a number of steps and we start in the next section by giving an overview of these different steps, followed by a detailed explanation in the subsequent sections.

5.1 From source to target in giant steps

In Figure 5.1 a schematic overview is shown of the entire phase space ray tracing process as explained in this chapter. The inputs of the process are the luminance function on the source $L_s$ and the optical system, the output is the intensity or illuminance function on the target. An important step in between is to find the luminance $L_t$ on the target. The numerical approach is therefore constructed as follows:

- Ray tracing is used to find the luminance on scattered points in the target phase space $\mathcal{P}_t$. Instead of randomly choosing a large ray set (as done in Monte Carlo ray tracing) we construct an iterative ray tracing process in which only the first ray set is (partly) randomly chosen. After tracing this initial set of rays through the system, a new ray set is assigned that traces rays close to the boundaries of the regions $M_k$ as described in Section 3.2. This is repeated a number of times until a sufficient number of rays is traced. Further details are explained in Section 5.2.

- Not all parts of the target phase space have positive luminance. Some parts are not illuminated by the source and therefore have zero luminance. To distinguish parts of phase space with positive luminance from those with zero luminance, we use $\alpha$-shapes, see Section 5.2.3.

- The luminance is known for points scattered on $\mathcal{P}_t$, but if we want to know the luminance for any point in $\mathcal{P}_t$, the data has to be interpolated, as described in Section 5.3. There are many options to carry out the interpolation, but most methods require too much computational time. In this Section an algorithm is presented that outputs a smooth function that approximates the data nicely, while keeping the computational effort fairly small.
Finally, the $\alpha$-shapes and interpolation can be used to integrate the luminance over its position or angle to obtain the intensity or illuminance on the target. In Section 5.4 it is shown how to do the integration analytically.

5.2 Finding the boundaries of the $M$-regions

As explained before, ray tracing relates points $(x, t)$ from the source phase space to points $(q, \theta)$ in the target phase space and both points have the same luminance:

$$L_s(x, t) = L_t(q, \theta). \quad (5.1)$$

Although $L_s$ is usually a well-behaved function, due to reflections and refractions in the optical system, the function $L_t$ can have discontinuities. All rays that leave the source have a positive luminance, so $L_s$ is positive on the entire source phase space $P_s$, but this is certainly not the case for $P_t$: not all points in the target phase space correspond to rays that are emitted from the source. In many cases, the part of $P_t$ where $L_t > 0$ is small compared to the total volume of $P_t$.

In Section 3.2 we defined $M_k$ to be a region in $P_s$ that contains all rays that reflect the same number of times. When considering more complicated optical systems than the two-faceted cup, the distinction in the number of reflections is not general enough. Instead, we group together rays that follow the same path $\Pi$ when propagating from source to target, where we define the ray path as the collection of optical segments that the ray has encountered and call this domain $M_\Pi$. Previously, all rays in one domain $M_k$ were mapped to the target phase space by the same mapping for which we had a closed expression. In the general case we do not have a closed expression to calculate a ray’s path, but at least all rays in the same $M_\Pi P_i$ can be calculated following the same prescription and moreover, they form a connected region in $P_t$. As before, $M_\Pi$ is used as shorthand notation for the image of $M_\Pi$ under $M$.

Not only is $M_\Pi$ connected, but also the boundary of $M_\Pi$ is mapped to the boundary of $M_\Pi$. If $\Pi_1$ and $\Pi_2$ denote two different ray paths, then $M_{\Pi_1}$ and $M_{\Pi_2}$ certainly do not overlap. The regions can have a common boundary, but often they are disconnected. Therefore the target luminance $L_t$ will have a discontinuity at the boundary of the $M_\Pi$’s, whereas it is a well-behaved function in the interior (at least as $L_s$ is). If we wish to integrate $L_t$ accurately, we need to know where these discontinuities occur. In the following section, an algorithm is explained that is designed to do this.

5.2.1 Grid refining algorithm

Before any ray is traced, the mapping $M$ is completely unknown. An algorithm is desired that, after an initial ray set is traced, selects a new ray set that gives more information about the boundaries of the $M_\Pi$’s.

The initial ray set can either be randomly chosen, but preferably we make a smart choice. Basically, there are two ways to do this: we either select a point set in $P_s$ and hence control the rays that leave the source, or we select a point set in $P_t$ and control the rays that hit the target. In the last case we need to trace every ray $(q, \theta) \in P_t$ back to a ray $(x, t) \in P_s$. Since we want to find the boundaries of regions $M_\Pi$, this seems to be a logical choice, but there is a big disadvantage of back tracing. As discussed before, the target phase space can be very
Figure 5.1: Schematic overview of the steps in the ray trace process from top to bottom. Inputs of the process are the optical system (a) and for example an input intensity (b). Then rays are traced from source (c) to target (d). Then the idea is to find the boundary of the resulting point cloud (e) and apply interpolation to create the surface of the luminance (f). Finally, integration of this surface gives the output intensity (g).
thin. Choosing a ray set in the target phase space inevitably leads to back tracing rays that do not hit the source. Also, (at a later stadium) it would be difficult to incorporate Fresnel reflections or diffuse reflections when ray tracing.

Luckily, we have the property that $M$ maps $\partial M_\Pi$ to $\partial M_\Pi$ [2], so we might as well choose an initial ray set in $\mathcal{P}_s$ and try to find $\partial M_\Pi$. Therefore, $\mathcal{P}_s$ is covered with a grid and every grid point $(x_i, \tau_i)$ is traced. Along with the luminance of the ray, also the path $\Pi(x_i, \tau_i)$ is stored.

The next ray set can be selected by refining the grid at places where a boundary between two $M$-regions can be expected: we halve the grid size and copy the paths for the overlapping grid points. For the new grid points we have to decide if we trace the corresponding ray or not. This is done on the basis of the criterium explained in Figure 5.2. A grid cell is shown for which, after ray tracing, it turns out that three out of four rays have followed path $\Pi_1$ and one ray has followed path $\Pi_2$. This means that a boundary crosses the cell and one could decide to refine the grid as shown in the picture, to trace rays that are closer to the boundary.

When all corners of the grid cell have the same $\Pi$, it is not necessary to refine. This method will eventually come closer and closer to the boundaries of the different $M$-regions, except when such a region is smaller than the initial grid size and lies completely inside a grid cell, see Figure 5.3.

The refining step can be repeated as many times as desired. However, since the grid size is halved with every iteration, memory costs increase rapidly unless a smart structure is used to remember which parts are refined and which are not. This can be a bit of a hassle. Therefore in the next section, an algorithm is given that uses a triangulation instead of a
triangulation. The new structure allows us to lower the memory costs and increases the ease of implementation.

