Physically Based Rendering of Ice Crystal Halos

by

Arthur Pereira Vala Firmino

B.Sc. with Honors, Physics, University of Alberta, 2016

A Project Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

University of Victoria

Physically Based Rendering of Ice Crystal Halos

by

Arthur Pereira Vala Firmino

B.Sc. with Honors, Physics, University of Alberta, 2016

Supervisory Committee

Dr. Brian Wyvill, Supervisor
(Department of Computer Science)

Dr. Kwang Moo Yi, Departmental Member
(Department of Computer Science)

Supervisor Committee

Dr. Brian Wyvill, Supervisor
(Department of Computer Science)

Dr. Kwang Moo Yi, Departmental Member
(Department of Computer Science)

ABSTRACT

The anisotropic scattering properties of atmospheric ice crystals lead to complex and visually appealing halos, however these have yet to receive due modelling in physically based rendering systems. This project aims to fill that gap and render high quality, physically accurate images of these phenomena. To capture the crystal's scattering properties, represented by a phase function, we construct a virtual gonioreflectometer simulating light rays incident on an ice crystal. A physically based renderer is extended to read our phase functions, allowing halos to be rendered by a volumetric path tracer. Multiple halos were rendered, with varying crystal shapes and orientations, and the resulting images were qualitatively compared with real photographs. The approaches used to approximate and sample the phase functions, despite yielding quality images, incurr an objectionable time and space cost. Possible alternatives and solutions are discussed in conclusion.

Table of Contents

List of Figures

List of Tables

Dedication

In memory of Sparkle,

who was a good kitty.

Chapter 1

Introduction

1.1 Physically Based Rendering

Physically based rendering (PBR) is a set of principles that combined aim to render more realistic images by accurately modeling the flow of light through a scene[1]. These principles are based on real world physics, such as the conservation of energy, guaranteeing that there is no more energy in the scene than is emitted by light sources, and Fresnel reflections, which describe the interaction of light at the interface between two media. Parameters in a physically based rendering system are often set using real world measurements and described using physical units. For example, a light source's power would be in terms of Watts.

Central to PBR is the rendering equation, also known as the light transport equation. It is a recursive function that describes light leaving a given point, for example the surface of an object. It was proposed in 1986 by David Immel [2] and James Kajiya [3], and can have the following form:

$$L_o(\mathbf{x}, \omega_o, \lambda) = L_e(\mathbf{x}, \omega_o, \lambda) + \int_\Omega f_r(\mathbf{x}, \omega_i, \omega_o, \lambda) L_i(\mathbf{x}, \omega_i, \lambda)(\omega_i \cdot \mathbf{n}) d\omega_i \tag{1.1}$$

Here $L_o(\mathbf{x}, \omega, \lambda)$ is the total exitant radiance of wavelength λ at a given position $\mathbf{x}$ along an outward direction ω. $f_r(\mathbf{x}, \omega_i, \omega_o, \lambda)$ is the proportion of light reflected in the direction ω_o from an incoming direction ω_i, also known as the *bidirectional reflectance distribution function* (BDRF). $L_i(\mathbf{x}, \omega, \lambda)$ is the total incident radiance at position $\mathbf{x}$ from an incident direction ω.

The equation boils down to saying that the total exitant radiance from a surface point is a function of the surfaces properties and the incident radiance at that point.

The rendering equation cannot be solved analytically, except for a few simple scenes, and is therefore solved numerically [2]. Path tracing, described later, can be thought of as performing a Monte Carlo integration of the rendering equation for each pixel in the image.

1.2 Volumetric Scattering

The rendering equation 1.1 does not describe volumetric scattering due to participating media, for example fog. Here there are three main processes affecting the transport of light that must be considered, *absorption*, *emission*, and *scattering*[1].

Absorption

Light travelling through a medium may be absorbed and converted to another form of energy, often heat. This reduction in radiance is described by the following differential equation:

$$dL_o(\mathbf{x}, \omega) = -\sigma_a(\mathbf{x}, \omega) L_i(\mathbf{x}, -\omega) dt \tag{1.2}$$

Here dt is a differential ray length, and $\sigma_a(\mathbf{x}, \omega)$ is the absorption cross-section [1].

Emission

Certain media consisting of luminous particle may emit light throughout its volume. The following differential equation can describe this effect [1]:

$$dL_o(\mathbf{x}, \omega) = L_e(\mathbf{x}, \omega)dt \tag{1.3}$$

In- and Out-Scattering

Light with incident direction ω may be scattered into some other direction. If only incident light with direction ω is of interest, out-scattering and absorption effects can be merged into one attenuation coefficient, σ_t [1].

Light may scatter into the direction of interest ω, increasing the radiance along that direction, in a process called in-scattering. For light scattering at a point, the *phase function* describes the angular distribution of the scattered radiance. The phase function is the volumetric analog of the BRDF, however unlike the BRDF it is normalized:

$$\int_{S^2} p(\omega, \omega')d\omega' = 1 \tag{1.4}$$

The effects of in-scattering and volume emission can be combined into one differential equation:

$$dL_o(\mathbf{x}, \omega) = L_s(\mathbf{x}, \omega)dt \tag{1.5}$$

$$L_s(\mathbf{x}, \omega) = L_e(\mathbf{x}, \omega) + \sigma_s(\mathbf{x}, \omega) \int_{S^2} p(\mathbf{x}, \omega', \omega) L_i(\mathbf{x}, \omega')d\omega' \tag{1.6}$$

1.3 Path Tracing

Path tracing is a Monte Carlo ray tracing algorithm that traces the path of individual light rays through a scene[2]. It is a Monte Carlo algorithm because, for example, when a surface interaction happens, a dice is rolled to decide whether the ray transmits though the surface or is reflected, the respective probabilities dependent on the surface properties.

Rays are often traced up to some maximum depth of interactions. Usually gathering rays are emitted from a sensor, such as the camera, out towards the scene to sample the incident radiance. The following algorithm [2] describes path tracing:

Algorithm 1 Path tracing algorithm.

