FAST APPROXIMATION OF THE INVERSE REFLECTOR PROBLEM

FAST APPROXIMATION OF THE INVERSE REFLECTOR PROBLEM

Thesis

to obtain the degree of Master of Science in Computer Science
at Delft University of Technology,
to defend publicly on Thursday May 20, 2021 at 14:00

by

Zhoufan JIA

Thesis Committee:

Prof. Dr. Elmar Eisemann,	TU Delft / EEMCS / CS, chair
Dr. ir. Sicco Verwer,	TU Delft / EEMCS / CS
Dr. Ricardo Marroquim,	TU Delft / EEMCS / CS, supervisor
Dr. Markus Billeter,	University of Leeds, co-supervisor
Dr. Aurèle Adam,	TU Delft / Applied Sciences / Optics Group

Student number: 4787498

Contents

Summary

Suppose we have a target radiance distribution, a light source, and a plane for receiving light. How do we design a reflector that can give a similar result as the target radiance distribution? The inverse reflector problem is of high interest for light designers and related industries, such as lamp manufacture, headlight and street light design, and interior designs. There have been some researches on this specific topic. Nevertheless, the existing algorithms are either not fully compatible with parallel acceleration or have a narrow application scope (can only handle the far-field problem).

This thesis's goal is to design a method that can have a fast approximation of the inverse reflector problem and handle different application scenarios. The core of the proposed method in this thesis is a modified simulated annealing algorithm. I first give the definition of the inverse reflector problem and explain why I choose simulated annealing in the proposed method by analysing the related works and various optimisation algorithms. Then, I detail what issues the plain simulated annealing algorithm will have and why it cannot achieve satisfactory results. Moreover, to solve the issues and improve performance, I propose additional strategies, such as Phong tessellation, randomisation strategy, multi-level strategy, history-decision strategy and penalising strategy. I also discuss the system design based on the proposed method and how to accelerate the optimisation process in this system.

I evaluate and validate the proposed method by running test cases in different scenarios and compare the results with other algorithms. The results show that the method, as a general optimisation algorithm, can achieve good results in different application scenarios. The compatibility, speed and accuracy are the main advantages. The result of the proposed method can also serve as the initial guess for a finer optimisation.

PREFACE

My Master's study is a long journey full of happiness and sadness, and this thesis is the end of this journey. The year I spent in TU Delft for my Master's thesis is one of the most memorable and challenging periods in my life. I used to get lost during this journey, but with other people's help, now I am in the final phase of my Master's Thesis.

The last year witnessed many special things: COVID-19 pandemic, online learning, coronavirus lockdown, etc. But I can still continue my research project thanks to the help from kind people. I want to express my sincere gratitude to my supervisor Dr. Ricardo Marroquim, and co-supervisor Dr. Markus Billeter. During the research project, they continually give me valuable guidance and suggestions. They teach me how to conduct research, write a thesis, and work on a project in collaboration with others. Also, I would like to thank Dr. Aurèle Adam and Alex Heemels, who provide useful information in the field of optics. All in all, I want to say thanks to all the kind people who help me during this special period of time.

Zhoufan Jia
Delft, Apr. 2021

1 Introduction

This thesis is concerned with the problem of designing light sources that project a specific light pattern onto a surface, such as illustrated in Figure 1.1. More formally, we define the **inverse reflector problem** as: given a target radiance distribution, a light source and a receiving plane, how do we design a reflector to match the target radiance distribution on the plane?

This is a problem of high interest for the light designing and related industry, such as lamp manufacturing, headlight and street light design and interior designs. For example, a car's headlight can be designed to illuminate farther straight ahead, but less on the adjacent lane to avoid obfuscating the view of the cars coming in the opposite direction, as illustrated in Figure 1.2. Traditionally, an experienced designer models the reflector by iteratively modifying it until matching the target radiance distribution. This method requires extremely high experience, and its efficiency is very low as it is very time-consuming.

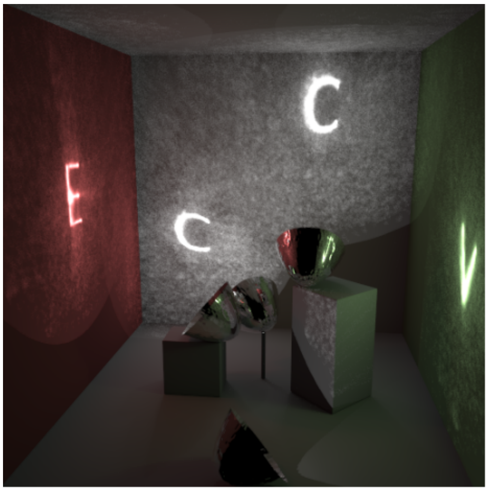
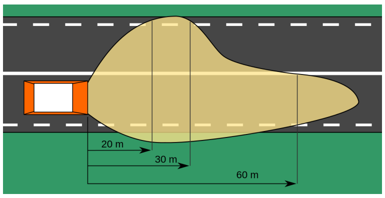


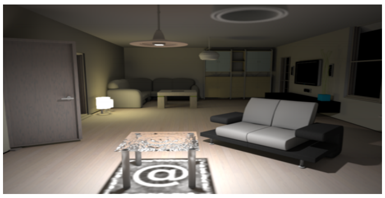
Figure 1.1: The designed reflectors can project specific light patterns onto the wall. Image reproduced from [1]

The motivation of our work is primarily the industrial need and the shortcoming of existing algorithms. The inverse reflector problem is a high-dimensional global optimisation problem, which is hard to be directly solved efficiently with existing algorithms. The goal of our method is to have a quick and decent approximation, that can then be used as input to more complex and slow solvers that require a good initial solution to converge properly.

Existing algorithms are either not fully compatible with parallel acceleration or have a narrow application scope. For example, some of them can only handle the far-field inverse reflector problem, in which the distance between the receiving plane and the reflector is assumed infinitely large. Some algorithms also restrict the incident light rays to be parallel and are not compatible with other kinds of light source. This simplification is not suitable for many real applications. Although some methods achieve impressive results for the restricted cases, they cannot be easily extended to more general and practical scenarios. In this work, we propose a general and efficient method to solve the inverse reflector problem that does not suffer from these limitations while still being able to achieve appealing results.



(a) A car headlight that illuminates a specific area. Image reproduced from [2]



(b) A design that projects an '@' symbol light pattern onto the floor using a lamp and refractive lens in the form of a center table. Image reproduced from [1]

Figure 1.2: The headlight and lamp can be designed for specific purposes by solving the inverse reflector problem

1.1. GENERAL PROBLEM FORMULATION

The inverse reflector problem can be abstracted into an optimisation problem with a scenario consisting of:

- the reflector
- the receiving plane
- the light source

The overview of the test scenario is shown in Figure 1.3. The light source can be of any type such as point light or directional, for instance. In addition, we assume that the reflector is perfect-mirror since most of the desired reflectors are designed to be mirror-like. Nevertheless, we will also show that our method is general enough to handle other types of materials such as refractive lenses. Furthermore, our method can also handle multi-reflectors and multi-light sources without any additional extension.

The inverse surface problem can be summarised as a optimisation problem:

$$\min_{S} Error(R_T, R)$$
$$where \quad \begin{cases} R = Trace(L, S) \\ h_{min} \leq H(S_{ij}) \leq h_{max} \end{cases}$$

- $Error(R_T, R)$: the distance between the target and current radiance distribution.
- $R = Trace(L, S)$: the radiance distribution R for given light source L and reflector S.
- $H(S_{ij})$: The *height* (offset relative to the initial position) of each vertex S_{ij} in reflector S must be within a specific range due to limitations of practical fabrication.

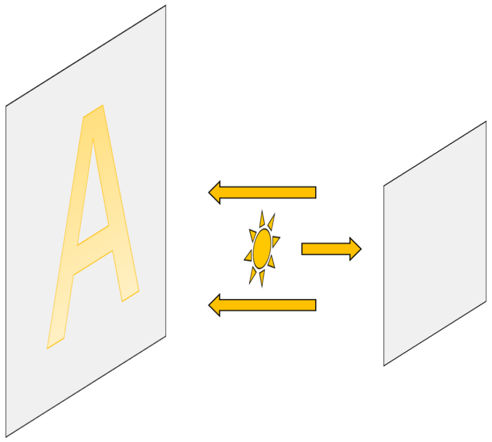


Figure 1.3: An illustration of the test scenario, with the receiving plane on the left, the light source in the middle and the reflector on the right.

(During optimisation, only the height of each vertex can be changed, which will be detailed in the following chapters)

- As an optimisation problem, there can be other constraints that are determined by users and application scenarios.

This is the high-level definition of the inverse reflector problem. The aim is to design the reflector S, which can minimise the distance between the target radiance distribution R_T and current radiance distribution R. The method should give a fast approximation of the inverse reflector problem within a given amount of time while ensuring the result quality. The final optimised mesh should also take into consideration the possibility to be fabricated by machines. It is much simpler to fabricate smooth surfaces or lenses than surfaces with high frequencies. Figure 1.4 shows an example with extreme discontinuities during optimisation. The result is almost impossible to be fabricated since most machines cannot mill the surface with such small concave angles. The strategies for ensuring surface smoothness will be detailed in the following chapters.

1.2. TECHNIQUES: CUDA AND OPTIX7

Since the goal is to achieve a fast approximation of the inverse reflector problem, we rely on the GPU to boost the performance of our method. At the same time, we aim to design a method that can leverage the GPU's parallel capabilities.

GPU is mainly used for graphics and rendering due to its specially designed architecture. Previously, there were many efforts to utilise the GPU's powerful parallel computing capability for general-purpose computing. There was, however, no general-purpose computing platform. One common method was to convert the computing problem into a form that OpenGL or DirectX can handle, such as matrix transformations. This method's low efficiency and narrow application scope are the main problems. The concept of GPGPU was proposed under such a background, which aims at utilising GPUs for

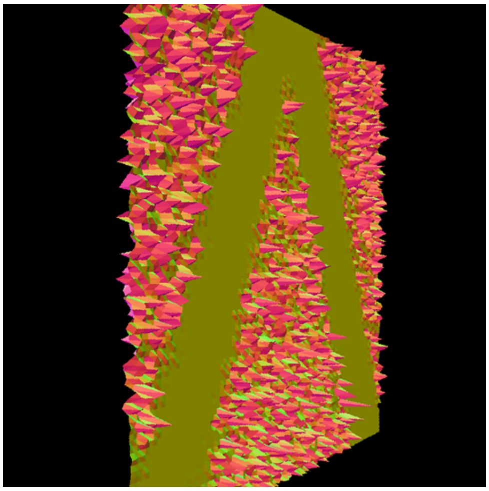


Figure 1.4: The reflector surface with high frequencies and extreme discontinuity is almost impossible to be fabricated by machines in practice.

general-purpose computing instead of only rendering.

The GPU architecture is continually updated in terms of design, memory, streaming processor. It is not realistic to design various programs for different GPU architectures. CUDA is a general-purpose parallel computing platform designed for GPUs. Programmers and researchers can ignore most of the hardware level details and still utilise the GPU's powerful computing capability. Compared with graphics APIs such as OpenGL and DirectX, CUDA is mainly designed for general-purpose computing instead of rendering, which means it is highly flexible and compatible with different computing purposes. CUDA-based programs can utilise GPUs, and GPUs can then provide a much higher FLOPS than CPU when guaranteeing high arithmetic intensity. Typical algorithms that CUDA can parallelise include reduce, scan, sort.

OptiX7 is a CUDA-centred ray tracing acceleration API. The core of OptiX7 is to accelerate the traversal of the scene, which aims to compute the intersection between the ray and scene objects. Despite giving built-in methods like instancing and primitive intersection, OptiX7 is still highly flexible. It provides explicit control for memory management, multiple GPUs, acceleration structure builds, shader binding table, custom primitive and intersection, making it possible to be integrated into most application scenarios.

Since Optix7 is CUDA-centred, CUDA's concepts and programming rules are still applicable when integrating OptiX7 into a CUDA project. OptiX7 and CUDA share the same API for memory management and device pointer. OptiX7 significantly accelerates the acceleration structure building and scene traversal, making programmers and researchers focus more on the algorithm itself instead of the performance. DXR and Vulkan can be candidates for ray tracing and also provide similar features. I chose OptiX7 to implement the system since the developer overhead brought by integrating it to the C++/CUDA based project with an Nvidia graphics card is small.

2 Related work and background

2.1. RELATED WORK

The inverse reflector problem has been a research field of high interest for a long time. One of the first methods for inverse reflector design is proposed by Patow et al. [3]. They solved the problem with a modified brute-force search algorithm. However, they simplify the problem scenario and focus more on far-field, in which the distance to the receiving plane is assumed to be infinitely large, as shown in Figure 2.1. Since their proposed algorithm relies on brute-force search, there is no straightforward way to narrow the high-dimensional search space during optimisation, which makes the complexity extremely high. The high computational cost and restriction to the far-field scenario renders their method inapplicable to many industrial scenarios.

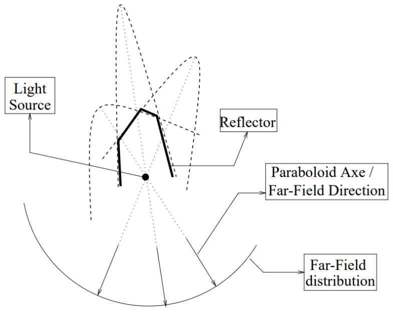


Figure 2.1: The distance between light source and receiving plane is infinitely large in far-field inverse reflector problem. Image reproduced from [3]

The inverse reflector problem can be abstracted as a non-convex optimisation problem in a high dimensional search space. Finckh et al. [1] proposed a method based on a general optimisation method, namely Simultaneous Perturbation Stochastic Approximation (SPSA) [4], and a spline-based surface representation, as shown in Figure 2.2. SPSA is a typical gradient-based stochastic optimisation algorithm proven to be robust and stable and to converge to a global minimum in high-dimensional cases. The gradients at the control points are used to guide the stochastic optimisation direction. SPSA requires a number of ray tracing simulations for the objective (error) function evaluation during the optimisation process. Generally, the Monte Carlo method is necessary for simulating different distributions of the incident light rays and their interaction with surfaces. For example, the light source might emit radiance with respect to specific distributions. However, the "gradient-based" algorithm requires a high sample-rate ray tracing to reduce the variance of the Monte Carlo, otherwise it would result in inaccurate gradients. Moreover, the ray-intersection computation for spline-based surfaces brings significant overhead.

The recently proposed differentiable renderer Mitsuba 2 [5] solved the inverse reflector and caustics design problem with gradient descent by differentiable rendering, in which the differentiable rendering can preserve the accurate gradient information. But compared with the traditional ray tracing techniques, the performance (speed and GPU memory usage) of differentiable rendering can be the issue. Also, gradient descent, as a local optimisation algorithm, cannot approximate the global optimum in many cases.

Weyrich et al. [7] solved this problem from a different perspective. In their method,

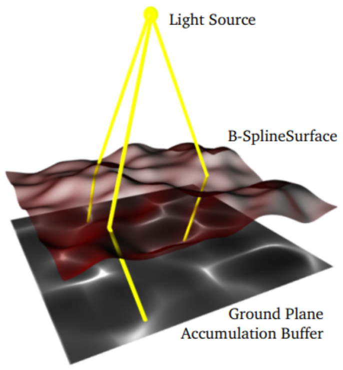


Figure 2.2: The SPSA-based method needs to compute the intersection of ray and B-spline surface. Image reproduced from [1]

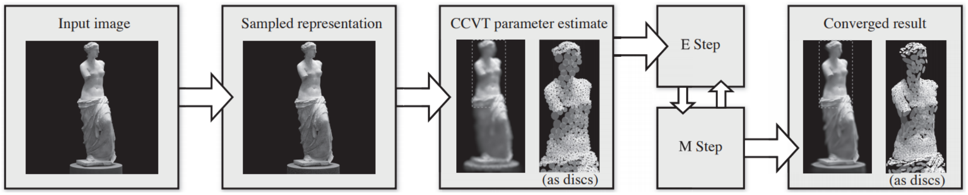


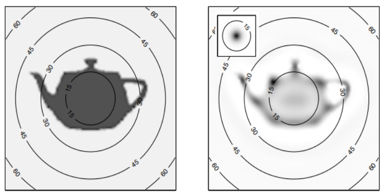
Figure 2.3: The Gaussian decomposition of the target radiance distribution. Image reproduced from [6]

the radiance distribution is abstracted as a reflectance function, from which it is possible to infer the required distribution of microfacets slope. It first uses the microfacets slope distribution to generate unconnected microfacets and then uses simulated annealing to optimise the energy (error) and smooth the surface, as shown in Figure 2.4.