5.2.2 Triangulation

Instead of making use of the grid structure explained above, dividing $P_s$ into triangles could give us the freedom to choose the initial ray set (partly) at random. First, some background information about triangulations is needed.

Let $P := \{p_1, p_2, \ldots, p_n\} \subset \mathbb{R}^2$ be a set of points. A triangulation of point set $P$ is a set of edges $E$ with vertices the points in $P$ such that all faces (triangles) are bounded by three edges and no edge can be added to $E$ without crossing one of the others [3]. With this definition, a triangulation is not unique. A triangulation that is often used and has natural properties is called Delaunay triangulation. It is closely related to a Voronoi diagram. The Voronoi diagram of $P$ is a subdivision of $\mathbb{R}^2$ such that every region around a point $p \in P$ contains all points that are closer to $p$ than to every other point in $P$. The Delaunay triangulation is the dual graph of the Voronoi diagram: it has $P$ as its vertices and it has an edge between two vertices if the two corresponding faces share an edge, see Figure 5.4.

The Delaunay triangulation tends to avoid small angles and long edges. It has the following two properties [9](here stated without proof):

- Of all possible triangulations of a point set $P$, the Delaunay triangulation has the largest minimum angle.
- Three vertices form a triangle in the Delaunay triangulation if and only if the circle through these three vertices contains no other point of $P$.

It can be calculated in $\mathcal{O}(|P| \log |P|)$ time, with a randomized incremental approach explained in [3].

More convincing why the Delaunay triangulation is a better choice than a random triangulation is perhaps Figure 5.5. The triangulation is much more regular and contains a lot less thin triangles.
Now that we no longer make use of the grid structure, we should make a new choice for the initial ray set. We split this into two parts: we purposely trace a large number $N_e$ ($N_{edges}$) of edge rays (rays that leave the edges of the source), because they will form a boundary in $P_t$ as well. In the interior of $P_s$ we select a random set of rays of size $N_i$ ($N_{interior}$). We apply the Delaunay triangulation to this set, and then a similar refining algorithm can be applied: if one of the vertices of a triangle in the triangulation has a different path than the others, new rays are traced at in the middle of the edges, leading to four new triangles, see Figure 5.6. This process can be continued until a certain accuracy is obtained. We will discuss the influence of $N_e$, $N_i$ and the number of iterations on the accuracy in Chapter 6.

5.2.3 α-shapes

The previously described methods map every data point to a point in the target phase space and together with the fact that the luminance is constant along a ray, the luminance is known for every data point in the target phase space and besides the luminance, also the path that every ray has followed from source to target is known. The result of ray tracing is now a point cloud in target phase space from which we want to derive, for example, the intensity.
Figure 5.7: From point cloud (a) we can make a Delaunay triangulation (b). The outer edges form the convex hull instead of the ‘J’ form. (c) and (d) show the α-shape with α = 4 and α = 1.5 respectively.

The point cloud describes areas of rays that have followed the same path. In the next part a method is described to approximately obtain the boundaries of these areas.

Suppose we have a sample of points inside the interior of a region that has a J-shape, see Figure 5.7a. Although the shape is clearly visible by eye, it is not straightforward to recover the boundary of the shape. A generalization of the convex hull is called α-shape. It starts with a Delaunay triangulation of the point cloud. As is shown in Figure 5.7b, the boundary of this triangulation does not give us the desired shape, but the convex hull of the point cloud. Suppose now that we have a circular eraser with radius α and all the area in the convex hull is erased, without removing any of the sample points. The boundary of the shape now consists of curved lines that connect the outer sample points. The curved lines are now straightened to obtain the so-called α-shape, see Figures 5.7c and 5.7d. The value of α directly influences the final shape, hence the name.

In more mathematical terms, this process is carried out as follows: for every triangle with edges with length \(a, b, c\), calculate the radius of the circumcircle given by

\[
\frac{abc}{\sqrt{(a + b + c)(a + b - c)(a + c - b)(b + c - a)}}. \tag{5.2}
\]

If this radius is larger than \(α\), the triangle is removed from the shape. The smaller the radius \(α\), the more matter is removed and the final surface area becomes smaller. When \(α = \infty\), the α-shape is simply the convex hull. Figure 5.7 shows that a good choice of \(α\) can (at least visually) lead to a good approximation of the shape represented by the cloud.

In the example above, the parameter \(α\) has a large influence on the final shape. \(α\) should be small enough such that nuances in the shape are still visible, but it has to be large enough not to destroy too much of the shape. Increasing the number of points allows \(α\) to be smaller, but of course this has a downside in terms of computation speed. Hence we should find a criterium for a good choice for \(α\).

It is very unfavorable if a triangle in the interior of the shape is removed from the shape. From this criterium a lower bound for \(α\) can be determined. Since a triangle can be removed only if the radius of the circumcircle is larger than \(α\), this is closely related to the problem: find the radius of the largest empty circle in a random data set. The radius of the largest
empty circle should change as the density of the data set changes, where the density $\Delta$ is defined as

$$\Delta = \frac{N}{\text{surface area}},$$

(5.3)

with $N$ the total number of points. Now suppose we scale the dimensions of the data set by a factor 2. Then also the radius of the largest empty circle increases by a factor 2, but the surface area increases by a factor 4 and $\Delta$ decreases by a factor 4. Therefore we have

$$\alpha \sim \frac{1}{\sqrt{\Delta}} \Rightarrow \alpha = C \frac{1}{\sqrt{\Delta}}$$

(5.4)

with $C$ a constant that can be determined by a simulation. In the simulation, in each run a random data set is chosen on a square domain and the $\alpha$-shape is determined for a range of values for $\alpha$. After a large number of runs, we find the smallest $\alpha$ such that in none of the runs the $\alpha$-shape contained wholes. We find $C \approx 3.5$.

This dependence of $\alpha$ on the data density can be used as follows: $N_1$ rays are chosen randomly in the interior of $P_s$. The interior has a known surface area and this area is conserved in $P_t$, hence the (global) density is known.

5.3 Interpolation for scattered data

After ray tracing points from $P_s$ we end up with information about the luminance on scattered points in $P_t$. If we want to integrate along a line to obtain the intensity or illuminance, we need to know the luminance on points in phase space that we do not necessarily have data of. Therefore, an accurate interpolation method is required and since the number of data points is large, the method has to be very efficient.