Function $L_i(\mathbf{x}, \omega_i, d)$:

$\mathbf{x}' \leftarrow getNearestIntersection(\mathbf{x}, \omega)$
$L \leftarrow L_e(\mathbf{x}, -\omega)$
if $d >= MAX\_DEPTH$ **then**
  return L
end if
$\omega' \sim f(\mathbf{x}', \omega', \omega)$
$L \leftarrow L + L_i(\mathbf{x}, \omega', d+1) \cdot |\omega \cdot \hat{n}(\mathbf{x})|$
return L

Input: Film, $N_{samples}$
Output: Film

for all pixel **in** Film **do**
  for 1 **to** $N_{samples}$ **do**
    $\omega \sim pixel.SampleDirections$
    $pixel \leftarrow pixel + L_i(pixel.\mathbf{x},\ \omega,\ 0)$
  end for
  $pixel \leftarrow pixel\ /\ N_{samples}$
end for

Bidirectional Path Tracing

Bidirectional rendering methods trace paths starting from both the camera, and from light sources. These traced paths are then linked together to find the resulting radiance arriving at the camera[1]. Each path consists of a sequence of vertices. When two vertices from different paths are connected, it is necessary to know the radiance transport between the linked vertices. In the case of vertices corresponding to surface interactions, the bidirectional reflectance distribution function must be evaluated, and for interactions with participating media the phase function must be evaluated.

Metropolis Light Transport

A more sophisticated rendering algorithm, known as Metropolis Light Transport, applies the Metropolis-Hastings algorithm to BDPT [4], mutating previously sampled paths to generate new paths. These mutations can be done in a variety of ways, such as applying small displacements to path vertices. The advantage of MLT is that it can find more highly-contributing but difficult-to-find light paths, by mutating previously sampled highly contributing paths. In some scenes, MLT converges much faster than *path tracing* or *bidirectional path tracing*[1].

1.4 Mitsuba Renderer

The Mitsuba Renderer is a research oriented open source rendering system, derived from the PBRT renderer written for the Physically-Based Rendering book [1]. It implements many rendering methods, including all the algorithms described above [5].

Figure 1.1: Photograph of a 22° halo and parhelia. © Markus Derrer, reproduced with permission.

1.5 Atmospheric Optics Phenomena

The scattering of light by tiny ice crystals suspended in the atmosphere, oriented or otherwise, leads to many beautiful optical phenomena [6]. When air temperature is cold enough, water vapour often deposes into hexagonal shaped ice crystals [7]. Thin hexagonal plates tend to orient themselves, as falling leaves do, leading to anisotropic light scattering. They can reflect light from nearby sources, giving the impression of light pillars suspended in the atmosphere.

Due to these crystals hexagonal shape, light cannot scatter off the edge at angles smaller than 22°. Oriented plate ice crystals are responsible for parhelia, colloquially known as sundogs. When ice crystals are randomly oriented, more often the case for column ice crystals, 22° halos appear around the sun, Figure 1.1.

Chapter 2

Project Objective

The goal of this project is to answer, through experimentation and implementation, the following questions:

Can we model these phenomena in a Physically Based Renderer, such as Mitsuba?

The goal of this project is to answer whether we can model ice crystal halos in a physically based renderer like Mitsuba, and how one might do so. Isotropic volumetric scattering is already present in most renderers, and is often used to model fog, smoke, clouds among other things. Being able to model atmospheric phenomena such as light pillars, sundogs, and halos would add those visually appealing and physically inspired effects to artists toolboxes.

How can we represent an anisotropic wavelength-dependent phase function, considering space and time efficiency, and robustness?

Central to the problem of modeling atmospheric scattering due to ice crystals is how we can compute, represent, evaluate, and sample the 4-dimensional phase function while limiting the curse of dimensionality.

Chapter 3

Related Work

3.1 Atmospheric Optics Simulations

Optical phenomena such as halos and sundogs that occur due to ice crystal scattering have been extensively studied in the field of Atmospheric Optics [6] [8] [9] [7]. Greenler et al. (1972)[6] used a simple model to generate circumscribed halos in a Monte-Carlo simulation. Macke et al. (1996)[9] and Bi et al. (2014)[8] studied in depth the scattering properties of the responsible ice crystals. Um et al. (2015)[7] did a survey on measurements of the various crystal dimensions.

3.2 Halos in Computer Graphics

Glassner (1996)[10][11] extended the methods by Greenler to propose an algorithm to be used in computer graphics, which included casting rays for specific wavelengths. Gonzato et al. (2001) [12] extended this algorithm further to more accurately capture the scattered light's intensity, as well using a reconstruction filter for a cleaner image produced from fewer samples.

The methods used by Greenler et al., Glassner, and Gonzato et al. involve simulating

the scattering of light through a hexagonal ice crystal for each ray cast, when rendering an image. They do not pre-compute or provide a means to evaluate a phase function $p(\omega_i, \omega_o)$, making them unsuitable for production rendering algorithms such as *Path Tracing*, *Bidirectional Path Tracing*, and *Metropolis Light Transport*.

To model the complex interaction of light through atmospheric bodies such as clouds, Riley et al. (2004)[13] designed a lighting approximation based upon the scattering phase functions of air, clouds, ice, and rain. An isotropic phase function was used, $p(\omega_i \cdot \omega_o)$, dependent only on the angle between the incident and exitant directions. They were able to render phenomena that have only a single angular dependence like the 22° halo, but not more complex halos that rely on anisotropic scattering.

3.3 Rainbow Rendering

Rainbows are physical phenomena similar to the halos of interest. The halos investigated here occur due to ice crystals in the atmosphere, where as small water droplets are responsible for rainbows. Rainbows have been extensively studied, including in the field of computer graphics.

Kanamori et al. (2010)[14] computed a look up table of the Airy rainbow integral for a discretized set of scattered angles and light wavelengths, and used it to render idealized images of rainbows. The Airy rainbow integral is a less rigorous and faster to compute approximation of Lorenz-Mie scattering. One shortcoming of the Airy and Lorenz-Mie solutions is that they assume water droplets to be perfectly spherical. Sadeghi et al. (2012)[15] pre-computed a tabulated phase function for direction pairs and light wavelengths by simulating rays through non-spherical water droplets, taking into account not only dispersion but also interference and diffraction. Their results predicted rainbow behaviour for larger and less spherical droplets that could not have

been predicted by Lorenz-Mie.

The work done in this project follows the approach of pre-computing a tabulated phase function, by simulating rays through an ice crystal for numerous incident directions and light wavelengths.

