

Stochastic Tree-based Optimization for Inverse Reflector Design

Albert Mas - Gustavo Patow - Ignacio Martín

Geometry and Graphics Group

Universitat de Girona

albertm@ima.udg.edu - dagush@ima.udg.edu - imartin@ima.udg.edu

Abstract

This paper presents a new global optimization algorithm specifically tailored for inverse reflector design. In these problems, the goal is to obtain a reflector shape that produces a light distribution as close as possible to a user-provided one. The optimization is an iterative process where each step evaluates the difference between a reflector illumination and the desired one. We propose a tree-based stochastic method that drives the optimization process, using some heuristic rules, to reach a minimum below a threshold that satisfies the user-provided requirements. We show that our method reaches a solution in less steps than most other classic optimization methods, also avoiding many local minima.

1 Introduction

This paper presents a new optimization method for the inverse reflector design problem. This problem focuses on solving a problem from reflector manufacturers (e.g. street lamps or car headlights) that arises when they do not know what shape the reflector must have to produce a desired illumination. These kinds of reflectors are specular reflecting surfaces made by a pressing procedure. The reflector and the bulb light source are embedded inside a holder (see Figure 1). The light source is represented by millions of light rays, and together with the holder, represent initial constraints specified by the manufacturer. The usual adopted solution is to produce a large set of reflectors and test each one in an iter-

ative process until the desired illumination is obtained. This method has high manufacturing costs, so it has to be done by experienced users with a deep empirical knowledge.

In recent years there have been some proposals to solve this kind of problems. These solutions are focused on computing the reflector lighting in a very fast way, allowing to check a huge set of reflectors in a short time. This checking is performed using classic optimization methods, like brute force algorithms. These methods define a function that evaluates the current reflector shape and returns a difference value between the current and the desired illumination, trying to obtain a global minimum value. However, this solution implies high computational costs due to the size of the search space.

We propose a new specific optimization algorithm that allows to reach the global minimum in a fast way. A tree is constructed on the fly, where each node is the evaluation of a reflector shape. We use a GPU based ray-tracing algorithm to calculate the illumination of each reflector and to evaluate the difference function. The new tree branches are constructed from the stochastically selected nodes following heuristic rules. The goal is to successively refine the reflector shape in regions where it produces the best results and where it better approaches the global minimum, avoiding to fall on local minima. When we are close to the solution, a classic local search optimization algorithm is started to faster reach the minimum.

The rest of the paper is organized as follows. We discuss the previous work in Section 2. We

present the problem formulation in Section 3, following an overview of method in Section 4. Next, in Section 5 it is described the algorithm to construct the stochastic tree. Then we show the results in Section 6 and discuss them in Section 7. Finally, conclusions are presented in Section 8.

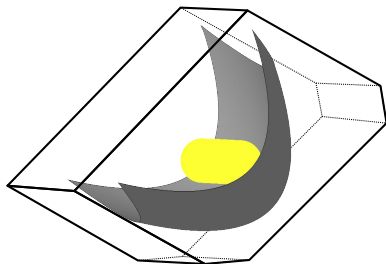


Figure 1: General configuration of a reflector and a light source inside the holder box.

2 Previous Work

The method presented here is specifically intended to solve the inverse reflector design problem. This kind of problems can be classified as an Inverse Geometry Problem (IGP) [13] in the context of inverse lighting problems, where we have to find the reflector shape that produces the desired lighting distribution. In [3] and [10], they presented two solutions for local lighting cases. [12] and [14] presented a solution for global illumination, but with many drawbacks, such as only being able to work with reflectors with low geometrical resolution, non-realistic bulbs and high computational costs in time that required several minutes for computing the light distribution of a single reflector. More recently, [1] presented a solution for NURBS-based reflectors. In [9] it was presented a new GPU-based method to compute the outgoing light distribution of a reflector working completely in the GPU, and computing millions of rays thanks to a hierarchical structure for high resolution reflector geometry. Although the method provided in [9] was faster than the one in [12], being able to compute the lighting distributions in just

a few seconds, the optimization algorithm remained the same. In both approaches, a set of reflectors are checked to get the best one with a brute force algorithm, which is a slow generic method to minimize a function.

There are many local optimization methods to minimize a function. The most commonly used is the conjugate gradients algorithm [15] that calculates function derivatives to know in which axis the function decreases faster. The Tabu Search algorithm [4] looks for the solution avoiding repeating already checked search paths, managing a list of tabu cases. The Hooke & Jeeves algorithm [5] is a kind of conjugate gradients algorithm, but avoiding the calculation of the derivatives, replacing them with a few function evaluations for each optimization step.