The mathematical formulation is as follows: given a domain $\Omega \subset \mathbb{R}^2$ and a finite data set $D \subset \Omega \times \mathbb{R}$, what function $f : \Omega \to \mathbb{R}$ exists such that $f(x,y) = z$ for all $(x,y,z) \in D$, or perhaps $f(x,y) \approx z$. There may exist many such functions and in general there is no way to determine which one of these functions is the best for points outside the data. Performance of methods can only be tested with data samples from known functions.

The problem of interpolating (or immediately integrating) a function on a grid is much easier to address, for example with quadrature rules or with the use of B-splines [9]. Accuracy of these methods is often known in terms of the grid size. However, interpolating becomes a significantly more complicated problem if the data is scattered over the domain. One of the older methods in this field, known as Shepard’s interpolation [13], is fast, but has some difficulties. It is briefly described in Section 5.3.1. Another popular method for interpolation is known as Radial Basis Function Interpolation [9], becomes computationally hard around $\sim 10^3$ data points, which is still too low for our purpose.

We chose to look for an alternative and found that in a multilevel B-spline method [8]. The method is explained in full detail in Section 5.3.2 and several examples are provided to obtain an idea about the accuracy of the method.

5.3.1 Shepard’s interpolation

Some interpolation methods lead to a (complicated) global function, others lead to a set of local functions. Our number of data points is too large for the former method. An intuitive
method of the latter form is called inverse distance weighting. We follow the notation from [13]. Let \( P \) be the point for which we want to interpolate the data points \( D_i = (x_i, y_i) \) with value \( z_i \) and define \( d_i := d[P, D_i] \) the Cartesian distance between \( P \) and \( D_i \). The interpolation function is then

\[
\hat{f}(P) = \left\{ \begin{array}{ll}
\left( \sum_{i=1}^{N} (d_i)^{-u} z_i \right) / \left( \sum_{i=1}^{N} (d_i)^{-u} \right) & d_i \neq 0 \quad \forall D_i, \\
0 & \exists d_i = 0.
\end{array} \right.
\] (5.5)

\( u \geq 0 \) is a free parameter in the problem, often \( u = 2 \). Lower exponents lead to steep gradients around the data points and cause steep gradients in between. Points far away from \( P \) have very little influence on the interpolated value, so instead of summing over all data points, choosing only the nearest \( M \) is sufficient, hence making it a local function (whereas in (5.5) it was still global.

The function \( \hat{f} \) has desirable properties such as:

- \( \hat{f} \) is exact for the data points and continuous everywhere.
- \( \hat{f} \) is differentiable if \( u \leq 1 \).

Undesirable is, however, that the gradient in data points is 0 which makes it inaccurate and in the neighbourhood of a data point taking the difference of two almost equal numbers leads to computational errors. These properties can be taken care of with methods presented in Shepard’s article, but we resort to a more recent method.

### 5.3.2 B-spline interpolation

Another interpolation algorithm, described in [8], makes use of B-splines. Assume that the data set \( D \) is subset of a rectangular domain

\[
\Omega = [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}].
\] (5.6)

We define a grid \( (\Phi_{ij}) \),

\[
\Phi_{ij} = \left( \begin{array}{c}
x_{\min} \\
y_{\min}
\end{array} \right) + \left( \begin{array}{c}
(i-2)\Delta x \\
(j-2)\Delta y
\end{array} \right), \quad i = 1, 2, \ldots, m+2,
\]

\[
\left( j = 1, 2, \ldots, n+2, \right)
\] (5.7)

with grid sizes \( \Delta x := (x_{\max} - x_{\min})/(n-1) \) and \( \Delta y := (y_{\max} - y_{\min})/(m-1) \), such that \( \Phi \) completely covers \( \Omega \), see Figure 5.8 (note that \( \Phi \) has nothing to do with the flux described in Section 2.2). We define a function \( v : \Phi \mapsto \mathbb{R} \) that maps every grid point to a certain value

\[
v(\Phi_{ij}) = \phi_{ij}.
\] (5.8)

The aim is to approximate a function \( f : \Omega \mapsto \mathbb{R} \) of which the function values are known only for points in \( D \), with a function \( \hat{f} : \Omega \mapsto \mathbb{R} \), defined as

\[
\hat{f}(x,y) = \sum_{k=0}^{3} \sum_{l=0}^{3} B_k(s) B_l(t) \phi_{i+k,j+l},
\] (5.9)
Figure 5.8: The rectangular domain $\Omega$ is completely covered by the grid $\Phi_{ij}$.

with $i = \lfloor (x-x_{\text{min}})/\Delta x \rfloor - 2$, $j = \lfloor (y-y_{\text{min}})/\Delta y \rfloor - 2$, $s = (x-x_{\text{min}})/\Delta x - \lfloor (x-x_{\text{min}})/\Delta x \rfloor$, 
$t = (y-y_{\text{min}})/\Delta y - \lfloor (y-y_{\text{min}})/\Delta y \rfloor$. $i,j$ indicate the left- and downmost grid point that still influences the value of $f(x,y)$ and $s,t \in [0,1)$ are generalized coordinates with respect to a grid cell. The $B_i$ are the cubic B-spline basis functions:

$$
B_0(t) = (1-t)^3/6, \\
B_1(t) = (3t^3 - 6t^2 + 4)/6, \\
B_2(t) = (-3t^3 + 3t^2 + 3t + 1)/6, \\
B_3(t) = t^3/6,
$$

(5.10)

with $0 \leq t < 1$, see Figure 5.9.

Every function value $\hat{f}(x,y)$ is now a weighted sum of 16 grid values $\phi_{ij}$ as in (5.8). We need grid points outside $\Omega$ so that data points at the boundary of $\Omega$ can also be dealt with. The values $\phi_{ij}$ are the unknowns that need to be determined.

If we want $\hat{f}$ to be exact in each data point, this gives an equation of the form

$$
z = \sum_{k=0}^{3} \sum_{l=0}^{3} w_{kl} \phi_{kl} = (w, \phi),
$$

(5.11)

with $w_{kl}$ a shorthand notation for $B_k(s)B_l(t)$ and $w, \phi \in \mathbb{R}^{16}$. Lee [8] suggests to take the Moore-Penrose inverse for vectors (least squares solution) to solve this equation for $\phi$.
Figure 5.9: The B-splines have compact support by construction and sum up to 1.

\[ \phi = \frac{w_z}{w \cdot w}, \]  

(5.12)

but there are infinitely many solutions to (5.11) and we will discuss other possibilities later on in this chapter. When we compare the different solutions, this solution is referred to as the B-spline least squares interpolation (BLSI).