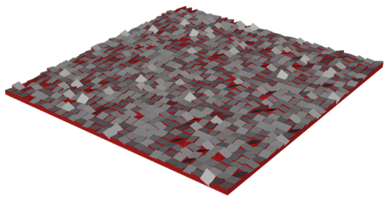
Inspired by the idea of a one-to-one mapping between microfacet and radiance, Papas et al. [6] proposed to first decompose the target radiance distribution into Gaussian radiance kernels, as shown in Figure 2.3, and designed a surface patch for each of the Gaussian radiance kernels. It then optimises the patches' permutation until achieving a similar result to the target radiance distribution. Their algorithm can provide a very accurate result, especially when handling complex radiance distributions, such as the Lenna image. However, the main limitation is that it can only solve the problem in scenarios with a collimated light source and each patch should be designed under the same conditions. Every patch needs to be ensured to receive exactly the same incident light rays because they will be permuted in further optimisation. Also, the complexity of their method increases exponentially with the number of Gaussian kernels in the decomposition. Their algorithm does not follow an "evaluation-update-evaluation" way of optimisation, so modern ray tracing techniques cannot accelerate the evaluation/simulation part in their algorithm's pipeline.

For the collimated light source cases, Schwartzburg et al. [8] proposed a method based on optimal transport. Their method computes the desired normal distribution by optimal transport between the initial radiance distribution and target radiance distribution. Then, it optimises the refractive surface according to the optimal transport result and generates a surface that has similar normal distribution as the desired normal distribution. Yue et al. [9] proposed a method that computes the mapping between incident light rays distribution and desired radiance distribution by solving a sequence of Poisson equations. Both the optimal transport and Poisson based algorithms can achieve accurate results when finally converging.

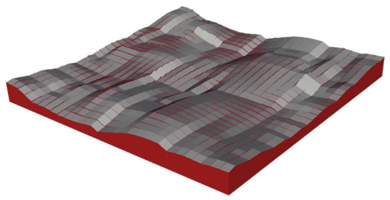
Although achieving accurate results, the optimal transport and Poisson based algorithms still have significant issues like low generality and compatibility. This kind of algorithms can only be applied to cases with a refractive lens and collimated light source, and additional pre-assumptions are needed for the algorithm to work. For example, interreflection is not taken into account in algorithms based on optimal transport or Poisson equation, since the mapping computation cannot infer the possible interreflection and self-occlusion.



(a) Target radiance distribution and deconvolved microfacet distribution



(b) Generate surface according to microfacets distribution



(c) Surface after optimisation on microfacets

Figure 2.4: Reflector design by microgeometry fabrication. Image reproduced from [7]

2.2. CHOICE OF OPTIMISATION METHODS

For the inverse reflector problem, the choice of optimisation methods depends on the problem characteristics. Various optimisation methods have different features and are not appropriate for all kinds of optimisation problems. In this section, the problem characteristics will be detailed, and the choice of the optimisation method will be justified.

2.2.1. PROBLEM CHARACTERISTICS AND GENERAL IDEAS

The characteristics of the problem indicate which kind of optimisation method is more appropriate. There are three main characteristics of the inverse reflector problem:

- Search space is continuous in high dimensionality

- Global minimum is surrounded by many local minimums
- Small error generally means closer to the global minimum in a global view

Firstly, the optimisation problem is in a continuous search space in high dimensionality. This characteristic makes it almost impossible to solve the problem with an analytic solution, in which case heuristic-based algorithms should be considered. This problem is also not a convex optimisation, and its global minimum is surrounded by many local minimums, which means it is necessary to apply global optimisation methods instead of local ones.

There are two main ideas for solving the inverse reflector problem. The first idea is to directly analyze the radiance distribution and design the corresponding surface patches, such as the algorithm proposed by Papas et al. [6]. The second idea is the analysis-by-synthesis method, which is to iteratively update the mesh and optimise until the mesh can lead to a similar result as the target radiance distribution.

In optics, one common way to solve the inverse reflector problem is to analyse the radiance distribution and then extract the useful information (direction, slope, normal, BRDF) to build the desired surface. The most widely used techniques include Monge-Ampère solver [10] and tackling the optimal transport problem. The advantage is that these methods can give an accurate result when they are able to converge. But the main shortcoming is that such an optimisation algorithm requires a good initial guess for convergence, and its process is a black-box that is hard for users to control and tune optimisation. Compared with analysis-by-synthesis optimisation methods, this kind of algorithm does not rely on the per-iteration ray tracing simulation. In the cases where fast approximations and process control are required, this type of algorithm lacks explicit control on the optimisation process. Moreover, this class of algorithms, such as the algorithm proposed by Schwartzburg et al. [8], normally has more constraints on the test scenario since it is not based on a general optimisation method. In brief, the main issues are the requirement for a good initial guess and the lack of both generality and explicit process control.

The second class of algorithms is based on general optimisation methods. The iterative local/global optimisation method includes gradient descent, quasi-Newton method, genetic algorithm, simulated annealing, and other stochastic/continuous or gradient-based algorithms, such as Simultaneous Perturbation Stochastic Approximation (SPSA) [4]. Our project focuses on this class of algorithms for a few reasons. First, they have been widely used and proven to be effective and robust in various optimisation contexts, especially in such high-dimension optimisation problems. Moreover, it is more general than the first class of algorithms, being able to handle more diverse scenarios. Second, the iterative analysis-by-synthesis process also makes it possible to gain significant performance improvement in the ray tracing simulation part with GPU-based implementations.

2.2.2. COMPARISON OF GLOBAL OPTIMISATION METHODS

There are several candidates for the global optimisation method, including gradient-based optimisation method, genetics algorithms, Tabu search, and simulated annealing. In this section, the candidates will be compared with each other, and the choice of the optimisation method for the inverse reflector problem will be reasoned.

GRADIENT-BASED OPTIMISATION METHOD

The gradient is thought to be a good way to guide the direction of the optimisation process, especially when handling high-dimensional optimisation problems. Simultaneous Perturbation Stochastic Approximation (SPSA) is a typical gradient-based stochastic optimisation algorithm, which is proven to be robust and stable and to converge to a global minimum in high-dimensional cases. Albeit such good attributes, the "gradient-based" characteristic can bring issues related to the variance of Monte Carlo ray tracing that cannot preserve accurate gradient information. A high sample rate is not practical for a performance-oriented system. Also, the choice of SPSA parameters is more based on experience and convention than mathematics proof and intuitive understanding [4], which makes it hard to tune it to order for it to be compatible with different test scenarios and cases. Generally speaking, most of the gradient-based methods share the same disadvantages as SPSA. Although the results are not necessarily bad, we still decide to avoid gradient-based methods as candidates for solving the inverse reflector problem since we target high performance.

GENETIC ALGORITHMS AND TABU SEARCH

Typically, for the high-dimensional global optimisation it is not possible to directly find an accurate analytic solution. Due to this nature of the optimisation problem, we mainly focus on heuristics and metaheuristics global optimisation, aiming to find a feasible solution within limited time and memory. Genetic algorithm and Tabu search are both candidates for global heuristic optimisation. As shown in Figure 2.5, they are algorithms which require a population to be allocated in memory.

Genetic algorithms [11] are a typical choice for global optimisation, which is one of the candidates for the proposed method in this thesis. The main idea is to simulate the natural selection process. The initial population containing a set of individuals (solutions encoded as chromosome and genes) will evolve by crossover and mutation until the population converges and becomes stable. Genetic algorithms can scale well to high dimensional problems. For the inverse reflector problem, it is trivial to encode surface representations, such as heightmaps and triangle meshes, into a chromosome string and then crossover and mutate. Nevertheless, the size of the population needs to be extremely large in order to have enough span over the whole search space. The crossover and mutation across the whole population may bring performance issues. Moreover, due to GPU memory limitation, storing a population with enough individuals for evolution is also a challenge. Currently, for test cases in this thesis, the GPU memory limitation is not an issue. However, when more complex scenarios are taken into account in practical applications, this issue can be difficult to be solved.

The Tabu search algorithm [12] is another widely-used heuristics global optimisation algorithm, which uses a table recording the previous state (movement) to ensure that the optimisation process will not go backwards and get stuck into a local minimum. The size of the table heavily depends on the complexity of the optimisation problem. For such an optimisation problem in a continuous high-dimensional search space, the search space needs to be first discretised to be further processed. Also, the table size needs to be extremely large to ensure that it records enough states to which the optimiser should not go back, since it might be stuck to the local minimum if the table size is not large enough. All in all, the inverse reflector problem's characteristics such as "continuous" and "high-dimensional" makes it very challenging to be optimised with such algorithms which need a state record table.

SIMULATED ANNEALING

Simulated annealing [13] is our choice for the optimisation method. It is a probabilistic method for approximating the global optimum within a given amount of time, appropriate for cases where a feasible solution and convergence speed matter more than an accurate solution. The main idea is to simulate the annealing in metallurgy. The atoms will move drastically when the temperature is high and gradually converge to a stable state in each cooling schedule temperature. The energy will converge to a minimum together with the temperature in the cooling schedule. The way of escaping from the local minimum in simulated annealing is by the judge function. Like the annealing in metallurgy, the optimiser will have a high probability of accepting a random guess in the neighbour solution space when the temperature is high, which makes it possible to escape from the local minimum. The energy will gradually converge to a minimum when the temperature is decreasing subject to the cooling schedule.

Simulated annealing has several useful attributes. It is well scalable with the dimensions. As shown in Figure 2.5, simulated annealing is a no-memory heuristics optimisation algorithm, which means it does not require large memory space for storing a table as the genetic algorithm does. In each iteration, simulated annealing does not require as many objective (error) function evaluations as the genetic algorithm does. Furthermore, simulated annealing's compatibility is high since it only requires the definition of the energy function and the solution space, without any additional requirements such as discrete space, Markov chain model and complex solution encoding. As a heuristics algorithm, simulated annealing is suitable for those problems which require a feasible solution within a given amount of time. Our proposed algorithm for the inverse reflector problem is a simulated annealing-centred algorithm, with many other optimisation strategies. In the next chapter, our algorithm and strategy will be explained in detail.

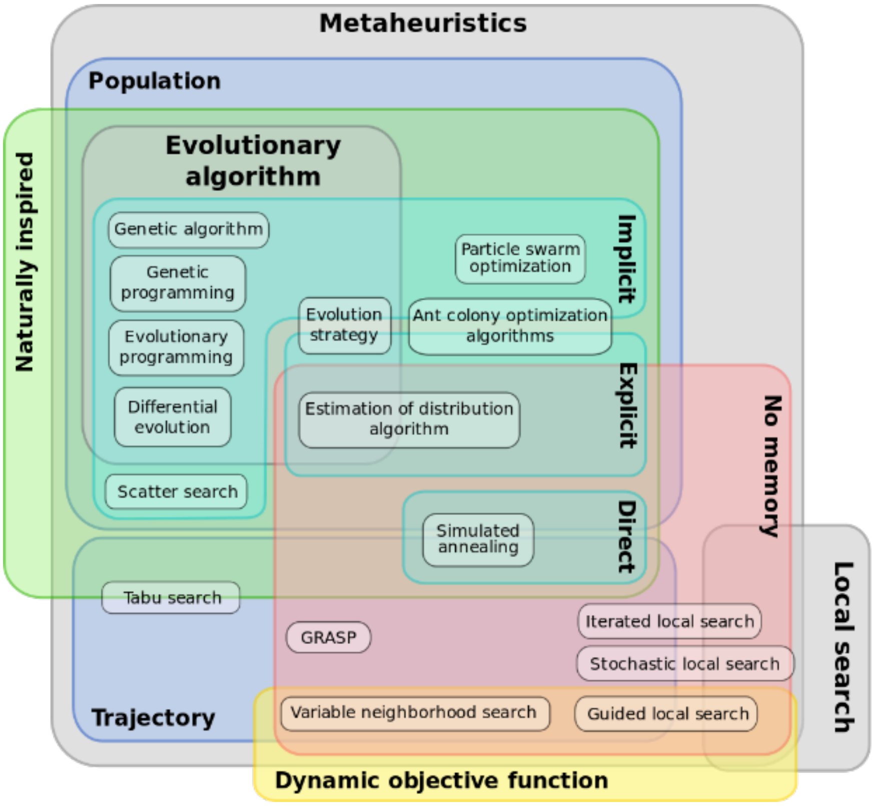


Figure 2.5: The category of heuristics algorithms [14]

3

General simulated annealing

Our proposed system is based on a general optimisation method, simulated annealing, with many additional strategies and methods. This chapter will describe how general simulated annealing works in the inverse reflector problem without any additional strategies. We will then discuss why the plain simulated annealing approach is not able to achieve satisfactory results, and discuss the proposed improvements in the following chapter.

3.1. GENERAL SIMULATED ANNEALING PIPELINE

The key idea of simulated annealing is to generate a new solution in the neighbour search space at each iteration and accept the modification with a probability. Unlike other local optimisation methods such as gradient-descent, it allows us to go in the wrong direction with a probability, giving it the ability to escape from local minima.

An overview of the simulated annealing algorithm is shown in Figure 3.1. For each iteration, the optimiser generates a neighbour solution based on the current solution and evaluates it. If the error has decreased the new solution is accepted. Otherwise, the optimiser can still accept it within a certain probability related to the current temperature. The probability function is defined as:

$$P(e, e^{'}, T) = D(x) = \begin{cases} 1, & \text{if } e^{'} < e; \\ exp(-(e^{'} - e)/T), & \text{otherwise} \end{cases} \tag{3.1}$$

where e is the current energy, $e^{'}$ is the energy of the new solution, and T is the temperature. Moreover, after each iteration the temperature decreases according to the following cooling schedule:

$$T_{new} = a * T_{current} \tag{3.2}$$

where a is the cooling schedule factor under the condition $a < 1.0$. Typically, a is set to a value around 0.95, and multiple iterations are conducted for each temperature in the cooling schedule.

As shown in Equation (3.1), if the new solution results in a higher error the acceptance probability is computed with the energy difference and temperature. Normally, the higher the temperature the higher the tolerance for a worse solution. After enough iterations the optimisation converges to a local or global optimum.

There are two crucial parts: the objective function evaluation, which in our case needs ray tracing simulation; and the neighbour solution generation, which needs an explicit surface representation definition. Both directly affect the optimisation performance.

The plain simulated annealing is a single-pass process since the temperature only decreases according to the cooling schedule and the optimisation finally converges when the temperature is low enough. As the inverse reflector problem is a high-dimensional, complex and non-convex optimisation problem, single-pass simulated annealing usually cannot reach the global minimum since it might not be able to escape from the local minima when it is far from the global optimum and the temperature is low. One standard method is setting a low-speed cooling schedule, in which the temperature will decrease slowly, but it will significantly decrease the performance, which means the convergence speed is low and the number of iterations needed for a good approximation is high, and cannot achieve a good approximation in the early phase.

To find the balance between quality and performance, we apply a re-heat strategy, turning it into a multi-pass process. In this case, the optimiser starts with a high initial temperature. If it cannot find a better solution within N iterations, the optimiser will re-heat with a smaller initial temperature $init_{new} = \beta * init_{current}$ and then continue the optimisation, as shown in Figure 3.2. The reason for using a smaller initial temperature is that after several passes the current error is already relatively low and the optimisation does not require the same initial temperature as before. Furthermore, a high initial temperature as before also slows the convergence process. This strategy can help the optimiser converge to a feasible solution in the early phase and finally converge to an accurate solution if it runs for enough iterations.

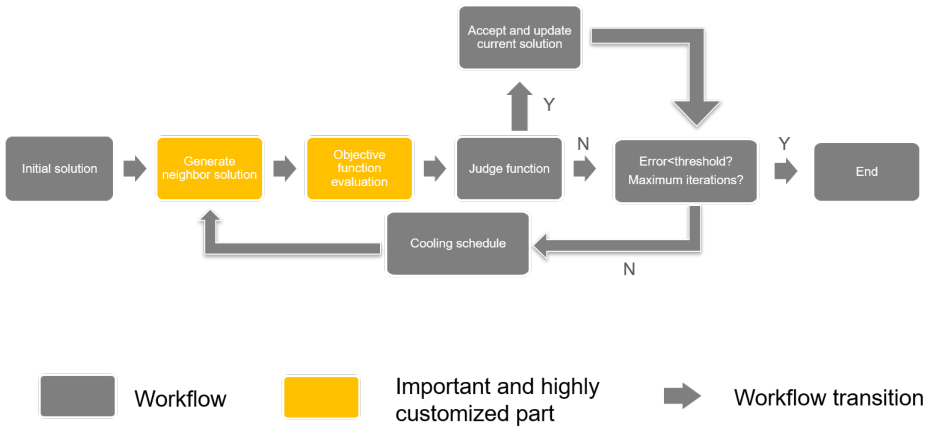

Accept and update current solution
Y
Initial solution
Generate neighbor solution
Objective function evaluation
Judge function
N
Error<threshold? Maximum iterations?
Y
End
Cooling schedule
N
Workflow
Important and highly customized part
Workflow transition

Figure 3.1: The pipeline of general simulated annealing

3.2. OBJECTIVE FUNCTION

The objective function is what the simulated annealing aims to optimise. For the inverse reflector problem, the objective function can be defined as the distance between the current and target radiance distribution. We will first define how we represent the radiance distribution followed by details on the radiance error function.