3.4 Anisotropic Cloth Rendering

Prior to Jakob et al. (2010)[16] volumetric scattering in all rendering systems was limited to media whose properties were invariant under rotation. "Anisotropic scattering" was defined to be a dependence on the scattering angle between the incident and exitant directions, while assuming the medium itself was isotropic.

To render cloth, hair, skin, and translucent materials using a volumetric model, Jakob et al. derived an end-to-end formulation of physics based volume rendering of anisotropic scattering structures. This framework combined with their Micro-flake model which accounts for the anisotropic structure of materials lead to fascinatingly detailed renders of a cloth scarf[16].

Their work was implemented in the Mitsuba rendering system, allowing this project to rely on their framework for rendering anisotropic media.

Chapter 4

Recording Scattering Properties

To compute the phase function $p_\lambda(\omega_i, \omega_o)$, which provides a statistical description of the atmospheric ice crystal's light scattering properties, we simulate light rays incident on a given ice crystal, then record their outgoing direction. This approach is akin to simulating a gonioreflectometer, a device used to measure the *BDRFs* of real world materials.

4.1 A Virtual Gonioreflectometer

The geometry of the ice crystal is represented using a triangular mesh, the crystal's dimensions defined using real world measurements of atmospheric ice crystals [7]. In this project we focus mainly on hexagonal shaped crystals, which can be either plates or columns depending on their aspect ratio, Figure 4.1.

Ice crystal may either be near aligned or randomly oriented. Prior to simulating a ray, this orientation is computed using a transformation matrix drawn from a distribution of orientations. Instead of applying the transformation to all vertices of the triangle mesh representing the ice crystal, we apply the inverse transformation to the incident ray, and later apply the transformation to the outgoing ray.

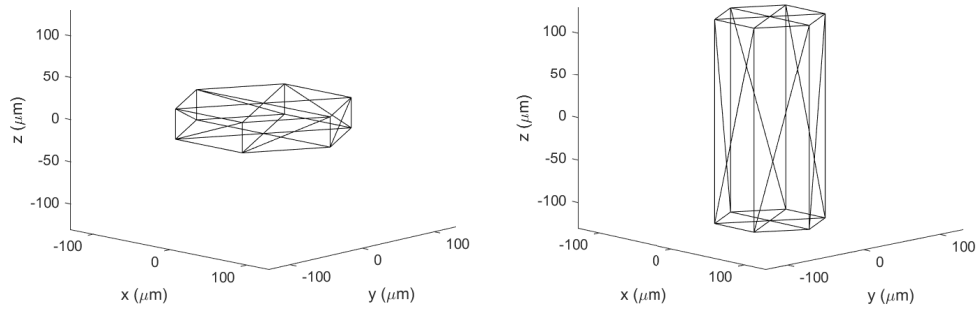

Figure 4.1: Generated triangle meshes for the Plate (left) and Column (right) ice crystals.

In terms of proper Euler angles, γ is sampled uniformly from $[0, 2\pi]$ (meaning the crystal is randomly oriented about its axis of symmetry), and β is sampled either uniformly from $[0, 2\pi]$ or from a narrow distribution such as $\mathcal{N}(0, \frac{\pi}{8})$ (the crystal's tilt $w.r.t.$ the vertical axis). The Euler angle α is sampled uniformly from $[0, 2\pi]$ when sampling the incident ray.

A ray direction incident on the origin (also the crystal center) is sampled. The ray origin is then sampled to be uniformly distributed over the projected area of the ice crystal's bounding sphere, Figure 4.2. This simplifies calculations, and rays which do not intersect the ice crystal will be flagged. The ray's wavelength is sampled from the visible light spectrum, which is used to determine the index of refraction used.

If the ray intersects the crystal it is then recursively traced until it exits, Figure 4.2. At each intersection with the crystal surface, the Fresnel Equations are solved, and a dice rolled to decide whether the ray transmits or reflects. Snell's Law is used to calculate the transmitted direction.

Finally, the ray's incident and outgoing directions, and its wavelength are integrated into the phase function being computed. As the ray origins are sampled from the projected area of a bounding sphere, we also use the sampled rays to compute the directionally varying scattering coefficient $\sigma_s(\omega)$.

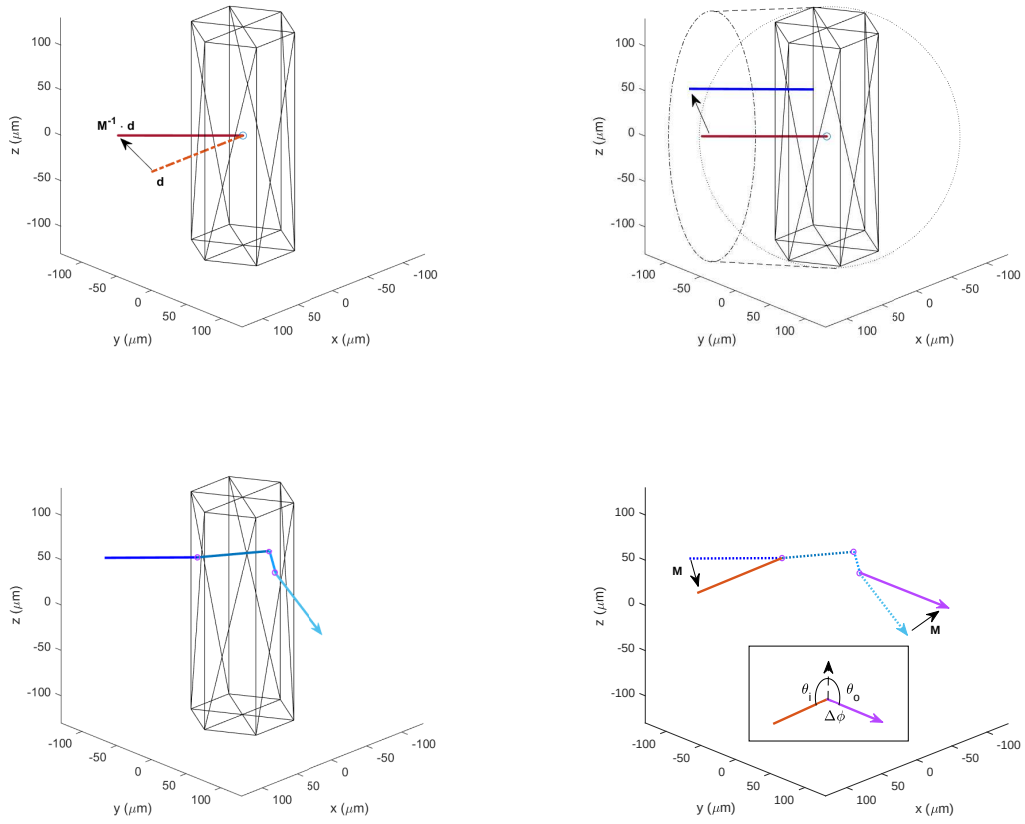

Figure 4.2: Illustration of the virtual gonioreflectometer.