If we solve Equation (5.11) for all data points, a grid point \( \Phi_{ij} \) gets assigned (probably different) values by all points close enough to \( \Phi_{ij} \), in other words, the function \( v \) is not well defined. Define \( \mathcal{C} \) as an index set, such that for \( c \in \mathcal{C} \), the data point \((x_c, y_c)\) is close enough to \( \Phi_{ij} \) and define \( \phi_c \) the value for \( \phi_{ij} \) assigned by that data point and \( w_c \) the corresponding weight. We then choose

\[ \phi_{ij} = \frac{\sum_c w_c^2 \phi_c}{\sum_c w_c^2}, \]  

(5.13)

such that \( \sum_c (w_c \phi_{ij} - w_c \phi_c)^2 \) is minimized.

The coarseness of the grid influences how spiky the surface is. If for example we would have one data point located at \((0, 0)\) with value 1, the choice for the grid size influences how broad the peak is, see Figure 5.10. Therefore, by applying this algorithm only for one grid size we cannot tell whether or not we have a good approximation of the function. It can however be used in an iterative way as described in the following algorithm:

- Start with an initial grid \( \Phi^0 \) with grid sizes \((\Delta x)^0\) and \((\Delta y)^0\) and apply the algorithm to obtain a first approximation \( \hat{f}^0 \) for \( f \). If the grid sizes are large (with respect to the data density), the function \( \hat{f}^0 \) is slowly varying and produces an error \( (\Delta z)^0_i = \hat{f}^0(x_i, y_i) - z_i \) for every data point.

- Now halve the grid sizes and apply the algorithm again but now on the points \((x_i, y_i, (\Delta z)^0_i)\). This produces another grid \( \Phi^1 \) and a function \( \hat{f}^1 \). Decreasing the grid size allows the surface to have more detail and the function \( \hat{f}^1 \) is able to correct part of the error made by \( \hat{f}^0 \), so the \( \hat{f}^0 + \hat{f}^1 \) is a better approximation. It might still have an error \( (\Delta z)^1_i = \hat{f}^1(x_i, y_i) + (\Delta z)^0_i \).
Figure 5.10: Spline approximation of the point \((0, 0, 1)\) with grid sizes (top to bottom) \(\Delta x = \Delta y = 2, 0.25, 0.0625\). The finer the grid, the spikier the surface. Left shows the approximation with BSPI, right with BGI where a gradient \((1, 0)\) is included in the approximation.
• The previous step is repeated, so that the method produces a function $\hat{f} = \sum \hat{f}_i$ that is either sufficiently accurate or, if needed, even exact at the data points.

![Diagram of grids and calculations](image)

Figure 5.11: Large dots form grid $\Phi^i$, all dots form the finer grid $\Psi^i$. (a)-(d) Values for the red dots can be calculated from the values for the black dots.

Every iteration in this algorithm produces a grid $\Phi^i$ with increasing number of grid points. The function $\hat{f}$ can only be evaluated in a point $(x, y)$ by summing the results of Equation (5.9) for every $\Phi^i$. There is a way to circumvent having to store all the $\Phi$'s.

With every iteration step the grid size is halved, such that all grid points of $\Phi^i$ are also contained in $\Phi^{i+1}$ and $\Phi^{i+1}$ contains also points in between. It is possible to create a new grid $\Psi$ from $\Phi^i$ that represents the same surface as $\Phi^i$, but has the same grid points as $\Phi^{i+1}$. This problem is generally known as knot insertion and there are many algorithms to solve it [5]. Because of the regularity of our “insertions”, we can follow the approach as in [8].

Figure 5.11 illustrates how the grid refinement works: the new grid points are a weighted average (with weights depending on the B-splines) of the surrounding grid points in $\Phi$, so that the inserted points do not disturb the surface described by $\Phi$. The values $\psi_{ij}$ of the finer grid $\Psi$ can be calculated as follows:

\[
\begin{align*}
(a) & \quad \psi_{2i-2,2j-2} = \frac{1}{4} \sum_{k=i-1}^{i} \sum_{l=j-1}^{j} \phi_{kl}, \\
(b) & \quad \psi_{2i-2,2j-1} = \frac{1}{16} (6 (\phi_{i-1,j} + \phi_{i,j}) + \phi_{i,j+1} + \phi_{i-1,j+1} + \phi_{i,j-1} + \phi_{i-1,j-1}), \\
(c) & \quad \psi_{2i-1,2j-2} = \frac{1}{16} (6 (\phi_{i,j-1} + \phi_{i,j}) + \phi_{i+1,j} + \phi_{i+1,j-1} + \phi_{i-1,j} + \phi_{i-1,j-1}), \\
(d) & \quad \psi_{2i-1,2j-1} = \frac{1}{64} (\phi_{i-1,j-1} + \phi_{i-1,j+1} + \phi_{i+1,j-1} + \phi_{i+1,j+1}) \\
& \quad + \frac{1}{64} (6 (\phi_{i-1,j} + \phi_{i,j-1} + \phi_{i,j+1} + \phi_{i+1,j}) + 36 \phi_{ij}).
\end{align*}
\]
Now if, in the next step, we add $\Phi^{i+1}$ and $\Psi^i$, there is no need to remember the previous $\Phi$’s and $\Psi$’s, memory costs are reduced significantly.

Equation (5.12) is just one of the many choices to solve Equation (5.11). It is the least squares solution, so every data point is approximated with a bump that tends to zero away from the point faster than with other choices. However, experiments show that this causes the interpolation to have the unwanted property that it gives too high values (especially in the first iterations). Intuitively, this is caused by the fact that a data point close to a grid point assigns a higher value to that particular grid point than to the other 15 and also has a larger weight, such that in Equation (5.13) $\phi_{ij}$ is likely to receive a slightly too high value.

If we want to interpolate data points that all have the same value, it is likely that they represent a flat surface with that same value. Another choice to solve (5.11) is with

$$\phi = z_c \mathbf{1},$$

with $\mathbf{1}$ the all one vector. This tends to approximate every data point with a flat surface around it, hence it interpolates a set of data points with constant value in only one iteration step. It is expected to work well for slowly varying surfaces, but perhaps worse for rougher surfaces. Can we improve our solution even more?

Suppose that at every data point we not only know the value, but also the gradient of the surface. Instead of approximating a data point by flat surface (zero gradient) it would be better to approximate with a surface with the same gradient. We find this surface by solving

$$\sum_{k=0}^{3} \sum_{l=0}^{3} w_{kl}(\phi + k \partial_x f + l \partial_y f) = z_c \quad (5.16)$$

for $\phi$. Then $\phi_{kl} = \phi + k \partial_x f + l \partial_y f$. The surface interpolates the point and has the same gradient.