3.2.1. RADIANCE DISTRIBUTION REPRESENTATION

To obtain the radiance distribution on the receiving plane, the optimiser needs to do the ray tracing simulation in the test scenario Figure 1.3. The radiance distribution is the result of the ray tracing simulation. It is highly related to the objective function definition and evaluation. Ideally, the radiance distribution should be discrete to be processed in GPU. In addition, radiance intensity in the distribution representation should also comply with the physical definition, which means there is a positive correlation between received radiance and the radiance intensity in the definition. An explicit discrete radiance distribution representation is necessary for the objective function evaluation.

Intuitively, the radiance intensity depends on how much radiance each unit surface

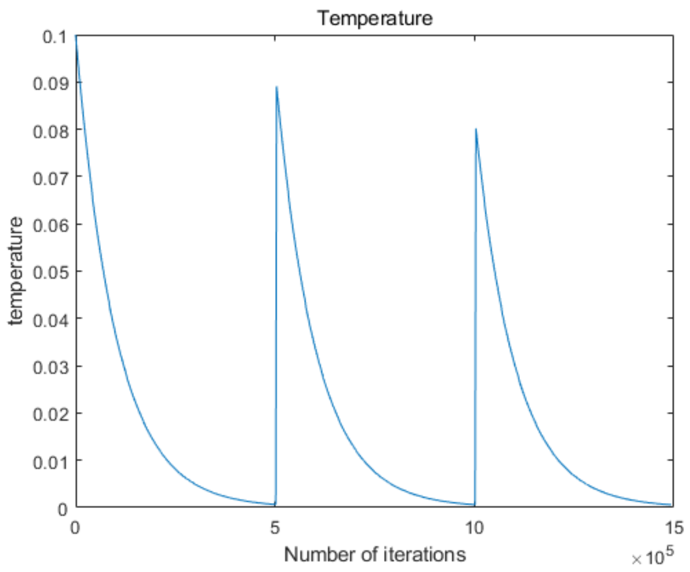


Figure 3.2: Re-heat temperature strategy: when the optimiser is unable to decrease the error after a number of iterations it re-heats by raising the temperature. For each re-heat pass the initial temperature is lower than the previous pass.

receives. Since the simulation generates light rays and traces them, the natural way to evaluate the radiance intensity is to record how many light rays hit each unit area. We define the sum of the intensity of all light rays that hit the unit surface as the radiance intensity. For the radiance distribution definition, the receiving plane is discretised by a regular subdivision, in which all cells have the same size, as shown in Figure 3.3. The number of cells depends on the resolution of the radiance distribution. High resolution will need more cells and vice versa. During ray tracing, the number of light rays that hit each cell is recorded. After the ray tracing simulation, every cell is mapped to a pixel in the radiance distribution in order to evaluate the objective function.

3.2.2. RADIANCE ERROR FUNCTION

The evaluation of the radiance error function measures the distance between the current radiance representation and target radiance representation. Intuitively, the objective function we want to minimise is the difference between target and current result, which is also called the error function. There are different ways to define an error function. The most widely-used ones include Root Mean Square Error (RMSE) and Mean Average Error (MAE). In our project, we choose MAE as the error function. The reasons are:

- the MAE error function is easy to be normalised in the range $[0,1]$ given that we know the total number of generated light rays. The normalised values comply more with the human's visual system and are more easily interpreted compared with RMSE.

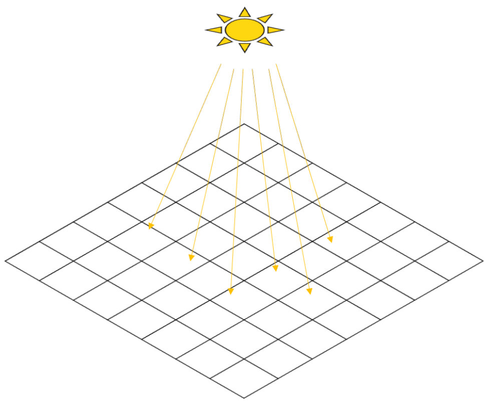


Figure 3.3: The discrete radiance distribution is represented by the number of light ray that hits each cell.

- With RMSE, the outliers' error has more influence since every element will be squared before averaged. In contrast, for the inverse reflector problem, the outliers should not contribute too much to the final error function since we mainly focus on the overall quality (of the radiance pattern/shape). An appropriate tolerance for outliers is needed as a small number of outliers do not affect the overall quality much.

The natural way for computing error is to convert radiance distributions to normalised grey-level images, and then compute the error between them. Nevertheless, the original radiance intensity value is lost after normalisation. A critical point is that the radiance distribution used in the objective function is not normalised to grey levels (0-255). Instead, it directly uses the count of ray hits inside each cell as the definition of radiance distribution. As shown in Figure 3.4, images in the same radiance pattern might be in totally different radiance intensity. For the inverse reflector problem, the radiance distribution with the same pattern is not sufficient for ensuring high-quality results since the radiance intensity should also be considered. If we measure the error with normalised grey levels, we will lose the radiance intensity information during optimisation. The normalised grey level can be easily disturbed by the maximum and minimum value, especially for Monte Carlo ray tracing without a high sampling rate. A single outlier that is above the maximum or below the minimum values affects the entire images' grey levels, i.e., almost all values are modified when normalised. Fluctuations in the values are undesired during the optimisation as it cannot approach the global optimum in a stable manner. Due to the reasons above, even if the input target radiance distribution is an 8-bit grey level image, our system still needs to normalise it by the total number of generated light rays and convert "grey levels" to "radiance intensity". In our proposed method, the normalised grey-level images are only used for display purpose.

Although we do not normalise the hit number to grey level during the MAE error computation, the final error value normalisation is still necessary for the optimisation process. The error value will be used in the judge function to decide whether the current solution should be accepted or rejected, which is the critical part of simulated annealing.

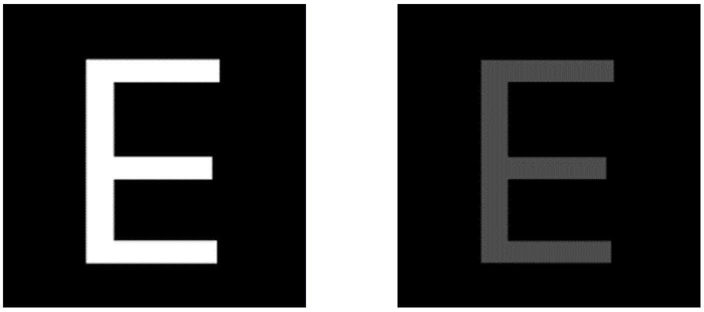

(a) High radiance intensity (b) Low radiance intensity

Figure 3.4: Images with the same radiance pattern but totally different radiance intensity

It is also related to the temperature, cooling schedule, and acceptance probability function. A normalised error in the range $[0,1]$ renders the choice of optimisation parameters much simpler, as will be detailed in the next chapter.

Assuming the total number of light rays is N in the ray tracing simulation, the largest possible error is then $2N$, in which case there is no overlap of radiance between target and result. The smallest error is naturally 0, in which case the target and result radiance distribution are exactly the same. In each iteration during the optimisation, the objective function (error) will always be normalised from $[0,2N]$ to $[0,1]$.

According to the analysis above, the formal definition of the L1-norm Error Function with respect to the radiance distribution R is:

$$Error_R(R_T, R) = (\sum_{ij} \left|R(i,j) - R_T(i,j)\right|)/(2N) \tag{3.3}$$

where R and R_T are, respectively, the current and target radiance distributions and $2N$ is the normalisation factor.

3.3. GENERATING NEIGHBOR SOLUTION WITH MESH UPDATES

The definition of surface representation is necessary for generating a neighbour solution. The choice of surface representation is related to the optimisation convergence, ray tracing simulation and performance. We consider two mains candidates for surface representation:

- spline-based meshes (such as NURBS)
- triangle meshes generated from the heightmaps

Spline-based representations, such as NURBS, is commonly used to generate smooth meshes, in which the control points define the mesh's geometry. The smoothness comes from the use of polynomials in its definition, that also guarantees C^1 or higher continuity. The problem of spline-based meshes is, however, performance. The polynomial definition of the geometry imposes extra challenges for ray tracing. There are two common ways to render spline-based mesh: *tessellation before ray tracing* and *direct intersection using Newton method*. Nevertheless, both are significantly costly in terms of performance. Furthermore, when creating acceleration structures for ray tracing, the bounding volumes for spline patches are also more challenging to define. Finally, for optimisation with a complex target radiance distribution, the spline-based mesh cannot be flexible enough to capture all details since it cannot be arbitrarily deformed and needs to comply with the polynomial surface definition.

On the other hand, triangle meshes can be generated from heightmaps, as shown in Figure 3.5. Compared with spline-based meshes, the advantage mainly lies in performance. The intersection point with a triangle primitive has an analytical solution, making the traversal much more efficient. Also, the bounding volumes of triangle primitives can be explicitly defined and require less computation than with spline patches. Further, the heightmap processes can generally be parallelised, such as the mesh update and surface normal computation. Hence, in terms of ray tracing, a triangle mesh generated from height map typically achieves better performance than spline-based representation. Unlike spline-based meshes, a single vertex movement in the triangle mesh affects a small local area, making it flexible to match any shape of geometry detail. Also, the impact of the vertex movement on a triangle mesh and corresponding radiance distribution is much easier to be predicted than with a spline-based mesh. The main problem of the triangle meshes is discontinuity as they only guarantee C_0 continuity, which might bring additional problems if the final mesh is to be physically fabricated.

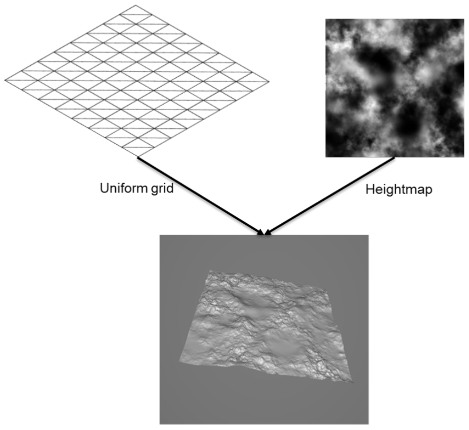


Figure 3.5: The heightmap is defined by a 2D spatial grid with a height value for each position. It is typically encoded in the form of a grey-level image. From the heighmap it is trivial to generate a triangle mesh by the elevating the vertices by their height (grey-level) values.

In brief, heightmaps are our choice for surface representation since they are highly flexible, compatible and fast. The problem of discontinuity can be avoided by additional strategies as will be seen in the next chapter. Finally, the heightmap can be mathematically defined as a 2D matrix with a height value for each vertex:

$$S : \mathbb{R}^2 \mapsto \mathbb{R},\ z_{i,j} = S(i, j) \tag{3.4}$$

where $z_{i,j}$ is the height in position (i, j). Before the ray tracing simulation, the heightmap is converted to a triangle mesh defined by a vertex list and an index list within a uniform grid. This is actually the same as the general definition of a triangle mesh and can be easily processed by CUDA and OptiX7.

In the general simulated annealing algorithm there are two common ways to generate a neighbour solution. The optimiser can either change one element or add an offset to each element during each iteration. Both of them can generate a neighbour solution that is close to the current solution. Since adding an offset to each element might affect the entire surface too much, introduce new random factors, and makes it hard to control the step size, we choose to move only one vertex during each iteration. The size of the displacement is related to the current temperature. The higher the temperature, the more drastic the displacement. The neighbour solution generation follows a two step-procedure:

1. randomly choose a vertex from the current mesh;

2. add a random offset to the vertex where the maximum displacement depends on the current temperature.

In this way, the optimiser will generate a neighbour solution and evaluate the solution with the objective function in each iteration. With the definition of radiance distribution and neighbour solution generation, the general simulated annealing follows the pipeline shown as Figure 3.1.

3.4. RESULTS

Based on the definitions of radiance distribution and surface representation a simple simulated annealing optimiser can be constructed. According to our tests, the general simulated annealing can indeed work in the inverse reflector problem, since it can achieve a radiance distribution similar to the target radiance distribution, as shown in Figure 3.6. Nevertheless, there are still many issues to be considered:

- The mesh may not be smooth enough having less chances to achieve a fabricatable result.

- The radiance distribution result is not smooth since the number of the triangle mesh vertices is limited due to the performance issue, as too many vertices can greatly slow the ray tracing simulation.

- The energy conservation is low since a large part of the radiance is reflected away and never reaches the receiving plane. The result has a similar radiance pattern to the target radiance distribution but not a similar radiance intensity. Consequently the error is not low even if the radiance patterns are almost identical.

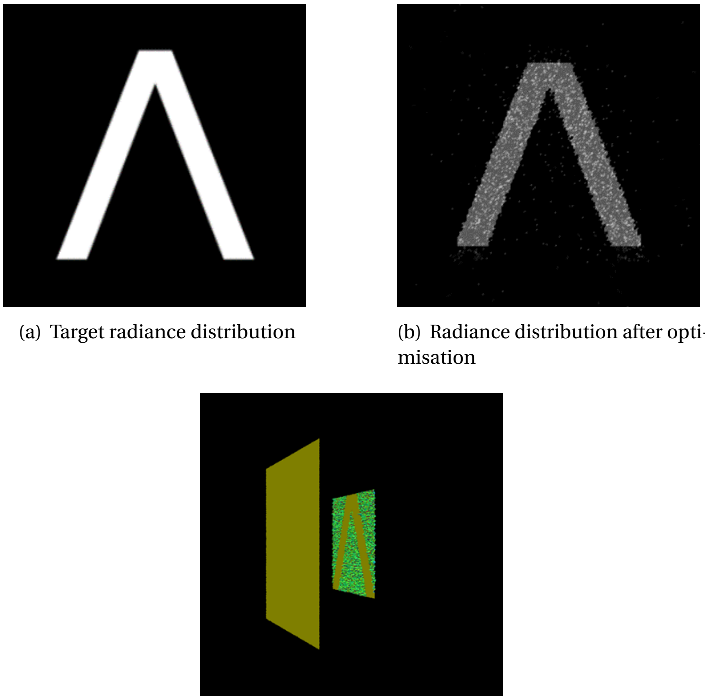

(a) Target radiance distribution (b) Radiance distribution after optimisation (c) Optimised mesh

Figure 3.6: Result with general simulated annealing (Error: 0.300265, energy conservation: 0.409506, Iterations: 288768, resolution: 200 × 200, mesh size: 201 × 201)

- Even though the final converged error is high, the convergence speed is not actually fast and the method converges slowly. For the example in Figure 3.6, the error cannot decrease any further after around 50000 iterations.
- The optimisation process cannot capture features in different sizes and directions. The final surface geometry may only contain sharp spikes which are different from the original shapes of the desired surface geometry that possibly exists, as shown in Figure 3.6.

Given the main issues listed above, we have developed several strategies to overcome these limitations. In the next chapter, we will discuss how we tackled each of these problems.

4

Optimisation strategies

4.1. SMOOTH RENDERING

The first issue of the general simulated annealing is the lack of smoothness in the resulting mesh. A low-resolution triangle mesh generated from a heightmap with the uniform grid has large discontinuities. It may also lead to many artefacts and convergence problems such as discontinuities in resulting radiance distribution. We can increase the number of vertices to achieve a smoother result, but it comes at a performance cost since ray tracing of a large triangle mesh can bring considerable overhead and any modification to a single vertex can only impact little on the final result. Instead, we consider applying smoothing techniques after the ray-triangle intersection test, thus approximating the result of ray-tracing a high-resolution mesh while keeping the complexity of the ray tracing algorithm low.

NORMAL INTERPOLATION

Normal interpolation is a common method for smooth shading. The idea is to first define the surface normal at each vertex and then interpolate inside the primitives using barycentric coordinates. In OptiX7 the barycentric coordinates are already available inside the shader program, so we do not need to explicitly compute them. Nevertheless, we still need to compute the normals per vertex.

Fortunately, when handling heightmaps, computing the normals is not a costly process. We can define the vertex normal vector by its gradient direction. Each vertex gradient can be computed by applying the central difference filter to the heightmap values:

$$v(i,j) = \left(\frac{S(i-1,j)-S(i+1,j)}{2}, \frac{S(i,j-1)-S(i,j+1)}{2}, 1\right) \tag{4.1}$$

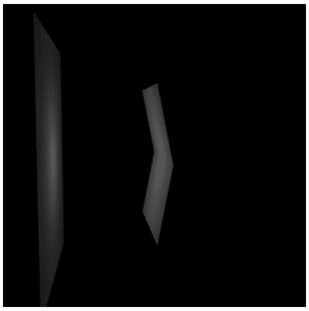
where $S(i,j)$ is the heightmap value at vertex (i,j), and $v(i,j)$ is the resulting normal at vertex (i,j). The resulting normal will be normalised before used in the shader program. By computing the normal at each vertex we can build a normal map associated to the heightmap. The normal map computation can be fully parallelised in GPU and, once ready, the optimiser can directly access the generated normal map in GPU memory. The ray tracing with normal interpolation can be summarised in the following two steps:

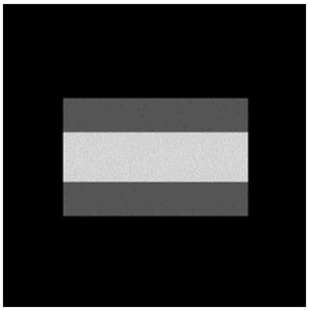
- Generate a normal map in the GPU by filtering the heightmap with the central difference kernel.
- Interpolate the vertices' normals using the barycentric coordinates and compute the direction of the reflected light ray using the interpolated normal.