Top left: Crystal is oriented by applying inverse transformation to sampled ray.
Top right: Ray origin is offset within crystal's projected bounding sphere.
Bottom left: Ray is recursively traced through the crystal following physical laws.
Bottom right: Incident and exitant ray directions are transformed to original coordinates, the output $[\theta_i, \theta_o, \Delta\phi]$ is computed.

4.2 Integrating the Phase Function

Following the approach of Sadeghi et al.[15], we compute the phase function by sampling rays and depositing them in a 4-dimensional regular grid. Each sampled ray returns the data $(\theta_i, \theta_o, \Delta\phi, \lambda, F_s)$, where θ_i is the incident zenith angle, θ_o is the exitant zenith angle, $\Delta\phi$ is the azimuthal angle difference between the incident and exitant ray, λ is the ray's wavelength, and F_s is a flag marking if the sampled ray scattered.

As many of sampled rays may not hit the ice crystal, or may transmit through with the same direction as the incident ray, we exclude these samples when integrating the phase function. This will mean that $p_\lambda(\omega, -\omega) = 0$, and this is accounted for by scaling the scattering coefficient $\sigma_s(\omega)$. Including those samples would result in a very peaked phase function, not suited for our tabulated format.

Each integrated sample contributes to the 8 vertices that enclose it in $(\theta_i, \theta_o, \Delta\phi)$, and it is binned in λ. The contribution weights $w_{ijk} : (i, j, k) \in \{+0, +1\}$ to each of the 8 vertices are linear:

$$t_{\theta_i} = (1 + cos(\theta_i))/2 \qquad t_{\theta_o} = (1 + cos(\theta_o))/2 \qquad t_{\Delta\phi} = |\Delta\phi|/\pi$$

$$i = \lfloor t_{\theta_i} \cdot (N_{\theta_i} - 1) \rfloor \qquad j = \lfloor t_{\theta_o} \cdot (N_{\theta_o} - 1) \rfloor \qquad k = \lfloor t_{\Delta\phi} \cdot (N_{\Delta\phi} - 1) \rfloor$$

$$w_i = 1 + i - (N_{\theta_i} - 1) \cdot t_{\theta_i} \qquad w_j = 1 + j - (N_{\theta_o} - 1) \cdot t_{\theta_o} \qquad w_k = 1 + k - (N_{\Delta\phi} - 1) \cdot t_{\Delta\phi}$$

$$w_{i+1} = 1 - w_i \qquad w_{j+1} = 1 - w_j \qquad w_{k+1} = 1 - w_k$$

$$w_{i,j,k} = w_i \cdot w_j \cdot w_k$$

Here t_x is the parameter x scaled to $[0, 1]$, N_x is the number of grid points for x's dimension, and i, j, k are the indices for $\theta_i, \theta_o, \Delta\phi$ respectively.

4.3 Phase Function File Format

The tabulated phase function is stored in a binary file, to be read into memory by Mitsuba just before rendering. The following specification is used to ensure compatibility between the two programs:

```
/* File Header */
uint32 nSpectrumSamples          // Number of spectrum samples
uint32 nThetaI                   // ThetaI tabulated resolution
uint32 nThetaO                   // ThetaO tabulated resolution
uint32 nDeltaPhi                 // nDeltaPhi tabulated resolution

/* File Data (arrays in row major format) */
float32[] phaseFunctionArray     // Size: nThetaI*nThetaO*nDeltaPhi
float32[] sigmaScaleArray        // Size: nThetaI

/* If nSpectrumSamples > 1 */
/* Size: nThetaI*nThetaO*nDeltaPhi*nSpectrumSamples */
float32[] spectralPhaseFunctionArrray
```

Chapter 5

Rendering Implementation

5.1 Implementation Overview

The Mitsuba rendering system was written to be extremely modular, consisting of a small set of libraries and over hundreds of smaller plugins that implement specific functionality. Adding a new plugin, such as a phase function, is as simple as inheriting from the abstract *PhaseFunction* base class and implementing a few functions. The new plugin can then be referenced in a scene's **XML** file, for example to be attached to a specific medium.

5.1.1 Extending the PhaseFunction Class

Mitsuba assumes phase functions and media scattering properties to not be wavelength dependent. Therefore to be able to model atmospheric phenomena such as ice crystal halos we must extend the rendering system.

We add three new virtual functions to the class, spectral analogs of existing functions, and one virtual boolean function to determine if a *PhaseFunction* instance is wavelength dependent:

```
/* Evaluates the phase function for a given pair of directions, returns
 * a spectrum of values. If not overloaded throws an error if called. */
virtual Spectrum spectralEval(
    const PhaseFunctionSamplingRecord &pRec) const;

/* Samples an exitant direction, returns the phase function
 * value as a spectrum over the sampling probability.
 * If not overloaded throws an error if called. */
virtual Spectrum spectralSample(PhaseFunctionSamplingRecord &pRec,
    Sampler *sampler) const;

/* Samples an exitant direction, returns the phase function
 * value as a spectrum over the sampling probability (pdf).
 * If not overloaded throws an error if called.*/
virtual Spectrum spectralSample(PhaseFunctionSamplingRecord &pRec,
    Float &pdf, Sampler *sampler) const;

/* Returns true if phase function is wavelength dependent */
virtual bool isSpectral() const;
```