In general we do not know the gradient, but we do have the triangulation. The gradient in a point will be approximately equal to the gradient of the surface through a triangle involving this point. Since every point is vertex of multiple triangles, we choose the triangle with the smallest perimeter, so the approximation of the gradient should be accurate. With this principle we get remarkably good results. The interpolation is exact after one iteration for all affine surfaces. From now on we will refer to this as method B-spline gradient interpolation (BGI). Figure 5.10 shows the difference between the two methods for approximating a point. BLSI gives the least squares solution of (5.11) and therefore produces a hump that tends to 0 faster then with BGI. BLSI gives the possibility to not only approximate a point, but also use gradient information in that point. A further analysis of the accuracy should point out which of the two methods is better.

### 5.3.3 Accuracy of B-spline interpolation

We have talked about approximating or interpolating data sets, but we have not even defined properly what we mean by this. The problem is that we only have information in the data points and not outside, so there is no ground truth. As described in [4] it is not easy to find good criteria for these methods. A “good” approximation could be a function $f$ that is exact in the data points and "visually" appealing in some sense: the human eye has its own way of creating a surface from a point cloud. That is however not a workable criterium, so we decide to test the accuracy of the interpolation methods as follows: choose a number of test
functions that have characteristics we are interested in (we soon discuss what functions are of our interest). We sample the data set \( D \) from such a function and perform the interpolation. To test the accuracy of the interpolation, we sample another data set \( D_T \) and define the error as

\[
e = \frac{1}{|D_T|} \sum_{D_T} |\hat{f}(x, y) - z|.
\]  

(5.17)

The goal of the approximation is to minimize this error. If the method is able to do this for the test functions, we have to assume that it works well for the actual purpose: interpolating the luminance function.

The spline based method explained in the previous section seems to offer a fast solution to the interpolation problem, but it is unclear how well it performs in various cases. The main concerns are:

- The target phase space can have a quite irregular shape and since all data points lie inside this shape, we have no information about the luminance outside. How accurate is then the interpolation at the boundary?
- How does the method perform when the luminance has large gradients or even jumps?
- How does the accuracy of the method increase when the number of data points is increased?
- How much does the initial grid size and number of iterations influence the final approximation?

These are a lot of questions to answer in full detail, but in the next part a number of test cases are described that give a first step to answering these questions.

Case 1. We start with randomly distributed data points inside a circular domain, see Figure 5.12, with constant luminance 1:

\[
f_1(x, y) = 1.
\]  

(5.18)

Case 2. To check the performance of the interpolation when the function is rapidly varying, we take the setting as in Case 1, but with

\[
f_2(x, y) = \cos 2\pi x \cos 2\pi y.
\]  

(5.19)

Figure 5.13 shows that both methods are able to reconstruct the surface inside the unit circle (where the data points lie), but it also shows the difference in behavior of the two approximations outside this region. BLSI tends to 0 quite rapidly, while BGI extends the surface in the direction of the gradient (but tends to 0 further along the axis, not shown in the figure).

Case 3. Things might get ugly when the function we want to approximate has a jump, since this can never be approximated by a \( C^2 \) function. We test this with

\[
f_3(x, y) = \text{sign}(x).
\]  

(5.20)
Figure 5.12: Data points inside the unit circle, all with luminance 1.

Figure 5.13: Spline approximation for Case 2: \( f(x, y) = \cos(2\pi x) \cos(2\pi y) \). |\( D \)| = 3200. (a): BLSI, (b): BGI, (c) exact surface.

In Case 1, with BGI, the correct surface is found immediately and no boundary effect is visible. With BLSI, the error decreases linearly with the number of points, see Figure 5.15. Although we have a circular domain, also BLSI does not show severe inaccuracies at the boundary. Also in Case 2 both methods are able to reconstruct the wilder surface, even at the boundary. Therefore, the irregularity of the domain does not arise to problems.

Figure 5.16 shows for Case 2 that BGI has a faster convergence than BLSI as the number of data points increases. For Case 1 BGI was exact after one iteration, hence we favor BGI over BLSI. The results in Chapter 6 are based on the improved interpolation BGI.

Case 3 shows that the interpolation algorithm is not designed to interpolate jumps, see Figure 5.14. This is intrinsic to the nature of the constructed function: a discontinuous function cannot be interpolated by a (twice) differentiable function. This leads to the important remark that the interpolation should be carried out for each domain \( M_\Pi \) separately.

The fourth and last question is tricky to answer. For Case 2 the error is plotted as function of the number of iterations in Figure 5.17. We start with different initial grid sizes, but the iteration process is stopped when the same final grid size is reached. It seems that the algorithm makes the largest improvement in accuracy at the same transition in grid sizes, in this particular case when the grid size changes from \( \Delta x = 0.25 \) to \( \Delta x = 0.125 \). When the
Figure 5.14: Spline approximation for Case 3: \( f(x, y) = \text{sign}(x). |D| = 3200. \) (a): BLSI, (b): BGI, (c) exact surface.

Figure 5.15: For Case 1 and BLSI, the error scales inversely proportional to the number of points.

initial grid size is larger than this, we see that the first steps do not lead to a big improvement. If the initial grid size is smaller than this, not the same accuracy is reached.

Which iteration step leads to the largest gain in accuracy is most likely dependent on the roughness of the function. This is unknown in advance, so it is difficult to make the right choice for \((\Delta x)^0\). Therefore we cannot rule out that the first steps in the algorithm are more or less redundant. This is a disadvantage, but on the other hand this is the power of the multilevel approach: if \((\Delta x)^0\) is chosen large enough, the interpolation will eventually reach the grid size where the most improvement is made. After that, the iterative process can be stopped, since Figure 5.17 shows that only a few steps are needed to obtain the final accuracy.

5.4 Integration of the luminance

Sections 5.2.3 and 5.3 together offer the domain in phase space where the luminance is nonzero and an approximation for the luminance on that domain. The only hurdle that is left to take
is integrating the approximated luminance function over the position $q$ or angle $\theta$. For the intensity we had:

$$ I(t)(\theta) = \cos \theta \int_S L_t(q, \theta) dq. \quad \text{(5.21)} $$

Recall from (5.9) that the approximating function is given by

$$ \hat{f}(x, y) = \sum_{k=0}^{3} \sum_{l=0}^{3} B_k(s) B_l(t) \phi_{i+k,j+l}, \quad \text{(5.22)} $$

and that the $\alpha$-shape gives for every $\theta$ the integration boundaries for $x$, see Figure 5.18.