During ray tracing, the ray tracer will use the interpolated normals to compute the directions of the reflected rays, which already gives a much smoother result compared with the flat shading method, as shown in Figure 4.1.

GEOMETRY SMOOTHNESS: PN TRIANGLES AND PHONG TESSELLATION

Although normal interpolation smoothes the normal inside each primitive, the intersection point between the ray and the triangle still lies on its generating plane. The origin of the reflected ray also affects the final radiance distribution, especially for low-resolution


(a) Test scenario in 3D view


(b) The radiance distribution on the receiving plane with flat shading

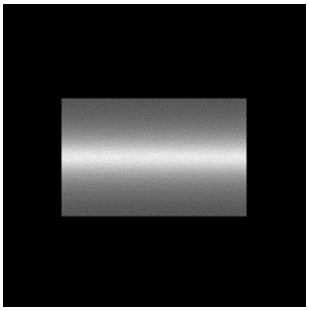
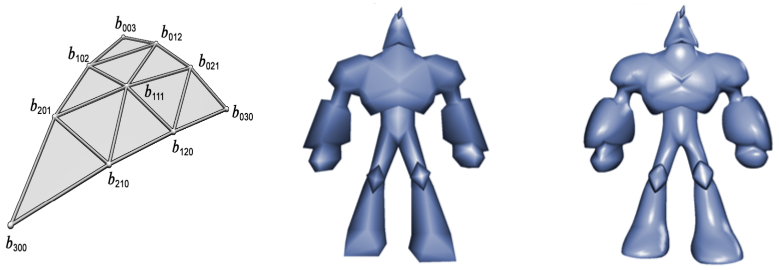

(c) The radiance distribution on the receiving plane with normal interpolation

Figure 4.1: Normal interpolation can achieve a much smoother result compared with flat shading.

meshes. One standard method for smooth rendering at geometry level is the spline-based surface which can guarantee C^1 continuity. Nevertheless, the performance overhead and incompatibility with the general ray tracing pipeline are still issues, as discussed in Section 3.3.

Another solution is to use local approaches, such as PN triangles [15] and Phong tessellation [16]. Compared with other spline-based surface representations, they are compatible with the existing rendering pipeline based on triangle meshes and do not require pre-tessellating the original triangle mesh at a higher resolution. The main idea is to build a smooth patch for each triangle on demand i.e., when a ray intersects the triangle. This kind of algorithm does not need any additional information except for the vertices coordinates and normals, hence, it is fully compatible and easily integrated into the current rendering pipeline.

The idea behind PN triangles is to define a list of 10 control points that control the geometry of each primitive patch, as shown in Figure 4.2(a). Every control point is interpolated from the original vertex and normals. These 10 points are then interpolated with a cubic polynomial to obtain a smoothed position on the surface. Using PN triangles to define the ray's intersection point can lead to a much smoother radiance distribution



(a) The PN patch geometry is determined by 10 control points. (b) Original triangle mesh (c) Mesh with PN triangles

Figure 4.2: PN triangle can smoothen the original triangle mesh. [15]

result on the receiving surface, as shown in Figure 4.3(d). However, the intense vector operations do bring an overhead of approximately 5-10% to the whole system.

Phong tessellation is another way of achieving smooth geometry. As shown in Figure 4.4, the basic idea is to first find a plane which is orthogonal to each vertex normal. Then, project the intersected point of the flat triangle primitive onto the corresponding tangent planes and obtain three projected points. Finally, use the barycentric weights to interpolate the three projected points to achieve the final interpolated position on the smooth patch. Compared with the flat shading method, Phong tessellation with normal interpolation results in a smoother radiance distribution and rendering, as shown in Figure 4.3 and Figure 4.6.

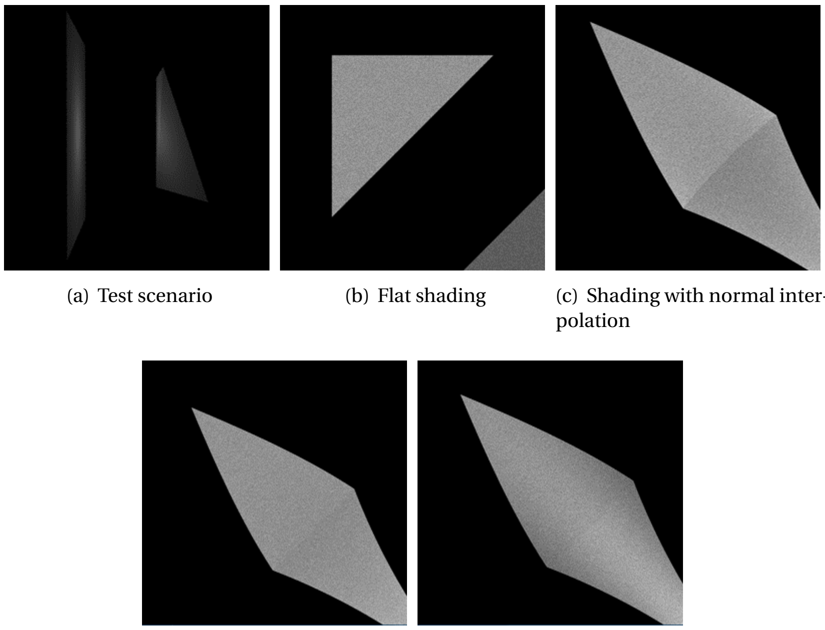
During the main optimisation phase, Phong tessellation is applied to achieve smooth rendering. However, during the optimisation, the ray tracer does not modify the triangle mesh's geometry but only corrects the position when computing the reflected point position. The actual geometry for computing reflected point position is different from the final triangle mesh result. After optimisation, the triangle mesh geometry still needs to be tessellated before fabrication. In the final phase of the pipeline, the optimiser will apply Phong tessellation to the optimised mesh and produce the actual triangle mesh that complies with the triangle mesh geometry used in ray tracing, as shown in Figure 4.5.

According to our test results, Phong tessellation is faster and smoother than PN triangles. Since the ray tracing result with the Phong tessellation is different from directly ray tracing the triangle mesh, it also means that the final optimised mesh will not produce the final radiance distribution if actually fabricated without further tessellation.

4.2. MESH UPDATES WITH GAUSSIAN KERNELS

In general simulated annealing, the way to update mesh has several issues that negatively affect the optimisation process. To solve the issues, the mesh update strategy with kernels and randomisation is introduced to the proposed method.

For each iteration during simulated annealing, the optimiser needs to generate a new guess solution in the neighbour search space. For the inverse reflector problem, a



(a) Test scenario (b) Flat shading (c) Shading with normal interpolation

(d) Shading with PN triangle (e) Shading with Phong tessellation

Figure 4.3: Comparison of different shading method: (b)(c)(d)(e) are radiance distribution on the receiving plane. Phong tessellation gives the smoothest result.

natural way to generate a neighbour solution is to update the triangle mesh/heightmap by displacing a single vertex. However, this trivial strategy brings additional issues. First, the change is too localised when the mesh size is large since each single vertex can only affect a small region of the whole surface. It also means that a single vertex cannot influence the radiance distribution too much and, consequently, the information provided by this iteration is not significant enough to decide whether the optimisation is going in the right direction.

The other problem is that, although the surface representation used in our project cannot ensure C^1 continuity, we still want the result to be as smooth as possible due to the limitation of fabrication. Changing a single vertex may result in large mesh discontinuity, as shown in Figure 4.7 and Figure 4.9(a).

To avoid excessive discontinuity and low stability during the optimisation we use Gaussian kernels to update the mesh. The idea is that in each iteration, the optimiser will randomly select the index of a vertex in the current heightmap mesh and update it by adding a Gaussian footprint centered on this vertex, as shown in Figure 4.8.

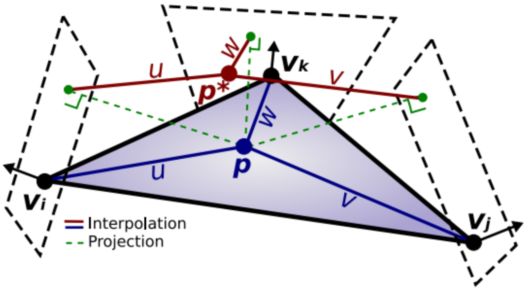


Figure 4.4: Phong tessellation: the position is interpolated with the three points projected onto the perpendicular surfaces. [16]

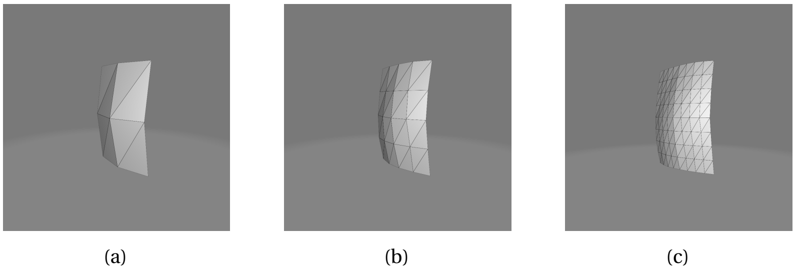


Figure 4.5: Final optimised mesh still needs to be Phong tessellated before fabrication

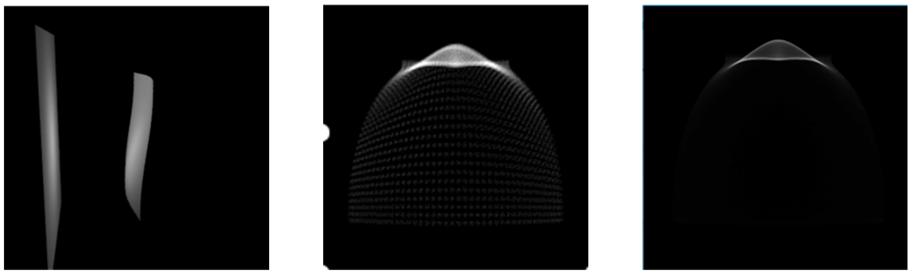
4.3. Randomisation of kernel shapes

In a practical application, the target surface geometry may have different feature shapes. An important observation is that a single Gaussian kernel is insufficient for capturing all different types of details. For example, if the target geometry is a surface as shown in Figure 4.10, the optimisation with a single default isotropic Gaussian kernel cannot converge to a satisfactory result. An anisotropic kernel may match the target geometry better and help the optimisation converge much faster. To ensure the optimiser can be compatible with different shapes of surface geometry, we apply a randomisation strategy to pick different types of kernels. The randomisation strategy among kernel shapes is introduced to solve the issues.

4.3.1. Basic randomisation strategy

The basic randomisation strategy is straightforward. The idea is to pre-define a list of different shapes of Gaussian kernels. The Gaussian kernel's shape is completely defined by the following 2×2 covariance matrix:

$$\sum = \begin{pmatrix} \delta_x^2 & \rho\delta_x\delta_y \\ \rho\delta_x\delta_y & \delta_y^2 \end{pmatrix} \tag{4.2}$$



(a) Original surface (b) Radiance distribution result without Phong tessellation (c) Radiance distribution result with Phong tessellation

Figure 4.6: Comparison of radiance distributions: shading with Phong tessellation (right) can give a smoother radiance distribution than flat shading (middle).

where δ_x^2 and δ_y^2 are the variances and ρ is the correlation. The covariance matrix is used to control the shape of the generated Gaussian kernels but note that the size of the kernel does not depend on the size of the covariance matrix, as shown in Figure 4.11. The kernel size is actually defined by the size of the discrete matrix used to represent the Gaussian distribution.

During optimisation, the optimiser randomly selects one Gaussian kernel shape to update the mesh at each iteration. This strategy ensures that various features can be captured while optimising the mesh, especially when handling complex radiance distribution and geometry. The overview of the strategy is shown in Figure 4.12. According to our test results, for a target geometry with different surface features, the optimisation with kernel shape randomisation converges around 15% faster than without it.

4.3.2. HISTORY-BASED DECISION STRATEGY

The randomisation strategy can work in most cases, but it still brings some overhead since the chances of selecting an unsuitable kernel are not low. The shape and the total number of Gaussian kernels selected for optimisation are also critical points. Although a large number of kernels with different shapes may help to achieve a better result, it may also decrease convergence speed in certain cases due to the randomisation across a large number of kernels with different shapes. An idea inspired by Q-Learning [17] is to dynamically adjust the probability of selecting each kernel and find the best kernel selection strategy to maximise the expected value (error decrease).

Q-learning is a typical model-free algorithm in reinforcement learning, maximising the expected value in any finite Markov decision process (FMDP). The key idea is to use a table to record the mapping between state-action and quality:

$$Q(s,a) : S \times A \rightarrow \mathbb{R}. \tag{4.3}$$

where S is the solution space including all the possible states and A is the space including all the possible actions. Table Q records the quality for each state-action transition combination. Then, the Q-table is used to adjust the probability of the state's transition.

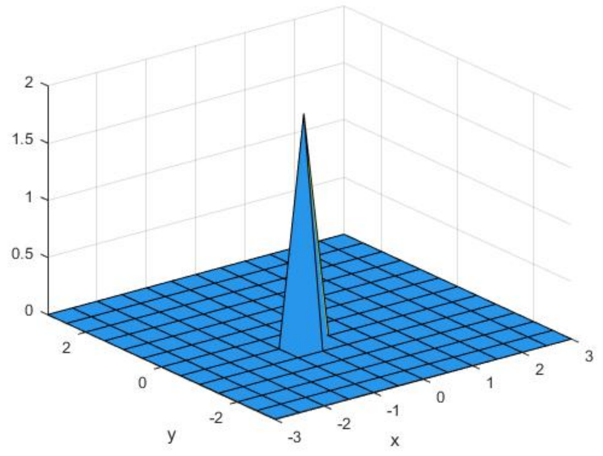


Figure 4.7: The mesh will form a very thin peak if we only displace the selected vertex

However, for the inverse reflector problem, the main problem of Q-Learning is that although it is model-free (which means it does not need to construct a full Markov process model), it can only guarantee that the optimisation works when the process can be theoretically modelled into an FMDP. In our global optimisation problem, the process we are handling is in a continuous space instead of a discrete space. Even if we merge the states by dividing the continuous search space into discrete blocks or by sampling, the number of states is still too large to construct a full Markov process model since the inverse reflector problem is an extremely high dimensional optimisation problem. There have been researches into Q-Learning which points out that, under specific conditions, Q-Learning can also have good performance even if the problem model is not an FMDP [18]. But unfortunately, our case still does not satisfy those conditions.

Our solution is to simplify the Q-Learning to a much simpler model, which is a history-based decision strategy. The history-based decision strategy can be summarised as the Q-Learning strategy in which all states are merged into one single state. The best-matched shape for optimisation is usually the shape with the largest *reward* (error decrease). The history-based strategy is to build a table to record the reward of each shape. The table is then used to adjust the probabilities of selecting each shape of kernels during the optimisation process. There are two phases where the history-based decision strategy can be applied: pre-process and run-time.

PRE-PROCESS

Before the primary optimisation process, the optimiser conducts a pre-process history-based strategy, which helps to obtain some pre-knowledge of the target geometry. The optimiser uses a list of kernels with different shapes (predefined or random-generated) to initialise the pre-process with a small number of iterations. The result of the *reward* record directs the decision of which kernel shapes should be used in the main optimisation phase.

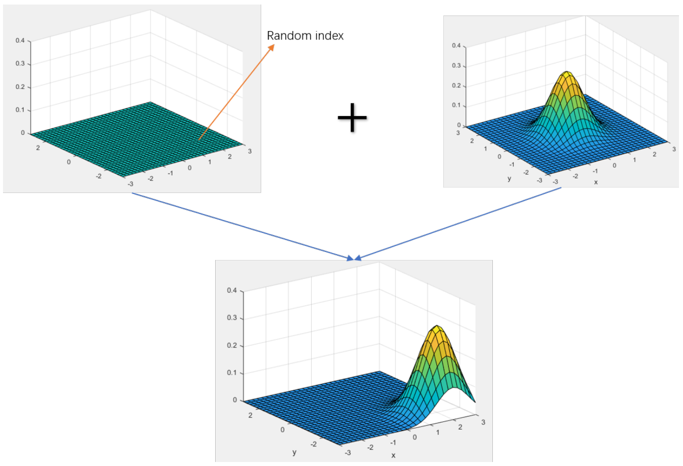
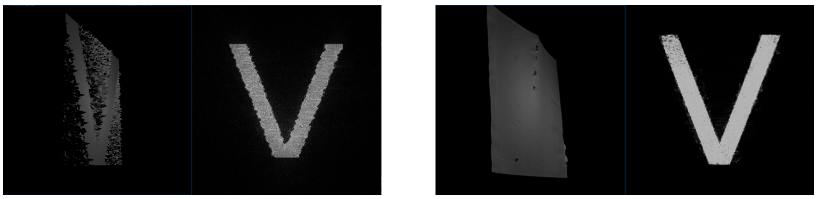


Figure 4.8: Generate neighbor solution by adding the Gaussian kernel (top-right) to a random position (top-left). The resulting surface (bottom) does not contain large discontinuities as it would if we had only displaced the randomly selected vertex as illustrated in Figure 4.7



(a) Surface with high frequencies: optimisation without Gaussian kernel

(b) Smooth surface: optimisation with Gaussian kernel

Figure 4.9: Optimised surface and the corresponding radiance distribution on the receiving plane: the smooth surface optimised with Gaussian kernel (right) is much simpler to be fabricated than the surface with high frequencies (left).