To use the spectral analogs of the evaluation and sampling functions, extensions have to be made to Mitsuba's volumetric path tracer, specifically the *volpath_simple* plugin. These changes are very simple to implement, for example:

```
if (phase->isSpectral()) {
    Spectrum phaseVal = phase->spectralSample(pRec, rRec.sampler);
    if (phaseVal.isZero()) break;
    throughput *= phaseVal;
} else {
    Float phaseVal = phase->sample(pRec, rRec.sampler);
    if (phaseVal == 0) break;
    throughput *= phaseVal;
}
```

5.1.2 IceCrystalPhaseFunction Plugin

The *IceCrystalPhaseFunction* class implements all the functions declared in the *PhaseFunction* base class, including the aforementioned spectral analogs. During construction the phase function is read from a binary file, the file path is specified in the scene's **XML** file:

```
<phase type="icecrystal">
    <string name="filename" value="spectral_plate_pf.data"/>
</phase>
```

The file should follow the specification written in Section 4.3, an error is thrown otherwise. Some pre-processing is done, in particular $max\{p(\omega_i, \omega) : \omega \in S^2\}$ is calculated for each ω_i sample. The implementations of *IceCrystalPhaseFunction*'s evaluation and sampling functions are described in detail in Sections 5.2 and 5.3.

5.2 Evaluating the Phase Function

The computed phase function is evaluated using tri-cubic interpolation over the 3-dimensional regular grid. The 1-dimensional spline basis functions were solved for such that the cubic spline is given in terms of the 4 interpolating points, and its derivative at the two inner points is determined by central differencing:

$$s(x) = \sum_{i=-1}^{2} f_i \cdot \beta_i(x) \qquad x \in [0,1]$$

$$s(0) = f_0, \;\; s(1) = f_1, \;\; s'(0) = \frac{f_1 - f_{-1}}{2}, \;\; s'(1) = \frac{f_2 - f_0}{2}$$

$$\begin{aligned} \implies \beta_{-1}(t) &= -\frac{1}{2}t + t^2 - \frac{1}{2}t^3 \\ \beta_0(t) &= 1 - \frac{5}{2}t^2 + \frac{3}{2}t^3 \\ \beta_1(t) &= \frac{1}{2}t + 2t^2 - \frac{3}{2}t^3 \\ \beta_2(t) &= -\frac{1}{2}t^2 + \frac{1}{2}t^3 \end{aligned}$$

The tri-cubic interpolating spline is then a function of 64 neighbouring points in the grid and can be computed using a series of recursive calls to the 1-dimensional interpolating function for each dimension. This is inefficient however as it does involve repeated computation of the basis functions, so following the approach of [17], we can unravel the recursion into the following sum:

$$s(x,y,z) = \sum_{i,j,k=-1}^{2} f_{i,j,k} \cdot \beta_i(x)\beta_j(y)\beta_k(z) \tag{5.1}$$

5.3 Sampling the Phase Function

Sampling the phase function is done using the rejection sampling algorithm [18]. Rejection sampling provides a simple way to sample from a distribution without bias, at the cost of multiple sampling iterations, for this reason it is important that we are able to efficiently evaluate the phase function. The algorithm, when sampling ω_o uniformly, is as follows:

Algorithm 2 Rejection Sampling the Phase Function

Input: ω_i
Output: $\omega_o,\ \lambda$

$m \leftarrow max\{p(\omega_i, \omega) : \omega \in S^2\}$
repeat
  $\omega_o \sim \mathcal{U}(S^2)$
  $r \sim \mathcal{U}(0, 1)$
until $r \cdot m < p(\omega_i, \omega_o)$
$\lambda \leftarrow p_\lambda(\omega_i, \omega_o)\ /\ p(\omega_i, \omega_o)$

Note that the algorithm actually samples the monochromatic phase function $p(\omega_i, \omega_o)$, as this distribution is faster to evaluate. A spectrum λ is returned which is the value of $p_\lambda(\omega_i, \omega_o)$ over the value of the distribution from which ω_o is drawn. The volumetric path-tracer then accounts for this type of importance sampling.

Chapter 6

Results and Analysis

6.1 Qualitative Validation with Phenomena

To validate the correctness of our approach we perform qualitative comparisons of the resulting renders against photographs of the actual optical phenomena involved.

6.1.1 22° Halo

The rendered image of a 22° halo, Figure 6.2, was created by computing the phase function of randomly-oriented column shaped ice crystals. In addition to the 22° halo, sundogs are also visible as well as a faint 44° halo and the parhelic circle.
The photograph of a real 22° halo, Figure 6.1, also shows the sundogs with similar coloration and the parhelic circle. At the top of the circular halo a tangent arc is visible, which did not appear in the render. Tangent arcs are formed by oriented column crystals, explaining their absence.

The inner edge of the rendered 22° appears jagged, this is indicative that the phase function resolution is not fine enough, Figure 6.2.

Figure 6.1: Photograph of a 22° Halo. © Johan Klovsjö, reproduced with permission.