Suppose we want to integrate a function $f(x, y)$ over $x$ for a fixed value $y$. Then in every interval between two black dots in Figure 5.18 the representation of $f$ is unchanged, i.e., the sixteen values of $\Phi$ are constant on such an interval. It is therefore convenient to write the

Figure 5.16: BGI (II) performs better than BLSI (I) in Case 2.
Figure 5.17: Plot of the error vs the iteration number for Case 2. (a) BLSI (b) BGI. Interpolation is started with different grid sizes, iteration is stopped at the same grid size. $|D| = 800.$

The remaining integrals are nothing more than integration of polynomials, so the entire integration can be carried out exactly. By multiplying with the factor $\cos\theta$, for every $\theta$ now the intensity can be calculated.
Figure 5.18: For every $y$ we can integrate $f$ over $x$ between the red integration boundaries, i.e., over the interval $[x_l, x_u]$. In intervals between the black dots, the representation of the function $f$ is the same.
Chapter 6

Results

In the previous chapter we have seen the machinery that offers an alternative to Monte Carlo ray tracing. With a couple of test cases we have checked the performance of the interpolation step, but in this chapter we investigate how well the method works from beginning to end and compare it to the MC method. Chapter 3 gave us two optical systems for which we have exact solutions: the two-faceted cup and the CPC. We start this chapter with results of these two systems. We then continue to explore more difficult systems. For the examples we will see, the luminance function on the source is either chosen to be \( L_s = \text{constant} \) or \( L_s(x,t) = \cos(t) \). The first choice corresponds to a Lambertian source and the interpolation step is not needed there. In the second case also the interpolation step becomes important.

6.1 Faceted cups

We study a two-faceted cup with \( a = 5 \), \( b = 20 \), \( h = 40 \) and \( \gamma = \arctan(3/8) \), recall Figure 3.2. The target is placed at \( 2h \). We take the source \( S = [-4,4] \). Figure 6.1 shows how the different areas in \( P_s \) are transformed by the optical system and end up in \( P_t \). The results in this figure are exact and it is possible to find the exact intensity as well. This gives us a nice test case for the numerical methods. First we will reduce the number of rays as much as possible, while still obtaining acceptable results. Then we increase the number of points and see how accurately we can reproduce the exact results.

When reducing the number of points, it becomes clear that it is important to trace rays emitted from the boundary of the source to determine the shape of the phase space. Assume for now that the source is Lambertian, so the luminance is constant and there is no data needed in the interior of the regions in \( P_t \). Results for 200 and 2000 edge rays are shown in Figures 6.2 and 6.3, respectively. The edge rays form the curved boundaries, there is only a small error at the left and right sides of the regions. It is, however, needed to trace rays from the interior of the source to correct this small error in the phase space shape. Also, if the luminance is non-constant, these rays are needed to interpolate accurately. Suppose in addition to the rays from the boundary of the source, we trace a set of randomly chosen rays from the interior of \( P_s \). Then with use of the triangulation of this set of points, a new ray set is induced: it contains only rays on edges between rays that have different ray paths, see Figure B.1a. Figures B.1c-B.1g then show the result of such a refinement. Clearly, more rays are traced close to the boundary between regions of different ray paths in \( P_s \). Figures B.1b-B.1h show that this has the desired effect in \( P_t \): the extra rays...
Figure 6.1: Phase space of the source (left) and the target (right) of the two-faceted cup. Likewise colors indicate the same ray path.

Figure 6.2: With only 200 edge rays we have a good approximation of the phase space shape, there is only a small error in the intensity.

get closer to the part of the boundary where there was no information yet.

From the ray set in Figure B.1h we can show an intensity plot as well, but it can hardly be distinguished from 6.3b. Instead we show how the error decreases as the number of points increases. The error in this case is defined as:

$$e = \sum_{k=1}^{N} |I_{\text{app}}(\theta_k) - I_{\text{exact}}(\theta_k)| / N,$$

where $\theta_k, k = 1 \ldots N$, are a regular partition of the interval $[-\pi/2, \pi/2]$.

There are three ways to increase the size of the final data set: by increasing the number
of edge rays, by increasing the number of rays in the interior, and by increasing the number of iterations. We have already seen the importance of the edge rays, so we fix this number and see how the other two parameters influence the final accuracy.

A comparison with the Monte Carlo method should point out how well the method really works. Figure 6.4a shows that the error of the Monte Carlo method decreases with order 1/2. The other lines are results from a simulation starting with the number of rays as indicated and then refining until $10^5$ rays are traced. Not only is the error of the phase space method much smaller, the larger slope indicates a higher convergence speed. Depending on the desired accuracy, the number of rays needed can be reduced by at least a factor 100. Furthermore, the figure shows that the size of the initial ray set has little influence on the convergence speed, but that a smaller initial ray set (and more iterations) lowers the error.

Figure 6.4: Errorplot for the 2 faceted cup (left) and the 6 faceted cup (right). The initial ray set has the size as indicated by the different colors and we iterate until we reach $10^5$ rays. The last line corresponds to a MC simulation.
A slightly more complicated optical system is the 6-faceted cup as shown in Figure 6.5a. The height of the cup is the same as the height of the 2-faceted cup, but both sides are divided into three linear segments. The input intensity here is $I_s = \cos^2 t$, so that $L_s = \cos t$.

Figure 6.5: The phase space (middle) of the 6-faceted cup (left) has more different regions, but the method still performs very well. Compared to a Monte Carlo simulation with a comparable number of rays ($10^4$) the intensity profile is much more accurate.

Although the phase space for the 6-faceted cup is more complicated, the algorithm still outperforms the Monte Carlo method.

### 6.2 CPC

We now leave the world of linear segments behind and look at the CPC, see Figure 6.6a, the concentrator with parabolically shaped reflectors for which we could still derive an exact solution for $I_t$ in the case that $L_t =$constant.

Figure 6.6: A CPC with the same height and source as the faceted cups. Most rays leave the source with at most one reflection, but some rays can ‘crawl’ along the reflector and reflect many times before leaving the CPC.

There is a difficulty in finding the boundaries of areas with the same number of reflections, since a ray can skim over the reflective surface and reflect multiple times, see Figure 6.6b), or in fact infinitely often. Hence it is not possible to discover all regions $M_n$ and the previously used method might not be accurate. The areas with two or more reflections are nested at the
boundary and eventually become infinitely small, see Figure 6.7. We chose to put all rays that reflect more than 10 times in one region. An intuitive argument for this choice is that these regions are then hopefully small enough so that they have very little influence on the intensity. Also, the rays in these regions are emitted at approximately the same angle, so the luminance will not have large jumps.