The best N shapes of kernels contributing the most significant error decrease are selected to the main optimisation phase. This selection is illustrated in Figure 4.13.

RUN-TIME

The run-time history-based strategy is to dynamically adjust the probabilities of each kernel after every *batchSize* iterations. During the main optimisation phase, a table records each kernel's contribution. The table is used to adjust the probability of acceptance of each kernel shape, as shown in Figure 4.14.

A typical downside of such history-based optimisation is that it can easily lead the global optimisation process to a local minimum since some shapes' acceptance probability can become too high. In contrast, others decrease to 0 and have no chance to be accepted anymore, as it will never be picked its probability will never increase. We solve this problem by setting a minimum probability for each shape of kernels. Also, to avoid the table from changing too fast, we update the table with a batch strategy, which means

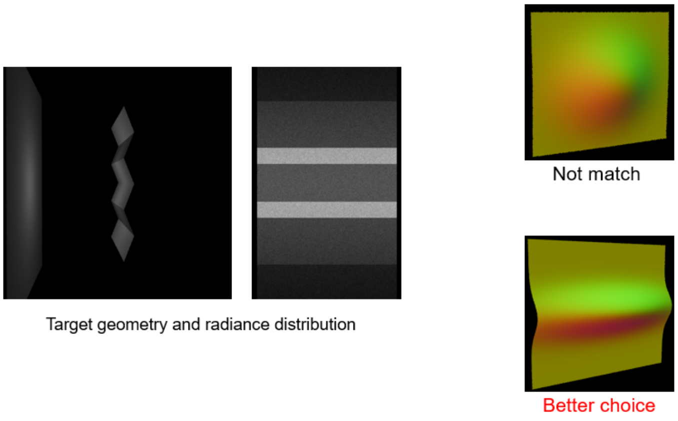


Figure 4.10: The shown radiance distribution is generated by the mesh on the left. For this case, an anisotropic kernel can be a better choice compared with an isotropic kernel. Note that in reality we do not actually have a target geometry, this is for illustration purposes only, but if we have this target distribution we expect the resulting mesh to be similar to this one.

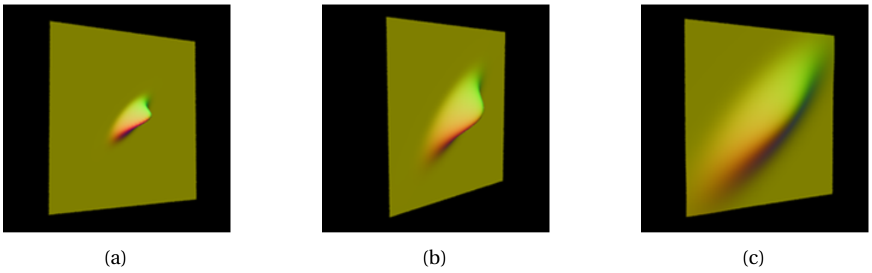


Figure 4.11: Three kernels with same covariance matrix, but different sizes

it only updates after a certain number of iterations (*batchSize*). It lowers the risk of being stuck in a local minimum and increases the convergence speed.

4.4. Multi-level strategies

One of the main problems of the inverse reflector design is that the search space's dimensions are too high to have enough search within a limited number of iterations. If we directly run global optimisation on such a large search space, it will be difficult for the optimiser to reach the global minimum within a reasonable amount of time.

One common way of solving such a problem is to apply "dimensionality reduction", that is, to reduce or compact the high-dimensional variables in the objective function into a lower-dimensional one while still ensuring that it is representative enough and maintaining most of the original characteristics. Spline-based surface representations can decrease the degree of freedom in such global optimisation problems since it only uses a small number of control points to represent the whole surface.

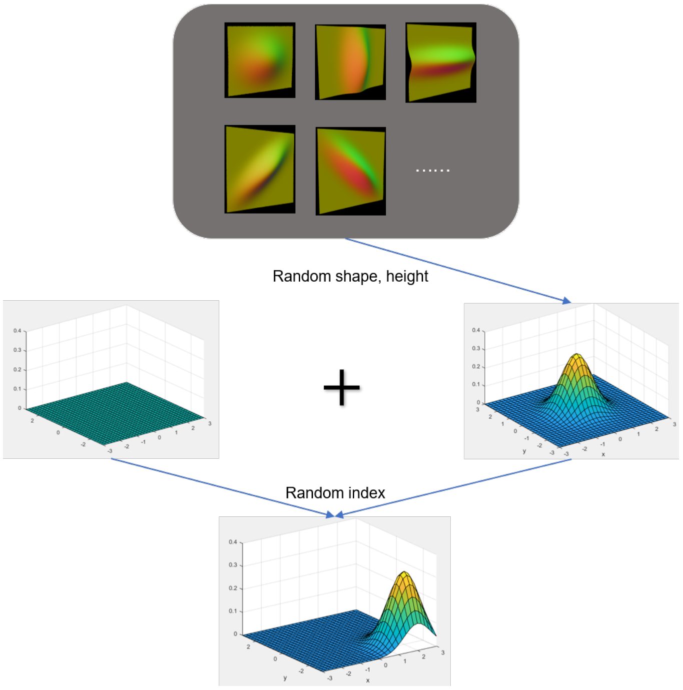


Figure 4.12: Randomisation: choose a random kernel shape for the mesh update.

Instead, we use a multi-resolution, or multi-level, strategy to limit the dimensionality during optimisation. With a multi-level strategy, the optimiser always starts from a coarse low-dimensional problem. When it converges to a solution at this level, the search space is refined into a higher dimension by interpolation. According to our test results, in a high-dimensional optimisation problem, the final solution with a multi-level strategy can be much closer to the global minimum than the direct single-level strategy within a limited amount of time.

In our project, we use multi-level strategies in three aspects: the resolution of the target radiance distribution; the resolution of a heightmap/triangle mesh; and the size of the Gaussian kernel.

MULTI-LEVEL RESOLUTION OF TARGET RADIANCE DISTRIBUTION

The optimiser starts with a coarse target radiance distribution by filtering the original radiance distribution with a linear box filter. Once convergence is reached, the resolution of the target radiance distribution is increased and the optimisation starts again. The different resolutions of target radiance distribution are obtained by interpolating the original input. After iterative repetition of this process, the optimisation will finally reach the final convergence. This strategy can significantly decrease (around 20%) the time needed for convergence, especially when handling complex target radiance distribution.

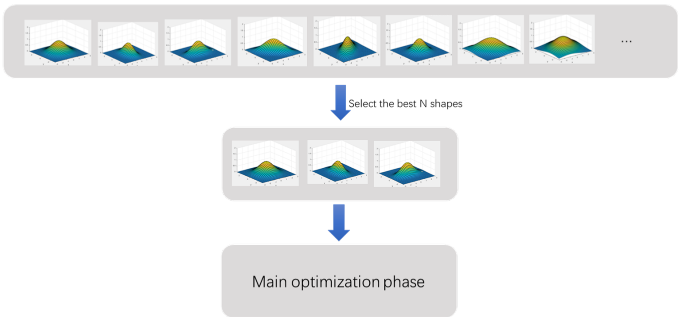


Figure 4.13: Pre-process: select N best-matched kernels for main optimisation phase. In this case $N = 3$

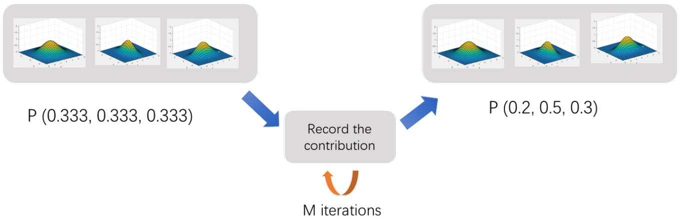


Figure 4.14: Run-time: dynamically adjust the probabilities of selecting each kernel.

Multi-level triangle mesh size

Similarly, we adjust the mesh resolution. The initial triangle mesh is coarse, containing a small number of vertices. After convergence, the optimiser increases the mesh's size by interpolating the triangle mesh to a finer mesh. The interpolation is bilinear inside each cell of the triangle mesh (or the heightmap), and after interpolation, the optimiser runs a triangulation and assemble the new vertices into a new triangle mesh. Every optimisation pass uses the previous level's optimisation result after interpolation as the initial triangle mesh. In this way, the optimiser arrives closer to the optimum without losing the result information of the previous level's optimisation. An example of this process is shown in Figure 4.15.

Multi-level Gaussian kernel

Finally, we also use a multi-level strategy for the Gaussian kernels. The reason for this is slightly different from the previous ones. Ideally, the reflector optimisation should have a good coarse shape first, and then optimise the small details on the reflector's surface. The target geometry might have features with the same shape but at different scales. To capture these features in different scales, we need to apply kernels with different sizes. The randomisation across the whole list of kernels with different shapes and sizes can be too costly, thus a multi-level strategy is again important. The optimiser starts with large Gaussian kernels and gradually decreases the kernel's size to capture the fine details.

The multi-level strategies mainly helps with the convergence speed. Generally, the optimiser using multi-level strategies can converge much faster than a single-pass optimiser, as shown in Table 4.1. The multi-level strategy is shown in Algorithm 1.

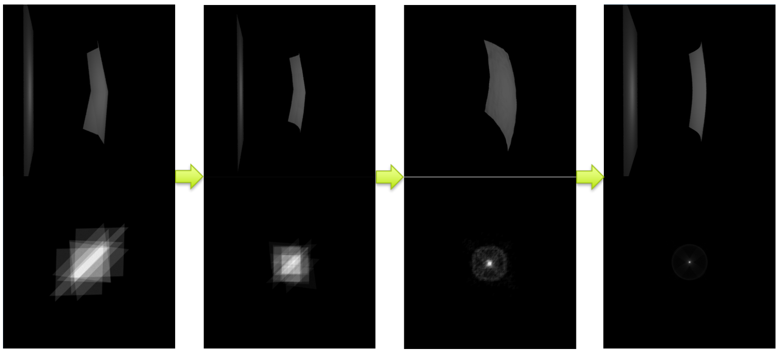


Figure 4.15: The result gradually approaches to the target by increasing the mesh size. (top row: mesh in 3D view; bottom row: resulting radiance distribution on the receiving plane)

Algorithm 1: Pseudo-code for the Multi-level strategy.

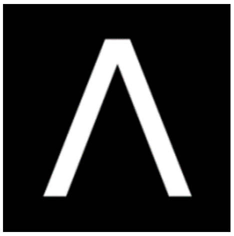
```
while !meshSize_max() do
    while !resolution_max() do
        while !kernelSize_min() do
            Optimize_SA();
            Decrease kernelSize;
        end
        Increase resolution;
    end
    Increase meshSize;
end
```

Case Num.	Iter.(No multi-level)	Iter.(multi-level)
Figure 4.16(a)	8.9×10^5	3.7×10^5
Figure 4.16(d)	2.1×10^5	1.3×10^5

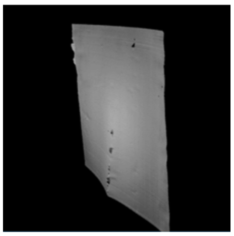
Table 4.1: Number of iterations needed for convergence (same error threshold for both 'multi-level' and 'no multi-level')

4.5. PENALTY AND MESH SMOOTHNESS

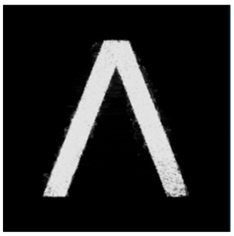
By far, the error function (objective function) only takes the radiance distribution information into account. Nevertheless, in practical applications, there might be additional constraints that are different from the ideal scenario. For instance, the boundary condition will force the surface to be within a specific range. The penalising strategy is a typical way to put constraints on the optimisation problem. Moreover, the way to achieve further mesh smoothness will be discussed in this section.



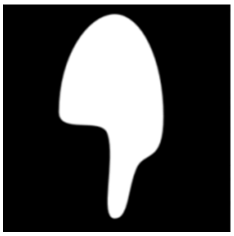
(a) Alphabet inverse 'V', Resolution: 200×200



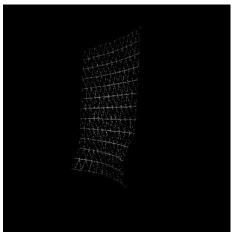
(b) Mesh size: 101×101



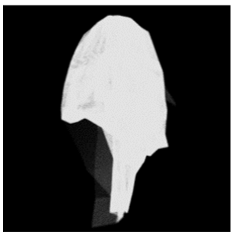
(c) Error: 0.087; Iterations: 3.7×10^5



(d) Headlight, Resolution: 50×50



(e) Mesh size: 13×13



(f) Error: 0.096; Iterations: 1.3×10^5

Figure 4.16: Test cases for Table 4.1 (left: target radiance distribution; middle: optimised triangle mesh; right: resulting radiance distribution; light source: collimated light source)

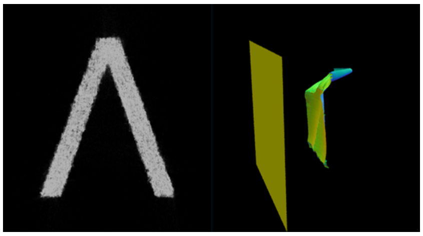
4.5.1. TYPICAL PENALISING STRATEGIES

For constraints/conditions, we can attribute a penalty function that can guide the objective function in certain conditions. In this way, the optimiser is forced to go into the search space where the penalty is not high. We explicitly created penalty functions for four cases:

- Energy conservation;
- High-frequency error;
- Concave valleys with a small angle;
- Boundaries.

ENERGY CONSERVATION

For the energy conservation constraint is to avoid solutions where the surface is reflecting light away from the receiving plane. Assuming the total radiance energy reaching the reflector is E, and the total radiance energy on receiving plane is E_r, then the energy conservation is defined as E_r/E. Energy conservation can be of high interest in practical applications since we do not only care about the radiance patterns, but also the radiance intensity. As shown in Figure 4.17, even if the resulting radiance distribution has the desired shape, the optimiser bent half of the surface away to lower the error, which is an


(a) Part of radiance cannot reach receiving plane

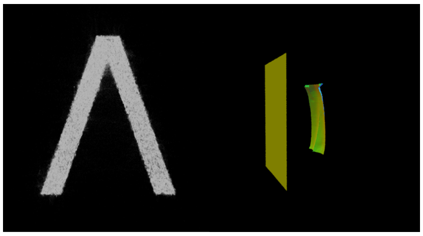

(b) Most of radiance can reach the receiving plane

Figure 4.17: The result can be in same radiance pattern but totally different energy conservation

undesirable solution. The idea is to add a penalty when the energy conservation is lower than a specific threshold. The total error then becomes:

$$\text{Error}_{\text{new}} = \frac{\text{Error}_{\text{old}}}{(\frac{E_r}{E})},\ if\ \frac{E_r}{E} < \text{Threshold} \tag{4.4}$$

The final error will be divided by energy conservation when it is lower than a specific threshold. If the energy conservation is too low, the error will be scaled up and the optimiser will be forced to increase the energy conservation in order to decrease the total error.

HIGH FREQUENCIES

Human eyes are normally more sensitive to high-frequency areas such as edges. Typically, we would prefer a radiance distribution result that has similar high-frequency features to the target radiance distribution. Sobel filter is a common way to extract high-frequency information from an image. The idea is to compute the MAE error between the target and current radiance distribution after applying the Sobel filter. The Sobel MAE error is added to the total error, hence, the optimiser tries to minimise the objective function together with the high-frequency error.

The corresponding penalty is:

$$\text{Penalty}_{\text{Sobel}}(R_T, R) = \frac{\alpha}{2N}\left(\sum_{ij}\left|\text{Sobel}(R(i,j)) - \text{Sobel}(R_T(i,j))\right|\right) \tag{4.5}$$

where α is a factor to give more or less importance to the Sobel error. It is worth noting that we do not recommend setting the Sobel α to higher than 0.2 under normal circumstances. One reason is that the high α values affect the optimisation process since it can make the optimiser put huge effort into optimising the high-frequency error instead of the radiance error part we mainly care about. The other reason is, in the actual optics design application, the light does not follow the perfect-mirror reflection as we assume in this work. The radiance on the receiving plane will become smoother and more blurry due to the scattering effect. For complex target radiance distribution, it is almost impossible for the reflector to capture all the high-frequency details. So it is unnecessary to over-weight the Sobel part.