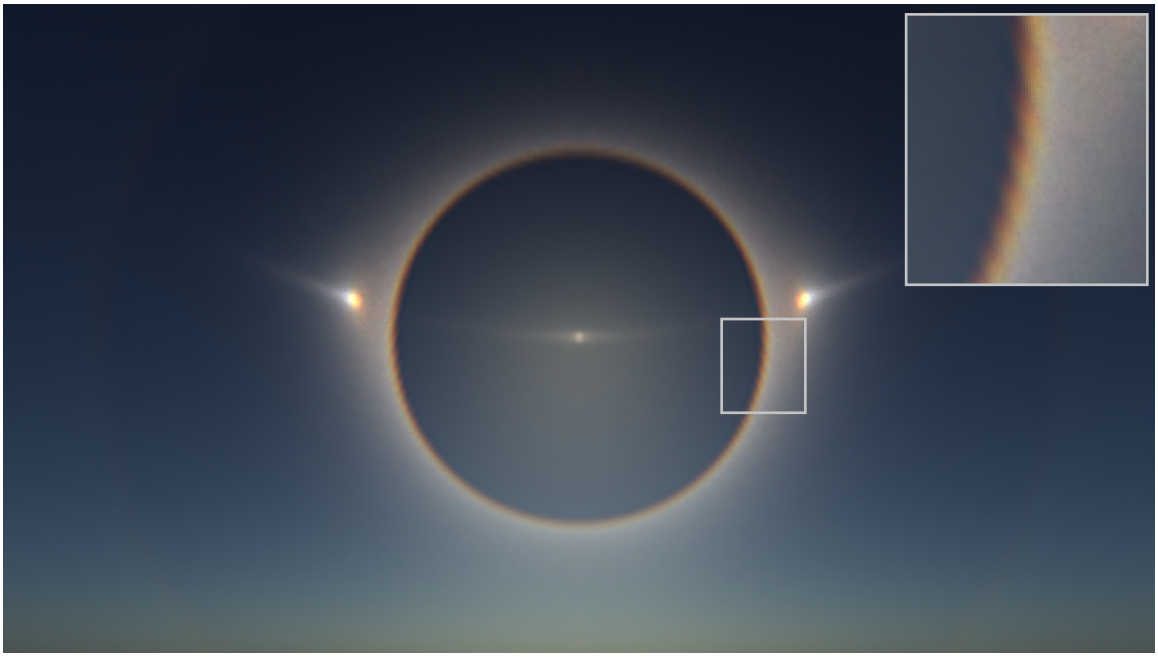

Figure 6.2: Rendered image (8192 samples per pixel) of a 22° Halo with environment lighting.

Table 6.1: Rendering parameters and performance.

	Crystal	PF Res.	PF Samples	Image Resolution	Samples per Pixel	Render Time	Avg. Rejection Sampling Iter.
Figure 6.2	Column	256^3x16	16e9	1280x720	8192	5202s	95.84
Figure 6.4	Plate	256^3x16	16e9	1280x720	8192	17108s	**556.05**
Figure 6.6	Plate	256^3x16	16e9	1280x720	8192	13945s	297.11

6.1.2 Parhelia

The rendered image of a parhelia, Figure 6.4, was created by computing the phase function of plate shaped ice crystals nearly aligned with the ground plane (Euler angle $\beta \sim \mathcal{N}(0, \frac{\pi}{8})$). In addition to the parhelia, three other structures are visible: an upper pillar, a lower pillar, and subparhelia (parhelia below the horizon line).

The photograph of a real parhelia, Figure 6.3, shows similar coloration on the sundogs. Other structures seen in the rendered image are not visible in the photograph.

6.1.3 Circumzenithal Arc

Nearly oriented plate shaped ice crystals are also responsible for circumzenithal arcs, a dramatic and colorful halo. Figure 6.6 shows a rendered image of a circumzenithal arc, as well as a rarer faint supralateral arc.

The photograph of a circumzenithal arc, Figure 6.5, shows similar structure and coloration.This particular image clearly demonstrates our approach's ability to model the phase function's wavelength dependency, given the resulting optical dispersion pattern.

Figure 6.3: Photograph of Parhelia. © Shelley O'Connell, reproduced with permission.

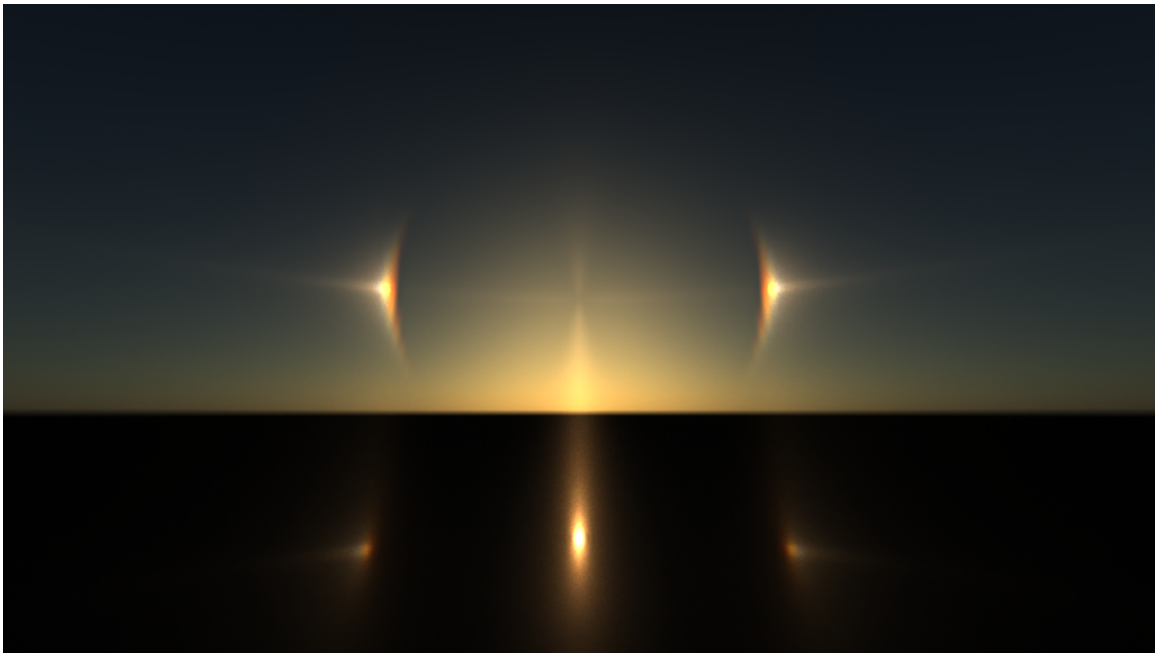

Figure 6.4: Rendered image (8192 samples per pixel) of a Parhelia (Sundog) with environment lighting.

Figure 6.5: Photograph of a Circumzenithal Arc. © Charlie Harvey.

Figure 6.6: Rendered image (8192 samples per pixel) of a Circumzenithal Arc with environment lighting.

6.2 Performance Results

All rendered images and performance results were run on an AMD Ryzen™ 5 1600 6-core CPU @ 3.57 GHz equipped with 16GB of memory.

6.2.1 Computing the Phase Function

The *virtual gonioreflectometer* described in Section 4.1 was initially written in MATLAB ® and then ported to C++ using MATLAB Coder™. The simulation parallelizes the tracing of rays through the target ice crystal.

To simulate 16 billion sampled rays it took 5.2 hours, approximately $1.17\mu s$ per sampled ray. This includes integrating the samples into the tabulated phase function.

6.2.2 Rendering Performance

There are multiple variables that can impact performance when rendering images, and improved performance can come at the price of lower image quality.