Figure 6.7: Rays in a CPC can theoretically reflect infinitely many times. The corresponding regions in phase space become smaller and smaller.

Figure 6.8: The target intensity of the CPC with a Lambertian source. $5 \cdot 10^4$ rays are traced for both the numerical method and MC method.

Figure 6.8 shows that it is still possible with only $5 \cdot 10^4$ rays to give a good approximation of the exact intensity. The simulation that leads to this result had a large initial ray set, 3200 rays. Choosing the initial ray set too small can lead to persistent errors in the $\alpha$-shapes, as can be seen in Figure 6.9: when we choose an initial ray set with 200 rays, the method does not converge, because $\alpha$ cannot be chosen small enough. However, with 800 or more initial rays, the method produces much smaller errors than the MC method and again a higher
convergence order.

![Errorplot for the CPC with Lambertian source.](image)

Figure 6.9: Errorplot for the CPC with Lambertian source. The initial ray set has the size as indicated by the different colors and we iterate until we reach $10^5$ rays. The last line corresponds to a MC simulation. A good choice for the size of the initial ray set is important: choosing it too small prevents the method from converging.

Just like with the faceted cups we want to know what happens if we take $L_s(x, t) = \cos t$, but for the CPC we do not have an exact solution anymore. We can run a Monte Carlo simulation with $10^8$ rays to obtain approximately the correct intensity and use this as ground truth. Figure 6.11a shows that the intensities found with the phase space method do not converge.

An alternative to using a Monte Carlo simulation as ground truth, we could use the phase space method with a large number of rays to obtain an accurate intensity, see Figure 6.10. It seems to have a smoother profile than the Monte Carlo intensity. Figure 6.11b shows the convergence results with this intensity. These results are very similar to the case where $L_s = \text{constant}$.

### 6.3 TIR Collimator

We have seen two optical designs with well-behaved reflective surfaces, but we now consider a luminaire that might be a real challenge to the phase space ray tracer. Total internal reflection (TIR) collimators are actually used optical designs, an example is shown in Figure 6.12. The (red) source is placed inside air, the volume inside the blue line is filled with a solid with $n = 1.6$. The central curve and the adjacent straight sides are refractive surfaces, the curved segments on each side are designed such that light is internally reflected. Eventually it hits the top surface and is refracted again before it leaves the collimator.

There is a lot of design freedom for the TIR collimator: the dimensions (size) of the collimator and the shape of the central segment and the side segments influence the properties of the light output. It is therefore desirable that such a collimator can be traced fast and accurately. The hope is that the method that worked well for the easier designs is robust enough to perform well for this design, but there is reason to assume that the curved surfaces
Figure 6.10: The Monte Carlo method with $10^8$ rays and the phase space method with $10^6$ rays should both give a good approximation of the exact intensity.

Figure 6.11: An approximately exact intensity can be found with the Monte Carlo method with a large ray set, or with the phase space method with a large ray set. (a) shows convergence results of approximate intensities compared to the (almost correct) MC intensity, (b) shows the results compared to the (almost exact) phase space intensity.

in the collimator change phase space more drastically than in the easier designs, so especially determining the $\alpha$-shape of $P_t$ might have increased in difficulty.

Figure 6.13 shows that phase space of the collimator is indeed much more irregular than before. The $\alpha$-shape in this case is not in all parts accurate enough and increasing the number of rays does not overcome this problem. The consequence is that the intensity, see Figure 6.14 is too high for these angles where the volume is inaccurate. The difference in the center between the two intensities is of the same order as with the faceted cup or the CPC, so there the phase space method outperforms the Monte Carlo method. Finding the correct shapes in $P_t$ is at this moment the bottle neck of the problem. Ideas to overcome this problem are presented in Chapter 7.
Figure 6.12: Example of a TIR collimator.

Figure 6.13: Phase space of the collimator has an irregular shape. Although the α-shape accurately finds the boundaries of most of the regions, in some parts detail is lost.

6.4 Calculation times

In the previous sections we have compared the Monte Carlo method with the phase space method and we have seen that in terms of the number of rays needed, the new method is much better. In this section we have a quick look at what this means for the time that can be gained.

We have seen that the accuracy of the Monte Carlo method converges proportional to $1/\sqrt{|D|}$. The good thing about this method is that the calculation time is proportional the number of rays. How fast the ray tracing actually is, depends on the complexity of the optical system. For the two- and multi-faceted cup there is a closed expression for calculating the intersection of the ray and the segments. The same holds for the CPC, but ray tracing through a CPC is even faster since it is predetermined by the sign of the initial angle if a ray hits the right or left reflector and most rays reflect at most once. For the most realistic case, where the optical system consists of reflectors and lenses represented by B-splines, the calculation times increase significantly since an iterative method is needed to calculate the ray-segment intersections. Calculation times can be found in Table 6.1. Note that these are calculation
times for 2D systems, 3D systems will take more time.

<table>
<thead>
<tr>
<th>Optics</th>
<th>Calculation time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>two faceted cup</td>
<td>~ 25</td>
</tr>
<tr>
<td>six faceted cup</td>
<td>~ 25</td>
</tr>
<tr>
<td>CPC</td>
<td>~ 10</td>
</tr>
<tr>
<td>TIR collimator</td>
<td>~ 300</td>
</tr>
</tbody>
</table>

Table 6.1: Calculation times for $10^5$ rays for the different optical systems considered.

In the phase space tracer, the refining method does not take too much time, nor does calculating the $\alpha$-shape. Finding a triangulation of a point cloud can be done in $|D| \log |D|$ time, with $|D|$ the number of points (so worse than linear), but for $10^5$ rays it is still very fast. It turns out that the bottle neck of the phase space method is the interpolation part. There is a trade-off between the number of iterations and the accuracy of the method. According to [8], the time complexity of the method is $O(h|D| + mn)$, where $h$ is the number of iterations and $(m + 3) \times (n + 3)$ the number of points in the finest grid. Although the interpolation method is quite expensive, it is still linear in the number of rays. In practice we see that interpolation for $10^5$ rays takes about 11 seconds for every iteration. Suppose we take 10 iterations, then in the same time we can trace about $10^6$ rays through the CPC, $4 \cdot 10^5$ through a faceted cup and $4 \cdot 10^4$ through a TIR collimator. We have seen that we need at least $10^8$ rays with the MC method to obtain the same accuracy as with the phase space method. That means that the time gain is incredible.