There are some methods that use knowledge of the function to try to reach a solution faster. The algorithms based on perturbation theory apply small offsets to parameters of a function that is difficult to solve, transforming it to a polynomial form. The linear and non-linear programming methods optimize an objective function using constraints on different parts of this function. But none of these methods guarantee the convergence in a finite time to the optimal solution of a non-convex problem like the one we are focusing here. Also, we do not have any a priori knowledge of the function.

On the other side, there are global optimization algorithms. These can be classified as deterministic, stochastic or heuristic methods. The Branch & Bound algorithms [7] is a deterministic kind of global optimization method that creates a tree, where each node represents a function parameter bound that is splitted to create the children nodes. This algorithm needs a function estimator to compute each node bound and discard tree branches. Therefore it also needs knowledge of the function to calculate a tight estimator. The Stimulated Annealing [6] is a stochastic kind of global optimization method. This is a variant of the Metropolis algorithm, so it is based in the movement of the current solution in the search space to a neighbour only if it is bet-

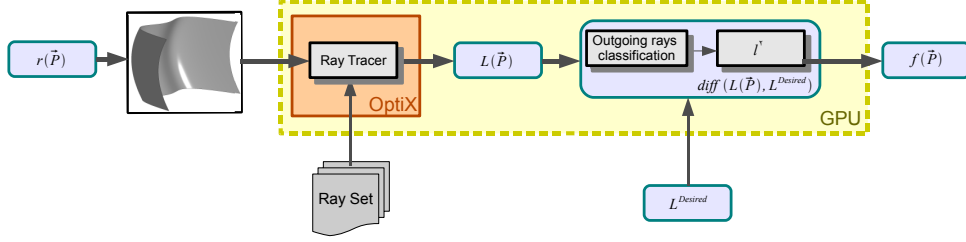


Figure 2: Scheme of the node evaluation system.

ter than current one. This movement is based on the combination of all function parameters, so there can be a huge number of combinations. The Tree Annealing algorithm [2] is a variant of Stimulated Annealing, and is used to fit noisy data, and converge to the solution in a piece-wise function. The heuristic methods use some known information about the function to increase the efficiency of a deterministic or stochastic method. There are other methods that combine classic ones, like the Clustering methods [8], that create clusters on the search space, finding local minima, and choosing the best one.

The algorithm proposed in this paper belongs to the stochastic family of global optimization methods, as it uses heuristics to reach in a fast way the optimal solution.

3 Problem formulation

In this section we will show the problem formulation, defining the function to be optimized and explaining the function evaluation system.

3.1 Function to optimize

The goal of the optimization is to obtain the reflector that produces a light distribution as close as possible to the desired light distribution. The reflector shape is created using a function $r(\vec{P})$ from a predefined family of functions, such that

$$r :: \mathbb{R}^N \rightarrow \mathbb{S} \quad \vec{P} \in \mathbb{R}^N$$

where $\vec{P}$ is a vector of N parameters in the function domain, and $\mathbb{S}$ is the space of reflector shapes. We define the function $l(r(\vec{P}))$ that returns the light distribution of reflector defined by $\vec{P}$, such as

$$l :: \mathbb{S} \rightarrow \mathbb{L}$$

We also define

$$L(\vec{P}) \equiv l(r(\vec{P})) \quad L :: \mathbb{R}^N \rightarrow \mathbb{L}$$

where $\mathbb{L}$ is the domain space of light distributions. We define a function f that returns the difference between a given reflector light distribution and a desired light distribution, such that

$$f(\vec{P}) = \text{diff}(L(\vec{P}), L^{\text{Desired}})$$

$$f :: \mathbb{R}^N \rightarrow \mathbb{R}$$

where L^{Desired} is the desired light distribution, and $\text{diff}(A, B) :: \mathbb{L}^2 \rightarrow \mathbb{R}$ is the difference between the light distributions A and B. The difference is computed in l^2 metric, such as

$$\text{diff}(A, B) = \sqrt{\sum_i^M (A_i - B_i)^2} \quad A, B \in \mathbb{L}$$

where M is the number of directional components of the discretization of $\mathbb{L}$. A_i and B_i are the light distribution energies of A and B for the directional component i of $\mathbb{L}$. Therefore, we have to calculate a value $q \in \mathbb{R}^N$ for the domain of f in order that

$$q = \min_{\vec{P}} \{f(\vec{P})\}$$

3.2 Reflector Evaluation

To evaluate f we use an algorithm with three steps (see Figure 2). First, the reflector is calculated, evaluating r . Second, the illumination is calculated from the reflector and a given light source computing L . Third, the desired and resulting light distribution are compared using $diff$.