Rejection Sampling

Due to the specular nature of scattering in a hexagonal crystal, the phase function distribution tends to be concentrated around a few smaller areas over $\theta_o \times \Delta\phi$. Surface plots of the phase function are shown in Figure 6.7. In our implementation of rejection sampling, descibed in Section 5.3, samples are drawn uniformly from $\mathcal{U}(S^2)$. This results in a large number of rejected samples due to the peaked phase function distribution, leading to poor phase function sampling performance, seen on page Table 6.1.

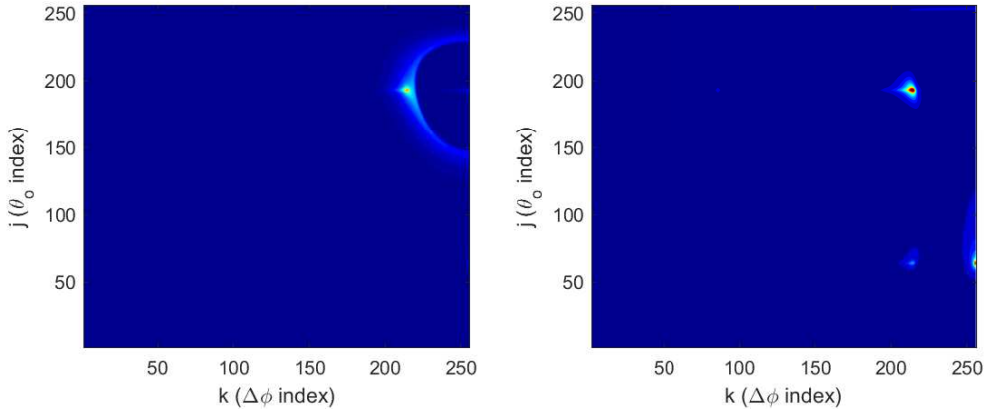

Figure 6.7: Surface plots of the phase functions, sliced at $\theta_i = \pi/4$. **Left**: Randomly-oriented column shaped ice crystal. **Right**: Nearly-oriented plate shaped ice crystal.

Phase Function Resolution

The tabulated phase functions used to render Figures 6.2 and 6.4 had a spatial resolution $(\theta_i, \theta_o, \Delta\phi)$ of $[256 \times 256 \times 256]$. Increasing the spatial resolution incurs a large space cost, however a sufficiently high resolution is required for acceptable quality, as shown in Figure 6.8.

	Phase Function Space	Render Time	Avg. Rejection Sampling Iter.
$64^3 \times 16$	17 MB	389s	45.5
$128^3 \times 16$	139 MB	521s	70.3
$256^3 \times 16$	$1,114$ MB	671s	95.8

Table 6.2: Space impact of various phase function resolutions, and render statistics for the images of Figure 6.8. Average number of rejection sampling iterations is lower for lower resolutions due to the phase function not being so peaked.

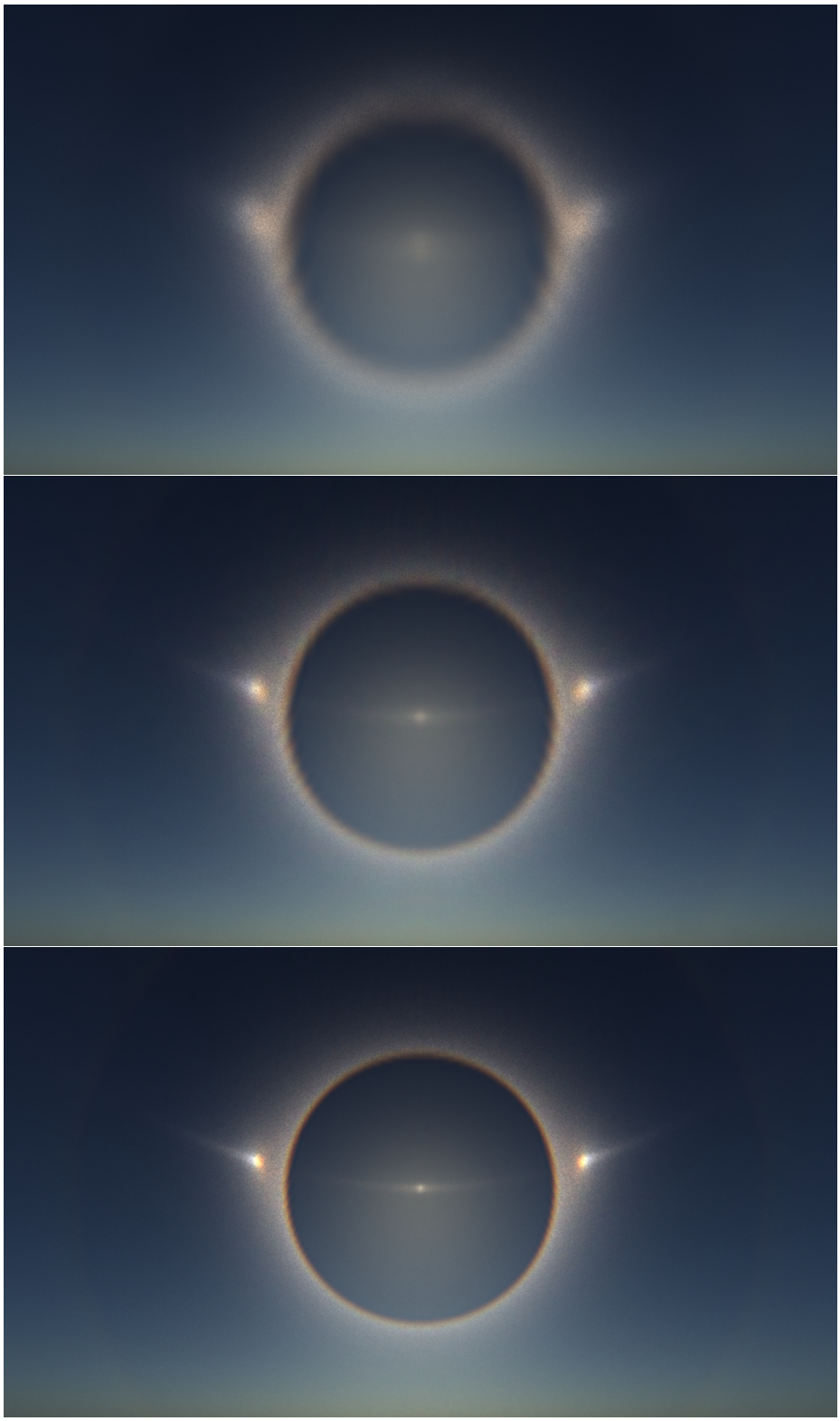

Figure 6.8: Renders of the 22° halo (1024 samples per pixel) with varying phase function resolutions, from top to bottom: $64^3 \times 16$, $128^3 \times 16$, and $256^3 \times 16$.