Concave valleys with a small angle and Boundaries

Concave valleys and boundaries can also be part of the penalties. Generally, the machine for fabricating lenses cannot mill surfaces with small concave angles or outside the boundaries (which means the height value must be within a specific range $[h_{min}, h_{max}]$). The way to compute the penalty is to scan over the whole surface and find the vertices which do not satisfy the constraints on concave valleys and boundaries. The constraints such as the angle of concave valleys and boundaries depend more on the limitation of the machine that will be used for fabrication. The advantage of the run-time penalising method is that it can avoid ruining the original optimised surface when compared to applying a post-processing method. Of course, it comes at the cost of more per-iteration overhead. Currently, according to some simplified tests, the penalising strategy works, although it is not yet fully implemented. Some further tests and research may be needed in future.

4.5.2. Further mesh smoothness

Even with the penalties above, the resulting surface might still have regions that are not smooth enough. For further mesh smoothness, we applied two kinds of methods: run-time and post-processing.

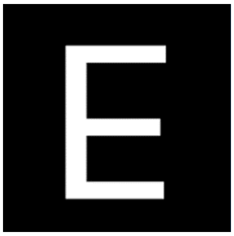
The ways to ensure the mesh smoothness in run-time include:

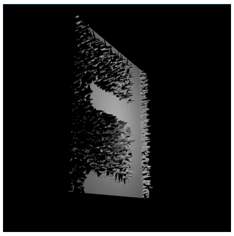
- Limit the minimal size of Gaussian kernel
- Penalise the non-smoothness surface by energy conservation

The size of the Gaussian kernel affects the smoothness of the mesh. A very small kernel can lead to many spikes in the final optimised surface mesh. For example, if we optimise the surface with only 1×1 Gaussian kernel, the optimisation process will degenerate to move a single vertex in each iteration. The final result will not be smooth, as shown in Figure 4.18(b). So one of the most direct ways to ensure mesh smoothness is to limit the minimal size of the Gaussian kernel.

Another way to smooth the mesh is post-processing filtering. We apply smooth filtering on the resulting heightmap to smooth the sharp areas. As shown in Figure 4.19, an appropriate post-processing filter can make the mesh smoother and maintain the most original radiance distribution when the limitation of the manufacturing machine is of high interest. The overhead of post-processing is negligible compared to run-time strategies. Nevertheless, for cases where the result accuracy is much more important than smoothness, run-time strategies can be a better choice.

The final pipeline of the algorithm is shown in Figure 5.1. The design considers the theoretical feasibility, complexity, compatibility and performance with CUDA and OptiX7. We mainly focused on the algorithm design in this chapter, and in the next chapter, we explain how to integrate the entire pipeline with CUDA and OptiX7.


(a) Target radiance distribution


(b) Optimisation with 1×1 Gaussian kernel: small angles of the concave valleys lead to high frequencies in the final mesh.

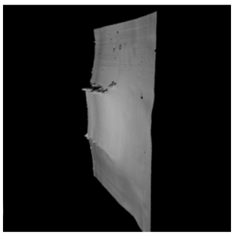
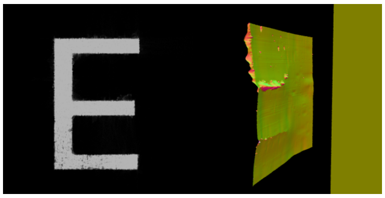

(c) Optimisation with multi-level Gaussian kernels: the final optimised mesh is much smoother in general.

Figure 4.18: The mesh (b) has concave valleys with smaller angles than the mesh (c), and it is more difficult to be fabricated.


(a) Before filtering

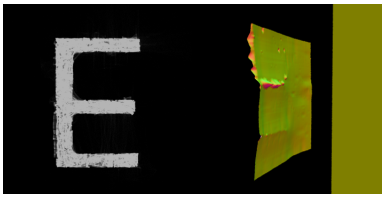

(b) After 3×3 box filtering

Figure 4.19: An appropriate post-filtering can smoothen the mesh

5

System design and development

The proposed reflector design system uses OptiX7 and C++/CUDA. In this chapter, we explain the details in building the system and how we optimised its performance. For the basic concepts of ray tracing, we refer the reader to an introduction to the topic [19].

5.1. OVERVIEW OF THE REFLECTOR DESIGN SYSTEM

The reflector design system is overall a global optimisation process. The GPU and CPU work cooperatively, in which the CPU is used for flow-control of the optimisation pipeline, and the GPU is used to accelerate compute-intensive tasks. The full pipeline is divided into two main phases: Pre-configuration and Main optimisation phase. The overview of the pipeline is shown in Figure 5.1.

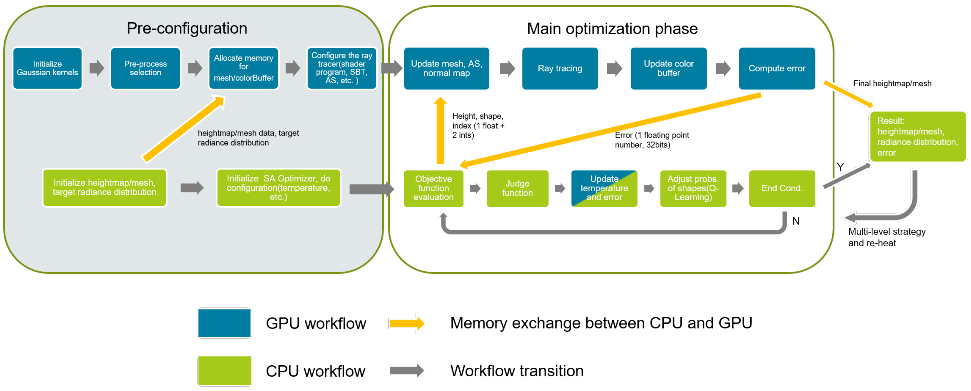

Pre-configuration
Initialize Gaussian kernels
Pre-process selection
Allocate memory for mesh/colorBuffer
Configure the ray tracer(shader program, SBT, AS, etc.)
heightmap/mesh data, target radiance distribution
Initialize heightmap/mesh, target radiance distribution
Initialize SA Optimizer, do configuration(temperature, etc.)
Main optimization phase
Update mesh, AS, normal map
Ray tracing
Update color buffer
Compute error
Height, shape, index (1 float + 2 ints)
Error (1 floating point number, 32bits)
Objective function evaluation
Judge function
Update temperature and error
Adjust probs. of shapes(Q-Learning)
End Cond.
N
Y
Final heightmap/mesh
Result: heightmap/mesh, radiance distribution, error
Multi-level strategy and re-heat
GPU workflow
CPU workflow
Memory exchange between CPU and GPU
Workflow transition

Figure 5.1: The pipeline overview: CPU and GPU work cooperatively

In the pre-configuration phase, the optimiser will first perform initialisation steps and allocate memory. More specifically, memory is allocated for the heightmap, radiance distribution buffer and Gaussian kernels. This single-time initialisation and memory allocation are much faster than dynamically operations during optimisation. Especially for the Gaussian kernels, computing a sample from a Gaussian distribution during optimisation can be much slower than looking up pre-computed values. In addition, in this phase we set the parameters for simulated annealing, such as temperature, max iterations, cooling schedule, step size and resolution.

We also want to avoid re-configuring the OptiX7 modules before launching the ray tracer in each iteration. The OptiX7 configurations mainly including shader program compilation, shader binding table and acceleration structure are all done in the pre-configuration. The acceleration structure supports the dynamic update without rebuilding if it uses built-in primitives, such as triangles. The performance gains more from pre-building the acceleration structure and dynamically updating than rebuilding it at each iteration.

The main phase is where the iterative optimisation happens. There are two key points here: workflow control and cooperative work on CPU and GPU. The workflow is controlled by the CPU, including all the steps like objective function evaluation, judge function, invoking the GPU execution, etc. The workflow controlled by the CPU is similar to the general simulated annealing algorithm running solely on the CPU, except for the parallelised parts optimised by GPU. The system is specifically designed for cooperative work on both CPU and GPU. The GPU is responsible for the compute-intensive tasks, and the CPU is responsible for the workflow control. For example, when handling the objective function evaluation, the CPU will invoke the GPU to run the ray tracing simulation. The cooperative work between CPU and GPU can significantly increase the performance of the optimisation process.

5.2. RAY TRACING SCHEME FOR SIMULATION

The ray tracing scheme for simulation is developed with ray tracing API OptiX7, which is responsible for the most compute-intensive part in the pipeline. In this section, the ray tracing scheme design and the application of Phong tessellation in OptiX7 will be detailed.

5.2.1. ITERATIVE RAY TRACING WITH OPTIX7

The core of the OptiX7 program is the shaders which mainly define how light rays interact with the traversable that include all objects in a scene. OptiX7 also builds an acceleration structure before the actual ray tracing. The main performance overhead of ray tracing lies in the intersection test, and the primary purpose of the acceleration structure is to accelerate the intersection test. In OptiX7, the shader binding table binds data to shaders; thus, the program can be compatible with different scenarios. The OptiX7's device-side API mainly contained eight kinds of programs, as shown in Figure 5.2. And almost all of them except for the scene traversal are programmable.

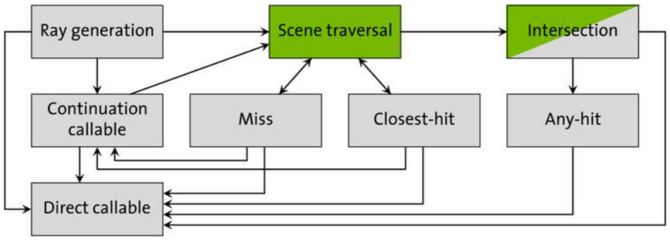


Figure 5.2: The full shading pipeline on GPU includes eight kinds of program [20]

Although the full OptiX7 shading pipeline includes eight kinds of programs, it can be simplified. For the purpose of our ray tracing simulation we mainly concentrate on the ray generation, closest-hit and intersection programs. Since the primitive in our scenario is the triangle, the built-in intersection shader is sufficient. The ray generation program is the entry point of the Optix7 Program, in which the light rays are generated according to the user definition of the light source. The closest-hit program is called every time when the ray tracer finds the closest hit point. It can be used to define how the light rays interact with the surface.

One of the most simple way to build a ray tracer is by shooting rays from the camera and recursively following the rays after each intersection. In this case the program needs to call the Trace() function recursively until it stops. The result would then be traced back to the camera to define the pixel colors. However, recursion affects performance, especially when handling inner-reflections. In recursive ray tracing, it is always necessary to build a stack to store the information required when returning.

We convert the original recursive scheme to an iterative ray tracing scheme with the per-ray-data that is to communicate between various shader programs. Note that we do not need to trace back to the origin of the rays, as we are only interested in the final intersection with the receiving plane. We also do not shoot rays from the camera but from the light-source, as the viewer is not relevant for our purposes.

As shown in Listing 5.1, the per-ray-data will carry ray origin, direction, depth and a boolean value indicating where it should trace again. The rayGeneration() generates new light rays according to the current per-ray-data. The closesetHitSurface() handles the ray reflected from the reflector and returns the ray information back to rayGeneration(), and the rayGeneration() then generates light rays with the per-ray-data until it meets the end condition. If it hits the receiving plane, it will call closestHitImage() and update the current radiance distribution according to the previous radiance distribution definition.

Listing 5.1: Shader programs

```
struct RayPayload
{
        vec3 rayOrigin, rayDirection;
        bool traceAgain; int depth;
};
void rayGeneration()
{
        RayPayload payload{};
        do
        {
                traceRay( ... );
        } while( payload.traceAgain && depth < threshold );
}
void closestHitSurface()
{
        payload.traceAgain = true;
        payload.rayOrigin = hitLocation;
        payload.rayDirection = ...; // compute
}
void closestHitImage()
{
        payload.traceAgain = false;
        UpdateRadianceImage();
}
```

5.2.2. CORRECT REFLECTION POSITION BY PHONG TESSELLATION

Phong tessellation in the rasterisation program is usually implemented in the geometry shaders, in which we can add/delete primitives. In OptiX7, there is no such shader that can directly handle geometry and primitive. Running Phong tessellation in GPU before every iteration is doable, however, the tessellation and rebuilding the acceleration structure implies a large overhead. Our method does not directly tessellate the triangle mesh. Instead, it corrects the reflected position according to the interpolation of Phong tessellation.

The reflected point is corrected by Phong tessellation inside the closest-hit shader. The shader program first retrieves the position and normal of three vertices of the intersected triangle. In addition, OptiX7 provides a built-in function to retrieve the barycentric coordinates and the primitive index. Hence, all the information needed to perform Phong tessellation can be obtained inside the closest-hit shader. The closest-hit shader saves the interpolated geometry position to per-ray-data and returns it to the ray generation program. The ray generation program then generates the reflected ray with the corrected reflected position saved in per-ray-data.

One issue is that the reflected ray might be blocked by the original primitive in the acceleration structure, since it is not modified. For instance, the corrected reflected point might move backwards compared with the original one, and the reflected light rays will be blocked by the original acceleration structure, as shown in Figure 5.3. To solve this problem, we make the per-ray-data carry a boolean value to indicate whether the corrected reflected point moves backwards. In this case, it skips the first intersection and continues ray tracing.

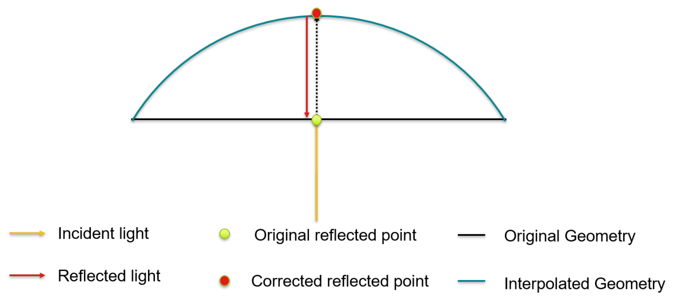


Figure 5.3: The corrected reflected ray can be blocked by the original flat primitive. In this case, another undesired call of the closest-hit will be generated.

5.3. ACCELERATION

For a system based on cooperation between CPU and GPU, the performance can vary a lot. Good system design and development are essential for achieving high performance. For developing the inverse reflector design system, there are several typical ways to get acceleration, including parallelised operations, minimisation of memory exchange and allocation, which will be detailed in this section. Moreover, practical tricks for accelerating a CUDA/OptiX7 program are also applied to this system for further acceleration.

5.3.1. PARALLELISED OPERATIONS

In the inverse reflector design pipeline, several compute-intensive operations can be accelerated by parallelisation, including error function evaluation, matrix computation and data update, which can be much more efficient than sequential operations.

ERROR FUNCTION EVALUATION

The most intuitive way to compute the error function is to transfer the radiance distribution result from GPU to CPU to compute the error. However, there are two issues with this approach. First, the data exchange between the CPU and GPU brings overhead. Second, the sequential computation of the error-function is slow when the resolution is high. Since OptiX7 and CUDA share the same API for accessing GPU memory, it is possible to compute the error function inside the GPU using CUDA kernels and avoid the workload on CPU.

To increase the reflector design system's performance, we strictly limit the data exchange between CPU and GPU. Before the main optimisation starts, the target radiance distribution is transferred once to GPU. During the main optimisation phase, the optimiser computes the errors in the GPU and only transfers back to the CPU a single floating-point scalar, the total error. This value is used for further optimisation, such as judge function. The overhead of sending one floating-point number is negligible when compared to sending the entire radiance distribution back to CPU. Moreover, the actual computation of the MAE between two matrices is faster in the GPU.

COMPUTATION AND DATA UPDATE

Our system performs a few local operations on 2D matrices, including computing normal maps, Sobel filtering, convolution, etc. This type of computation can be easily parallelised since each entry's result is independent of the others. For 2D convolution, further acceleration is possible by using separable filters, in which the 2D Kernel is decomposed into two 1D kernels and filtering is realized in each dimension separately. The complexity of computing the result will decrease from $O(NM)$ to $O(N+M)$.

In the pre-configuration, the GPU memory is allocated for the data and will not be re-allocated in the main optimisation. For all the data update that can be parallelised, we accelerate it with GPU. In addition, most update operations are also run in the GPU, such as updating the heightmap, triangle mesh, acceleration structure, radiance distribution and normal map. For example, the mesh update with the Gaussian kernel is implemented in GPU. As mentioned before, the Gaussian kernels are initialised and pre-allocated as matrices in GPU memory. During the main optimisation, the mesh update is an *add* operation between the heightmap matrix and Gaussian matrix, which can be fully parallelised. Other operations are similarly parallelised.

Since all the data updates are in-place operations inside GPU, there will be no additional memory exchange overhead between CPU and GPU during optimisation. The data update with GPU is much faster than it with CPU. For example, in certain cases, such as the mesh update, the GPU implementation is 100% faster than the CPU version, as shown in Table 5.1. Furthermore, the GPU implementation time increases much slower with mesh size than the CPU version.

MeshSize	Time (CPU multi-thread)	Time (GPU)
51×51	69 secs	48 secs
101×101	87 secs	50 secs
201×201	160 secs	53 secs
401×401	563 secs	70 secs

Table 5.1: Time spent in every 1×10^5 iterations with different mesh update methods (Radiance distribution resolution: 100×100)

5.3.2. PRACTICAL ACCELERATION TRICKS

Programming with CUDA and OptiX7 has a lot of tricks that can significantly improve performance. In this part, we will introduce some acceleration tricks used in our project.