This section is concluded with an error versus time plot for the six-faceted cup with $L_s(x,t) = \cos t$, see Figure 6.15. Depending on the desired accuracy, the phase space method can be up to a factor 100 faster or even more.

Figure 6.14: Output intensity when $L_s =$constant, $|D| \sim 3 \cdot 10^4$, compared to a Monte Carlo simulation with $10^7$ rays.
Figure 6.15: Error versus time plot for the six-faceted cup.
Chapter 7

Conclusions and recommendations

In the previous chapters we have put ray tracing in a mathematical framework and provided a new method for fast ray tracing. Optical phase space offers new possibilities for ray tracing, but requires more mathematics than the standard Monte Carlo methods. We have extensively discussed the different steps in Chapters 5 and 6, from which we draw the necessary conclusions in this chapter. Along the way some suggestions for improvements are given.

• The first step in the method was to find the boundaries of areas with distinct ray paths, the regions $\mathcal{M}_{\Pi}$. Because these regions usually cover only a small part of the target phase space, forward tracing from the source is preferred over back tracing from the target. We therefore aimed to find the regions $\mathcal{M}_{\Pi}$ rather than $\mathcal{M}_{\Pi}$.

At first we chose to divide the source phase space into a rectangular grid, but since memory costs are then increasing quadratically, it is more convenient to use a triangulated structure. The initial ray set consisted of two parts: a random set of $N_i$ rays from the interior of $\mathcal{P}_s$ and a set of $N_e$ rays emitted from the edge of the source.

We have seen that choosing $N_e$ large ($\sim 10^3$) already makes a great first step to finding the boundaries of the $\mathcal{M}_{\Pi}$-regions. The choice for $N_i$ is more difficult. For the two-faceted cup choosing $N_i$ small ($\sim 25$) and refining many times lead to better results than with $N_i$ larger and less refining iterations. However, for the CPC and the TIR collimator we needed $N_i$ to be larger, to be able to choose $\alpha$ smaller.

The iterative refining process works well: with every step more rays are traced that end up close to the boundary of the $\mathcal{M}_{\Pi}$-regions and the process is robust. In the example with the two-faceted cup we have seen that we are able to reconstruct the regions $\mathcal{M}_n$ in the target phase space very nicely and visually it worked well for the more complicated optical systems.

• The second step was to find the boundaries of the point clouds in $\mathcal{P}_t$ found in the previous step. A method was used to construct the $\alpha$-shapes of the point clouds and although it worked well in the cases of the faceted cups and the CPC, there were issues with the TIR collimator, the most realistic case. The $\alpha$-shape method was unable to detect the finer detail in the boundaries. This part of phase space ray tracing is probably the most tricky part and it still needs some attention.
Improvement can be looked for in the fact that we have not used to the fullest that near the boundary of the $\mathcal{M}_H$-regions the density of points is much higher than in the interior. The $\alpha$-shape can be improved by removing edges that are much longer than the typical distance between points near the boundary. The downside is that this brings into play yet another parameter.

Another suggestion is to make better use of the fact that part of the boundaries of the $\mathcal{M}_H$-region are exactly formed by the edge rays of the source. This insight, combined with the $\alpha$-shape, should result in a better shape, but implementation could be devious.

- In the cases where the luminance $L_s$ was not constant, we had to apply an interpolation algorithm on scattered data. These algorithms are usually quite slow when the data set is large, but we found an accurate method that had a calculation time linear in the size of the data set. We even managed to use the earlier obtained triangulation to improve the accuracy of the interpolation without too much computational effort. The approximate function had the form of a sum of B-splines and this structure made it easy to integrate.

The interpolation method is subject to quite some choices: the initial grid size, the number of iterations and the choice for the solution of the underdetermined system in (5.11). This gives the interpolation method a somewhat uncomfortable feel, even though simulation results show how remarkably accurate and fast the method works. Therefore, effort should be put into finding a convergence proof of the interpolation and a good motivation of the choices made.

Speed improvements can be made by applying the interpolation not to all, but only a selection of the data points. This can reduce the calculation time when the ray sets become large, while the accuracy is most likely high enough.

A somewhat strange remark at this point is that the interpolation might be superfluous. In BGI we used the triangulation and the gradient information to improve the interpolation, but linear interpolation in each triangle and then integration could be accurate enough. This should be considered when continuing the research.

### 7.1 From two to three dimensions

The work in this study can be seen as the start of the development of a new way of ray tracing. It was at first not clear where the difficulties of this method would lie and how much time the new method could save compared to Monte Carlo tracing. The above conclusions show that there still are difficulties, but also ideas to overcome these.

The next step is then to translate the two-dimensional method to 3D. Although the 2D designs we have seen were symmetric and therefore in principle rotationally symmetric, the results cannot be translated to 3D directly. Ray tracing of a three-dimensional optical system is needed. In 3D, phase space has (at least) four dimensions, namely two position coordinates and two angular coordinates, and the luminance then adds one more dimension.

This means that we have to determine an $\alpha$-shape in 4D and that we need to extend the interpolation algorithm to 4 dimensions. In principle this should be possible: a 4D point cloud can be ‘triangulated’ with so-called 4-simplices, 4D volumes that are bounded by 5 tetrahedrons (3D simplices). The $\alpha$-volume can be determined with the same criterium as in
2D. For the interpolation, we should take the product of four splines instead of two and a 4D grid is needed, but the same algorithm should hold. Implementation will become a lot more difficult, therefore an even better understanding of the 2D case is needed.

A ray tracer preferably demands very little effort from the optical designer and should be robust and accurate. The phase space ray tracer has not yet reached that status and more test cases are needed to give a good indication for the choice of the parameters. For now, we can conclude that the method can yield considerable time savings compared to Monte Carlo tracing and certainly a way of ray tracing worth further investigation.
Appendix A

List of frequently used symbols

\( n \)  Refractive index  
\( \Phi \)  Luminous flux  
\( I \)  Luminous intensity  
\( E \)  Illuminance  
\( U \)  Étendue  
\( L \)  Luminance  
\( S \)  Light source  
\( T \)  Light target  
\( P \)  Phase space  
\( x \)  Ray position on light source  
\( t \)  Ray angle with normal of light source  
\( q \)  Ray position on target  
\( \theta \)  Ray angle with normal of target source  
\( \tau \)  \( n \sin t \)  
\( \eta \)  \( n \sin \theta \)  
\( z \)  Distance along the optical axis
Appendix B

Phase space pictures
Figure B.1: The initial data set is triangulated and a new ray set is assigned containing only points on edges that connect rays with different ray paths, here indicated with different colors.
Bibliography