Chapter 7

Conclusion

The goal of this project was to answer if and how we can model volumetric scattering due to ice crystals, in a physically based renderer. To accomplish this goal we created a virtual gonioreflectometer, and used it to compute the anisotropic wavelength-dependent phase function of the ice crystals in question. This computation involved a Monte Carlo ray tracing simulation of the ice crystals, and integrating the resulting samples into a tabulated phase function.

We then implemented a plug-in in Mitsuba to evaluate and sample the computed phase function during rendering. The results were visually appealing images of various halo phenomena, and qualitative comparisons were made with real photographs of the corresponding halo validating our approach.

This project's implementation could in principle be added to artists toolboxes so they could incorporate these physically inspired effects into their work, however its performance is currently objectionable.

Rejection sampling of the phase function simply takes too many iterations, drastically increasing rendering times. Other approaches to sampling the phase function remain to be tried, and likely could offer better performance.

7.1 Future Work

As mentioned above, to lower rendering times sampling methods other than rejection sampling should be investigated. One standard method [1] would be to pre-compute a lower resolution marginal distribution function, as well as cumulative distribution function. These would then be used to importance sample the phase function.

In addition to finding better sampling strategies, better representations of the phase function should be sought. The tabulated format used here incurs a high space penalty, and is still unable to produce an acceptably circular 22° halo without a jagged inner edge. The phase functions shown here tend to be near zero except in relatively few areas, indicating an adaptive sparse grid such as one introduced by Pflüger (2010)[19] might be the better approach.

Other interpolation schemes in combination with a regular or sparse grid, beyond the tri-cubic interpolation used here, should also be investigated. Spectral methods, such as using a series of higher dimensional spherical harmonics to approximate the phase function, may work but would likely require a large number of terms and be costly to evaluate.

Last but not least, once these challenges are overcome, a plug-in should be written such that artists using a toolset such as Blender, in combination with a physically based renderer like Mitsuba, could have access to these stunning visual phenomena.

Bibliography

[1] M. Pharr, W. Jakob, and G. Humphreys, *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016.

[2] D. S. Immel, M. F. Cohen, and D. P. Greenberg, "A radiosity method for non-diffuse environments," in *Acm Siggraph Computer Graphics*, vol. 20, pp. 133–142, ACM, 1986.

[3] J. T. Kajiya, "The rendering equation," in *ACM Siggraph Computer Graphics*, vol. 20, pp. 143–150, ACM, 1986.

[4] E. Veach and L. J. Guibas, "Metropolis light transport," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pp. 65–76, ACM Press/Addison-Wesley Publishing Co., 1997.

[5] J. Wenzel, "Mitsuba renderer (2010)," *URL http://www. mitsuba-renderer. org*.

[6] R. G. Greenler and A. J. Mallmann, "Circumscribed halos," *Science*, vol. 176, no. 4031, pp. 128–131, 1972.

[7] J. Um, G. McFarquhar, Y. Hong, S. Lee, C. Jung, R. Lawson, and Q. Mo, "Dimensions and aspect ratios of natural ice crystals," *Atmos. Chem. Phys*, vol. 15, pp. 3933–3956, 2015.

[8] L. Bi and P. Yang, "Accurate simulation of the optical properties of atmospheric ice crystals with the invariant imbedding t-matrix method," *Journal of Quantitative Spectroscopy and Radiative Transfer*, vol. 138, pp. 17–35, 2014.

[9] A. Macke, J. Mueller, and E. Raschke, "Single scattering properties of atmospheric ice crystals," *Journal of the Atmospheric Sciences*, vol. 53, no. 19, pp. 2813–2825, 1996.

[10] A. Glassner, "Solar halos and sun dogs," *IEEE Computer Graphics and Applications*, vol. 16, no. 1, pp. 83–87, 1996.

[11] A. Glassner, "Computer-generated solar halos and sun dogs," *IEEE Computer Graphics and Applications*, vol. 16, no. 2, pp. 77–81, 1996.

[12] J.-C. Gonzato and S. Marchand, "Efficient simulation of halos for computer graphics," in *Proceedings of ECSIA2001, the 8-th European Congress for Stereology and Image Analysis*, p. 181, 2001.

[13] K. Riley, D. S. Ebert, M. Kraus, J. Tessendorf, and C. D. Hansen, "Efficient rendering of atmospheric phenomena.," *Rendering Techniques*, vol. 4, pp. 374–386, 2004.

[14] S. Kanamori, K. Fujiwara, T. Yoshinobu, B. Raytchev, T. Tamaki, and K. Kaneda, "Physically based rendering of rainbows under various atmospheric conditions," in *Computer Graphics and Applications (PG), 2010 18th Pacific Conference on*, pp. 39–45, IEEE, 2010.

[15] I. Sadeghi, A. Munoz, P. Laven, W. Jarosz, F. Seron, D. Gutierrez, and H. W. Jensen, "Physically-based simulation of rainbows," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 1, p. 3, 2012.

[16] W. Jakob, A. Arbree, J. T. Moon, K. Bala, and S. Marschner, "A radiative transfer framework for rendering materials with anisotropic structure," in *ACM Transactions on Graphics (TOG)*, vol. 29, p. 53, ACM, 2010.

[17] C. C. Lalescu, "Two hierarchies of spline interpolations. practical algorithms for multivariate higher order splines," *arXiv preprint arXiv:0905.3564*, 2009.

[18] G. Casella, C. P. Robert, M. T. Wells, *et al.*, "Generalized accept-reject sampling schemes," in *A Festschrift for Herman Rubin*, pp. 342–347, Institute of Mathematical Statistics, 2004.

[19] D. M. Pflüger, *Spatially adaptive sparse grids for high-dimensional problems*. PhD thesis, Technische Universität München, 2010.