USE PAYLOAD REGISTERS TO SAVE PER-RAY-DATA

The per-ray-data is the data used to communicate between various shader programs. Typically, the easiest way to use the per-ray-data is to compact the structure into a device pointer. When the program needs to use it in other shader programs, it can decode the pointer and retrieve the per-ray-data. However, in this case, the device pointer refers to the global memory, which is much slower.

One way to avoid such performance overhead is to make use of the payload registers. In GPU programming, the registers are by far faster than the global memory. The Optix-Trace() function provides payload registers with a maximum of 8×32 bits unsigned ints. So if the struct of per-ray-data size is less than 8×32 bits, we can encode the per-ray-data with the bits and communicate between various shader programs.

In this way, the shaders will not need to encode the data structure with a device pointer and decode it from global memory. The shaders can directly access the payload registers, saving all the information needed for the computation inside shaders.

REDUCE SCHEDULING OVERHEAD AND INCREASE GPU USAGE

The computation cost of the ray tracing simulation in each iteration is low. The total number of generated light rays is far more than the maximum number of parallel lines, making it impossible for all light rays to be computed in parallel simultaneously. If each thread only generates a single light ray, the overhead brought by thread/block scheduling is very high. It is better to set a fixed number of generated rays for each thread, which can minimise the overhead of thread/block scheduling. In our case, we set this number to 32. Compared with 1 ray per thread, it gives about 20-40% performance improvement in our test. And in most cases, 32 rays per thread is a bit faster (5%-10%) than 16 and 64.

Since the kernel launch time is too short, the GPU cannot be fully occupied due to the frequent status switch between CPU and GPU. One way to increase GPU usage is multiple streams launch. The system will launch concurrent ray tracing kernel functions with multiple streams (one kernel launch per stream), and separate streams can overlap to fill up the idle time. In this way, the system will batch the ray tracing simulations, including multiple mesh updates and objective function evaluations, to maximise GPU usage.

COMMON ACCELERATION TRICKS IN CUDA

There are also some typical acceleration tricks in GPU programming that we tried to follow in this project:

- Shared memory: shared memory is the memory shared with all threads within the same block. It is much faster than directly accessing the global memory;
- Memory coalescence: memory coalescence with half-warp alignment helps improve the performance of CUDA programming;
- Avoid double floating-point computation;
- Avoid divergence and idle threads.

These tricks can accelerate the system to some extent, but not very significantly. They are common tricks instead of specifically appropriate for our system since the main overhead inside the ray tracing simulation has already been fully optimised inside OptiX7. For more details about acceleration tricks in CUDA, we refer the reader to the CUDA programming guide [21].

6
Results and analysis

6.1. RESULTS

We tested our method in two ways. In the first, we generate the target distribution from a given geometry, hence, we can compare our resulting geometry with the ground-truth. In the second, we do not know the ground-truth geometry and we only have the target radiance distribution, which can be an image, for example.

Figure 6.1(a) illustrates the original geometry for the first type of test. Given this geometry we produced the radiance distribution depicted in Figure 6.1(b) by ray-tracing. Given this distribution as input, the optimiser arrived at the geometry in Figure 6.1(c) with resulting radiance distribution shown in Figure 6.1(d). Note that, even though the target and resulting radiance distributions are very similar, the ground-truth and resulting geometries are not.

For the second type of test, we defined simple radiance patterns, such as alphabet letters and a headlight shape as shown in Figure 6.2 and Figure 6.3. We also defined more complex patterns by using a gray-scale image as input, such as the one in Figure 6.4. The optimisation process is the same as the test cases with simple radiance patterns in the first type of test. The only difference is that the target distribution is not generated by ray-tracing and we do not know the ground truth geometry.

Case	Radiance Res.	Mesh size level	Iteration	Time
Figure 6.2(c)	$50^2 \to 200^2$	$25^2 \to 101^2$	5.3×10^5	337 secs
Figure 6.2(f)	$50^2 \to 200^2$	$25^2 \to 101^2$	3.7×10^5	239 secs
Figure 6.2(i)	$25^2 \to 50^2$	$7^2 \to 13^2$	1.3×10^5	75 secs
Figure 6.3(c)	$50^2 \to 200^2$	$25^2 \to 101^2$	9.2×10^5	548 secs
Figure 6.4	$50^2 \to 200^2$	$51^2 \to 401^2$	5.6×10^6	83 mins

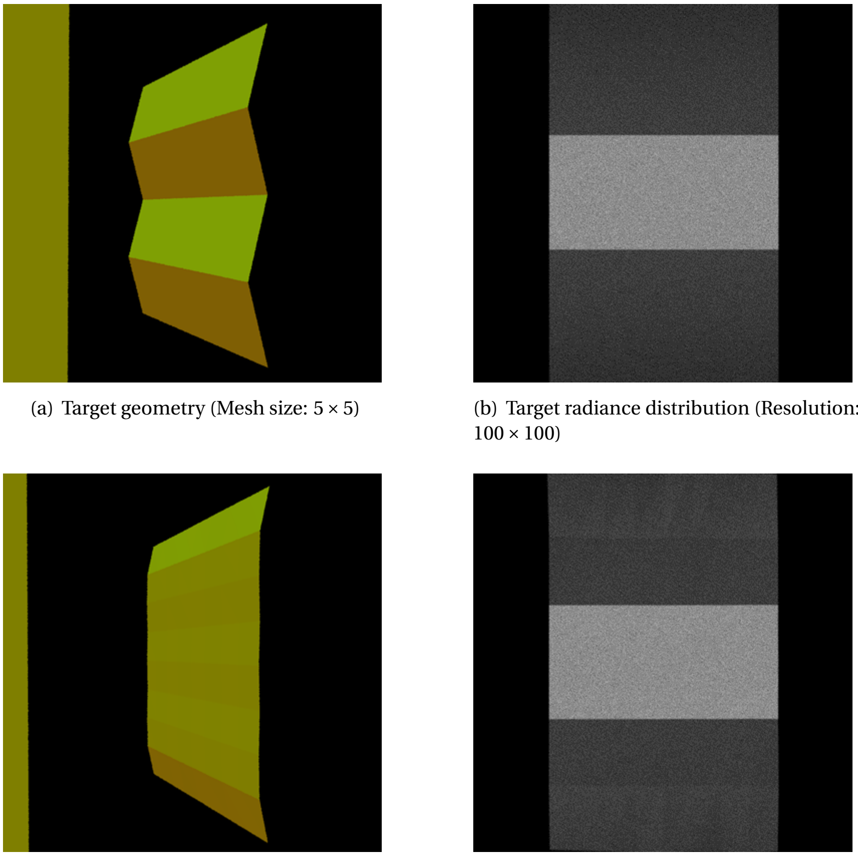
Table 6.1: Parameters and records in test cases (Multi-level: for level N, $Res_N = 2 \times Res_{N-1}$. The mesh size also doubles for each level, but it will be automatically set to the closest odd number if it is an even number.)

Run	Iteration	Time	Energy	Error
1st	1.3×10^5	75 secs	0.989	0.096
2nd	1.7×10^5	92 secs	1.0	0.092
3rd	1.4×10^5	81 secs	0.997	0.103
4th	1.7×10^5	95 secs	1.0	0.089

Table 6.2: Multiple runs for the same test case 6.2(g) (headlight shape)

Case	Kernel(a)	Kernel(b)	Kernel(c)	Kernel(d)	Kernel(e)
Figure 6.2(a)	0.30	0.13	0.25	0.15	0.17
Figure 6.8(a)	0.23	0.32	0.10	0.21	0.14

Table 6.3: Final estimated weights of pre-defined kernels in Figure 6.5



(a) Target geometry (Mesh size: 5 × 5)

(b) Target radiance distribution (Resolution: 100 × 100)

(c) Result geometry (Mesh size: 9 × 9)

(d) Result radiance distribution (Resolution: 100 × 100)

Figure 6.1: Optimised result with both target geometry and radiance distribution (Error: 0.0354, Iterations: 4096, Energy Conservation: 0.9987)

6.2. ANALYSIS AND DISCUSSION

To evaluate and validate the proposed method, I perform a performance analysis based on the recorded test data in different scenarios. As a general inverse reflector problem solver, the proposed method still has several issues to be discussed. Both the advantages and disadvantages will be explained in this section.

6.2.1. PERFORMANCE ANALYSIS

The performance of the proposed method includes three parts: convergence speed, accuracy and compatibility. In this section, each of them will be analysed in detail.

CONVERGENCE SPEED

The convergence speed is highly related to the desired result quality and hardware. Our proposed method aims to find a balance between quality and speed, and its convergence speed is good according to tests. In our experiment environment, for simple cases like Figure 6.2 and Figure 6.3, our algorithm normally converges between 1-30 minutes. For complex cases, like Figure 6.4, our algorithm converges within 30 mins to several hours. For cases with a large triangle mesh, although the mesh process runs fully on GPU, the speed (iterations/sec) is slower since it reaches the maximal number of parallel running threads for mesh processing on GPU, as shown in Table 6.1. It works the same for cases with a large radiance resolution.

Compared with other gradient-based algorithms like SPSA shown in Figure 6.10, our algorithm's significant advantage is that it does not require high sample-rate ray tracing. The reason is that the gradient itself cannot be accurate when ray tracing with a low sample-rate. And for the gradient-based algorithms, any random factor can significantly affect the computed gradient, considerably lowering the algorithms' compatibility and generality. Most of the overhead in the pipeline is in the ray tracing simulation, and the required high sample-rate ray tracing in SPSA may bring a lot of overhead compared with our proposed algorithm.

Some accuracy-oriented algorithms are specifically designed for certain application scenarios, such as "parallel incident light rays", "incident rays perpendicular to the surface", "far-field", etc. Since our algorithm is performance-oriented, the final result is not as highly accurate as accuracy-oriented algorithms, as shown in Figure 6.11. The proposed method can achieve an accurate solution fast in simple cases. When solving problems with a complex radiance distribution, although the result is not as precise as the accuracy-oriented algorithms, it is still reasonable and fast enough to serve as the initial guess for further approximation, which is, also the goal from the start (Section 1).

By now, the analyse of the convergence speed comparison between different algorithms is theoretical, since the original papers do not provide detailed data for all test cases. Moreover, even for cases with detailed data, the comparison is not in the same conditions (same machines). This analysis aims to show the potential advantages of the proposed method.

ACCURACY AND ENERGY CONSERVATION

For the tests in Figure 6.2 and Figure 6.3, the normalised error is under 0.10. As shown in Figure 6.1, we also note that the optimisation does not necessarily arrive at a geometry close to the original one. Nevertheless, it can still give a radiance distribution that is similar to the target radiance distribution. We conclude that this is because there are multiple solutions for a single optimisation problem, which means different geometric shapes might give the same result radiance distribution. Compared to the SPSA algorithm, our method presents significantly better results, as shown in Figure 6.10 and Figure 6.11.

Energy conservation is highly related to the penalty in the objective function. For test cases where no energy conservation penalty is applied, the energy conservation will fluctuate intensively during optimisation. Moreover, the energy conservation of the final optimised result is low, as shown in Figure 6.7. For test cases where energy conservation constraint is applied, the energy conservation will gradually increase to a high level and then becomes stable since the low energy conservation makes the error extremely high.

When handling test cases with a point light source, the initial energy conservation is low, and the error will be extremely high since it is amplified when divided by the energy conservation rate. During optimisation, the energy conservation gradually increases until it is close to the threshold, as shown in Figure 6.6(b).

COMPATIBILITY

Since our proposed method is based on a general global optimisation algorithm, it is highly compatible with many scenarios, in contrast to methods that are specifically designed for certain scenarios. For example, our method has no restriction on the light source, as long as we can generate sufficient light rays. The method proposed by Papas [6], on the other hand, only works for collimated light source.

Moreover, we are not limited to the reflector case, our algorithm naturally handles the refractive lens design problem. We only have to consider the refractive index for lens design during ray-tracing. The assumption is that we can consider the lens interface as a surface, thus ignoring volumetric effects. Figure 6.8 shows an example of the refractive lens case optimised with our method. The optimisation algorithm is also stable since it does not require a good initial guess and the results do not vary much in different runs, as shown in Table 6.2. We expect that running the optimisation twice is usually enough to avoid a "bad" run.

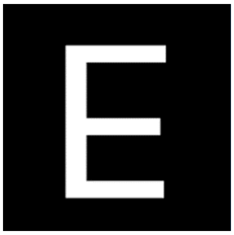
In brief, any scenario that can be ray-traced is a valid case for our method, making it highly general.

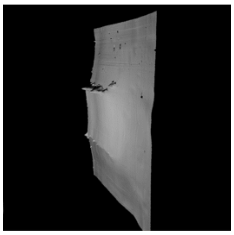
6.2.2. DISCUSSION

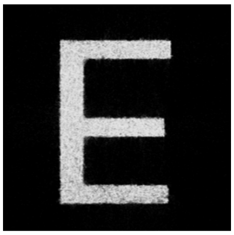
According to the analysis, the results are accurate, especially with simple target radiance distributions. The converge speed is also very reasonable given our initial goal to produce a first approximation for more complex optimisation algorithms. It can converge in a few minutes, especially for the test cases with simple patterns. Moreover, our proposed algorithm has much higher compatibility and generality than other specifically designed algorithms, enabling it to perform well in various test scenarios.

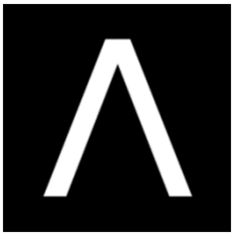
One of the main problems is that it cannot accurately result in complex radiance distributions, such as the Lenna case. When compared with other accuracy-oriented algorithms, we are still far from the optimal, as shown in Figure 6.11. Our optimiser does not thoroughly learn the correspondence between radiance distribution and geometry since our optimisation process solely relies on a random walk in the neighbour search space. We use strategies to accelerate the optimisation process, however, some of them have side-effects that might affect performance. For example, the randomisation strategy may increase the performance and compatibility for most test cases. On the other hand, it might decrease the convergence speed since some of the kernels might not contribute to the total error decrease in rare cases, although in most cases the issue can be solved by the history-based decision strategy. Further, the history-based decision strategy can increase the convergence speed. Still, it will also increase the risk of being stuck to a local minimum since some kernels might have a much higher probability than others.

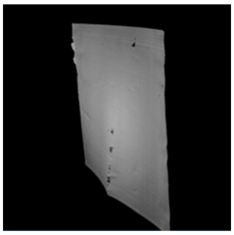
Another issue is the choice of parameters. The optimisation process has many parameters, and different choices of parameters can lead to different results. For example, the slow cooling schedule can usually lead to a more accurate solution than the fast cooling schedule, but it will also slow the convergence speed. To ensure this algorithm's effectiveness and efficiency, the parameters must be carefully chosen, including temperature, cooling schedule, kernel size, kernel shape, etc. This can be a problem for those who do not have experience tuning an optimisation process. For more experienced users, our method's parameters can be understood intuitively, and it can also give much higher flexibility to users when applied to different applications.

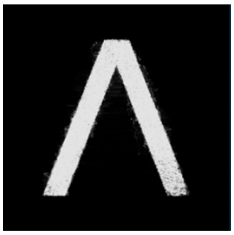

(a) Alphabet 'E', Resolution: 200×200

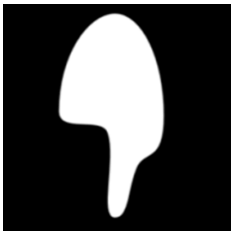

(b) Mesh size: 101×101

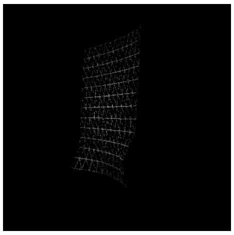

(c) Error: 0.083; Energy Conservation: 0.976; Iterations: 5.3×10^5


(d) Alphabet 'V', Resolution: 200×200


(e) Mesh size: 101×101


(f) Error: 0.087; Energy Conservation: 0.988; Iterations: 3.7×10^5


(g) Headlight, Resolution: 50×50


(h) Mesh size: 13×13

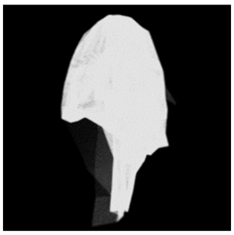
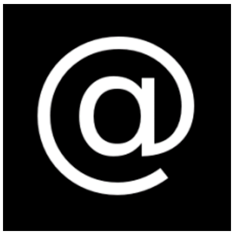
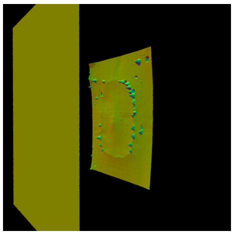
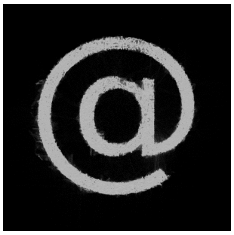

(i) Error: 0.096; Energy Conservation: 1.0; Iterations: 1.3×10^5

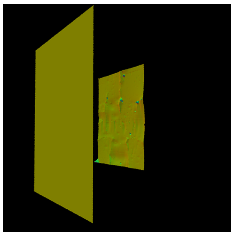
Figure 6.2: The results with simple target radiance distribution (collimated light source)


(a) Symbol '@', Resolution: 200×200


(b) Mesh size: 101×101


(c) Error: 0.086; Energy Conservation: 0.965; Iterations: 9.2×10^5

(d) Chinese character 'Light', Resolution: 200×200


(e) Mesh size: 201×201

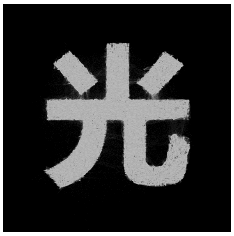
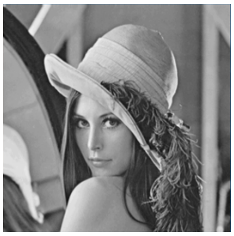
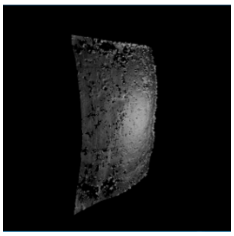

(f) Error: 0.070; Energy Conservation: 0.994; Iterations: 1.3×10^6

Figure 6.3: The results with other binary image inputs (collimated light source)


(a) Lenna (Resolution: 200×200)


(b) Mesh size: 401×401

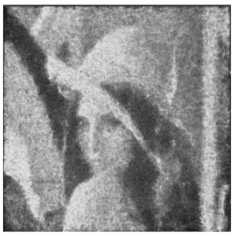
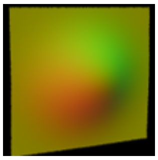
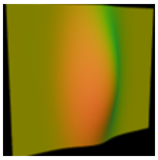
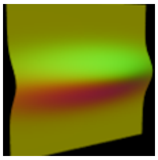
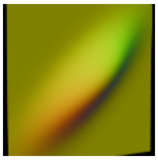

(c) Error: 0.104; Energy Conservation: 0.948; Iterations: 5.6×10^6

Figure 6.4: The results with complex target radiance distribution


(a)


(b)


(c)


(d)

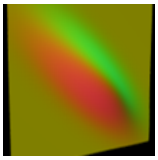
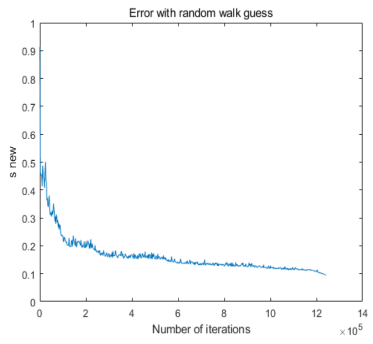
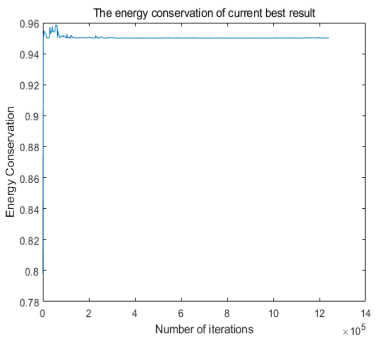

(e)

Figure 6.5: Five selected kernels used in tests of kernel weights



(a)



(b)

Figure 6.6: Reflector design case Figure 6.9: error and energy conservation

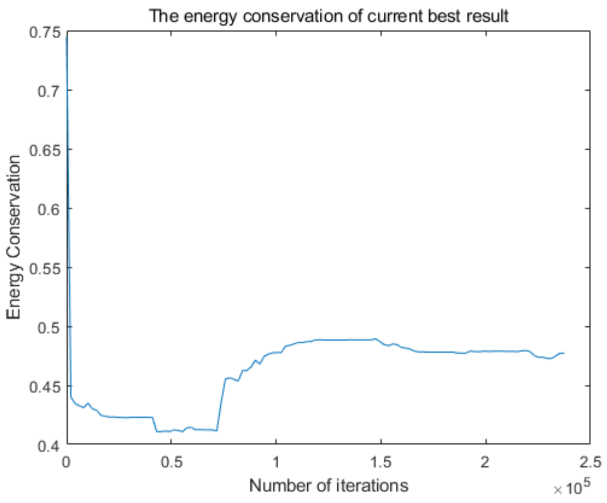


Figure 6.7: Energy conservation will fluctuate without constraint

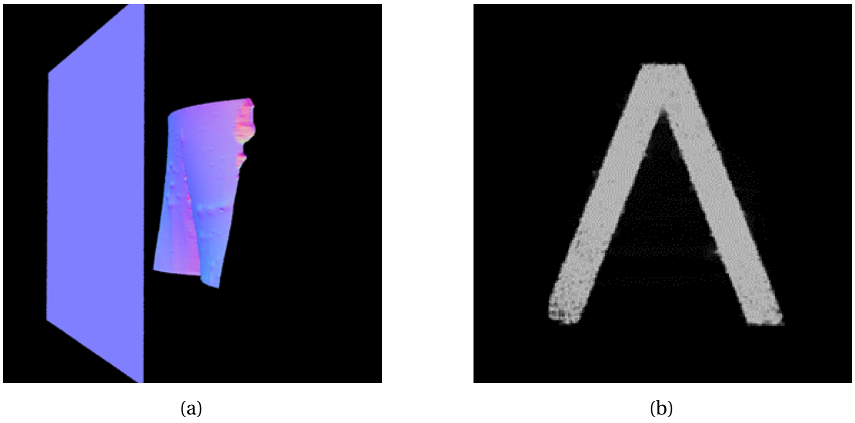


Figure 6.8: Refractive lens design (collimated light source, square initial surface)

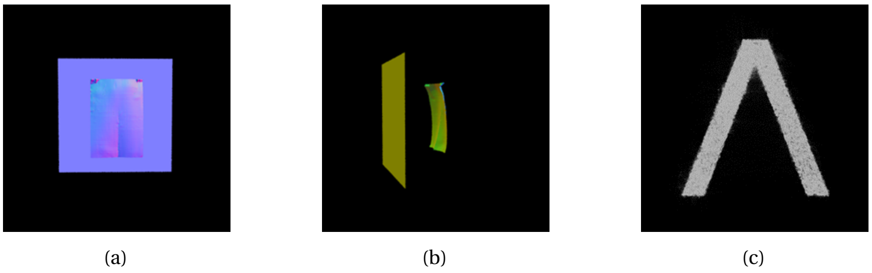


Figure 6.9: Reflector design (point light source, non-square reflector, energy threshold: 0.95)

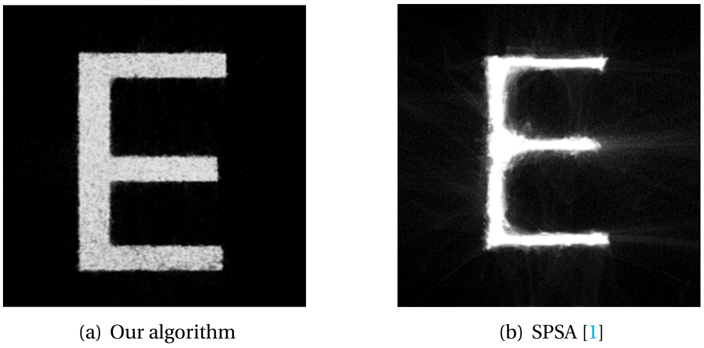
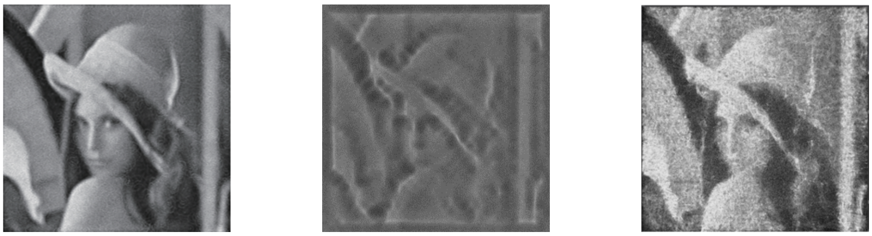


Figure 6.10: Accuracy comparison with SPSA in simple cases



(a) Gaussian decomposition + permutation [6]

(b) SPSA [1]

(c) Our algorithm

Figure 6.11: Accuracy comparison between different algorithms

7

Conclusion

7.1. CONCLUSION

This thesis proposes a general optimisation method for a fast approximation for designing reflectors and lenses. Compared with other algorithms which are also based on general optimisation methods, the proposed method has a high convergence speed and accuracy. It can obtain an accurate result when handling complex radiance distributions. Although our proposed algorithm is not as accurate as the accuracy-oriented algorithms, the result accuracy is enough to serve as an initial guess for further optimisation. Its advantage in generality is clear when compared against accuracy-oriented algorithms. Our proposed method can handle a wide range of scenarios, such as point light source, user-defined light source, refractive lens design, etc.

The design and implementation of the proposed method was also presented in detail in this thesis. Our system uses OptiX7 and CUDA to leverage the compute capability of GPUs, while the CPU is only used for workflow control and not responsible for any intensive computing tasks. To gain performance, we parallelise all the intensive computations such as ray tracing, mesh updates, error computation, etc. This system also minimises the memory exchange, memory allocation and adopts multiple strategies.

Nevertheless, our proposed system design is independent of the ray tracing framework and should be able to be implemented in any sequential-parallel computing platform.

We believe the high performance and generality of our system is a useful step towards creating faster tools for many application scenarios, such as lamp manufacture, headlight design, lens design, among others.

7.2. FUTURE WORK

We believe there is still plenty of room to further increase the performance and accuracy of our method. Below are a few possibilities that we deem particularly interesting.

EXTRACTION OF SURFACE/RADIANCE FEATURES

In the field of Computer Vision, it is common to extract features from images and analyse them. In our project, the optimiser does not learn by extracting features (except for the high-frequency features used in Sobel error) from the target radiance distribution since the main optimisation process is based on a random walk in the neighbour search space. Basically, the input radiance distribution is not thoroughly analysed before the main optimisation. Instead, it is simply used as the target radiance distribution in the error computation. By further analysing the target distribution, valuable information could be extracted to help guide the optimisation process.

Moreover, in some researches of accuracy-oriented algorithms [6] [8], it has been shown that we can find the relevance between surface patch geometry and radiance distribution. The issue of such approaches is the high overhead of computing the mapping between radiance distribution and the geometric surface. However, in the modern optics industry there is a lot of training data that has a mapping between radiance distribution and surface geometry. This data could be used to train a neural network to learn this mapping between radiance and geometry. After training, using the neural network should be fast enough to be incorporated in our method and possibly achieve faster and more accurate results than the current random walk strategy. Furthermore, it can also help find the best initial shapes of kernels used in the randomisation strategy.

Adaptive block-wise optimisation

Compared with other global optimisation problems, the inverse reflector problem has the potential to be divided into several small problems. According to research like the Gaussian decomposition used in inverse reflector problem [6], the radiance distribution has the potential to be divided into small blocks and solved, respectively. Also, the research in Mitsuba 2 [5] confirmed that even a local optimisation method like gradient descent can solve the problem. Theoretically, there must be a reflector that can give the same radiance distribution in the corresponding area. Then it is also possible to solve the whole problem by solving the sub-problems and combining the solutions. The complexity of a target radiance distribution in various blocks can differ a lot, then the small problem in different complexity can be solved with different computing intensities. In our current optimisation scheme, the degree of refinement (mesh size, resolution, kernel size) is the same for the whole surface. The search space dimensionality can be extremely high when handling complex radiance distribution, which needs a large mesh size. If the block-wise optimisation is feasible, we can then tune the refinement level for different blocks. The optimiser will apply the most suitable optimisation strategy for each block. The adaptive block-wise optimisation, if available, can significantly reduce the search space dimensionality and increase the convergence speed, especially when handling complex radiance distributions.

Material and scattering

So far, we have assumed that the surface used in the optimisation process is a perfect mirror or a lens with a fixed refractive index. Our proposed method can work well under such assumption. However, for practical applications, it might be interesting to consider different materials such as metal, plastics, etc. The generality and compatibility of the proposed method are essential in practical applications, and it should be compatible with different materials and scattering/transmittance BxDFs. For such cases, we need to take the BxDF of different materials into account, as the light rays will not reflect in a perfect-mirror way. Instead, they will scatter and transmit with respect to a certain probability density distribution or even absorbing part of radiance energy during transmitting. The problem complexity can be far higher than the perfect-mirror case.

Nevertheless, since our algorithm is based on a general global optimisation method, it is also compatible with application scenarios like this. The time needed for convergence will naturally increase as many more rays are necessary to capture the scattering effects properly.

We have tested some cases with a fuzzy factor when computing the reflected direction, which is a very simple BRDF. Still, the compatibility and generality with different materials need further research and experiment, and further modification might also be applicable if it is necessary to improve the compatibility with different materials.

BIBLIOGRAPHY

[1] M. Finckh, H. Dammertz, and H. Lensch, "Geometry construction from caustic images," vol. 6315, Sep. 2010, pp. 464–477, ISBN: 978-3-642-15554-3. DOI: 10.1007/978-3-642-15555-0_34.

[2] Wikipedia contributors, *Headlamp — Wikipedia, the free encyclopedia*, https://en.wikipedia.org/w/index.php?title=Headlamp&oldid=1013828904, [Online; accessed 27-March-2021], 2021.

[3] G. Patow, X. Pueyo, and À. Vinacua, "Reflector design from radiance distributions.," *International Journal of Shape Modeling*, vol. 10, pp. 211–236, Dec. 2004. DOI: 10.1142/S0218654304000675.

[4] J. Spall, "An overview of the simultaneous perturbation method for efficient optimization," Feb. 2001.

[5] M. Nimier-David, D. Vicini, T. Zeltner, and W. Jakob, "Mitsuba 2: A retargetable forward and inverse renderer," *ACM Trans. Graph.*, vol. 38, no. 6, Nov. 2019, ISSN: 0730-0301. DOI: 10.1145/3355089.3356498. [Online]. Available: https://doi.org/10.1145/3355089.3356498.

[6] M. Papas, W. Jarosz, W. Jakob, S. Rusinkiewicz, W. Matusik, and T. Weyrich, "Goal-based caustics," *Comput. Graph. Forum*, vol. 30, pp. 503–511, Apr. 2011. DOI: 10.1111/j.1467-8659.2011.01876.x.

[7] T. Weyrich, P. Peers, W. Matusik, and S. Rusinkiewicz, "Fabricating microgeometry for custom surface reflectance," *ACM Trans. Graph.*, vol. 28, Jul. 2009. DOI: 10.1145/1531326.1531338.

[8] Y. Schwartzburg, R. Testuz, A. Tagliasacchi, and M. Pauly, "High-contrast computational caustic design," *ACM Transactions on Graphics*, vol. 33, 74:1–, Jul. 2014. DOI: 10.1145/2601097.2601200.

[9] Y. Yue, K. Iwasaki, B.-Y. Chen, Y. Dobashi, and T. Nishita, "Poisson-based continuous surface generation for goal-based caustics," *ACM Trans. Graph.*, vol. 33, no. 3, Jun. 2014, ISSN: 0730-0301. DOI: 10.1145/2580946. [Online]. Available: https://doi.org/10.1145/2580946.

[10] C. Prins, J. Boonkkamp, J. Roosmalen, W. Ijzerman, and T. Tukker, "A monge–ampère-solver for free-form reflector design," *SIAM Journal on Scientific Computing*, vol. 36, Jan. 2014. DOI: 10.1137/130938876.

[11] S. Thede, "An introduction to genetic algorithms," *Journal of Computing Sciences in Colleges*, vol. 20, Oct. 2004.

[12] F. Glover, M. Laguna, and R. Marti, *Tabu Search*. Jul. 2008, vol. 16. DOI: 10.1007/978-1-4615-6089-0.

[13] S. Kirkpatrick, C. Gelatt, and M. Vecchi, “Optimization by simulated annealing,” *Science (New York, N.Y.)*, vol. 220, pp. 671–80, Jun. 1983. DOI: 10.1126/science.220.4598.671.

[14] Wikipedia contributors, *Metaheuristic — Wikipedia, the free encyclopedia*, https://en.wikipedia.org/w/index.php?title=Metaheuristic&oldid=1007795331, [Online; accessed 27-March-2021], 2021.

[15] A. Vlachos, J. Peters, C. Boyd, and J. Mitchell, “Curved pn triangles,” *Bi-Annual Conference Series*, vol. 2001, Feb. 1970. DOI: 10.1145/364338.364387.

[16] T. Boubekeur and M. Alexa, “Phong tessellation,” *ACM Trans. Graph.*, vol. 27, p. 141, Dec. 2008. DOI: 10.1145/1457515.1409094.

[17] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[18] S. J. Majeed and M. Hutter, “On q-learning convergence for non-markov decision processes,” Jul. 2018, pp. 2546–2552. DOI: 10.24963/ijcai.2018/353.

[19] P. Shirley, *Ray Tracing in One Weekend*, 3.0.2. 2020. [Online]. Available: raytracing.github.io/books/RayTracingInOneWeekend.html.

[20] Nvidia, *Nvidia optix 7.2 – programming guide*, https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html, 2021.

[21] Nvidia, *Cuda c++ programming guide*, https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html, 2021.