Since we need to calculate f many times, the evaluation must be fast enough. The most time consuming step is the lighting calculation $L(\vec{P})$, because we are using light sources with millions of rays. Therefore, an optimized ray tracing algorithm, from the light source bulb to the reflector, is a good way to perform this step rapidly and accurately.

We use a ray tracing engine based on NVIDIA® OptiX™ system [11]. This is a programmable pipeline to achieve fast ray tracing results on the GPU. OptiX implements some optimized geometry structures for ray tracing on the GPU, such as kd-trees or BVHs, and incorporates algorithms for traversing these structures in a fast way.

After $L(\vec{P})$ is calculated, the light distribution rays are classified in a histogram. This histogram is compared with the previously created histogram from $L^{Desired}$ using $diff$. The differences are computed in l^2 metric. Both steps are also calculated on the GPU.

The overall system permits the evaluation of a reflector in less than one second.

4 Overview

The objective of this algorithm is to obtain a reflector that produces a light distribution as close as possible to the desired light distribution in a rapid way.

The algorithm searches parameters in $\mathbb{R}^N$ space for the best combination such that it produces a reflector that minimizes the function f . To do it, the method is based on the construction of a binary tree, where each tree node represents a reflector where the function f is evaluated. In each tree construction step, an already created node is chosen in a stochastic way, using heuristics based on node

evaluation and current statistical information of surrounding tree nodes. The chosen node is replaced by two children nodes that are created. These nodes contain all the same parameter ranges as their parent except for one, that is splitted in two subranges. For each new nodes, we use an heuristic, based on its ancestors, to choose what parameter range to split. Since we evaluate the nodes after they are created, and because we are using a greedy breadth-first search algorithm, we do not need to construct the full tree structure. The process stops when a node evaluation result is below a user termination threshold. However, when the chosen node is close enough to a minimum, or when the parameter space size is smaller than a user defined threshold, a local optimization method is used to converge in a rapid way. In this case, the chosen node becomes a tree leaf. If the minimum found by the local optimization process is under the user termination threshold, the process stops. Otherwise, the process continues choosing a new tree node.

5 Tree construction

As mentioned in the previous section, the tree is lazily created. On each optimization algorithm iteration, a tree branch is chosen depending on a greedy breadth-first search algorithm. It depends on two stochastic selections: the node selection and the selection of parameter space to split. To better guide the process in order to obtain good rates in the optimization convergence, we use a set of heuristics on both stochastic selections.

Now we are going to explain the node selection process, and after that the parameters splitting mechanism.

5.1 Node selection from the tree section

Each time a node is created it is stored into a list called “tree section”, which is a sorted list where all current tree leaves are stored and sorted by their evaluation values. When a node is selected, it is removed from the list, and the two new children nodes are created

and stored into the list.

The tree section objective is to help on the decision of what node to choose to continue the optimization. The simple node evaluation is not enough to decide if we are approaching to the global minimum or not. We should also consider the differences between the current evaluation and the last evaluated values to get information about how we are approaching a minimum. But we cannot know if this is a local or global minimum. However, we can use the information of already created nodes to know how the function f behaves on different parameter ranges, and decide what domain zone is more suitable to be a global minimum.

For each tree section node, we calculate a weight value. Then, the list is sorted by the node evaluation values multiplied by their respective weights. This weight is the linear combination of three weights, that are estimations of how close the current node is to a minimum.

The first weight, named w_{diff} , represents the difference between the current node and the parent node evaluation values, so

$$w_{diff_i} = \frac{m_{ax}(f(P_{par_i}) - f(P_i), 0)}{m_{ax}Val} \cdot w_{eight}D_{iff}$$

where $f(P_{par_i})$ is the evaluation of the parent node, $m_{ax}Val$ is the maximum accepted node evaluation value, and $w_{eight}D_{iff}$ is a factor for controlling the importance of this difference. The nodes with greater evaluation values than $m_{ax}Val$ are stored at the end of the tree section with a weight value of 0. We cannot simply discard these nodes because the parameter values used only are representative values of their ranges. Therefore, these nodes are considered after all the other suitable branches are processed. For the other nodes, if w_{diff} is high it means that $f(P_i)$ is on a steep slope of the function, thus we are approaching faster to a minimum (see Figure 3)

The second weight, named $w_{density}$, represents the density of nodes around the current one, so

$$w_{density_i} = \left(1 - \frac{n_{ear_i}}{N_{TreeNodes}}\right) \cdot w_{eight}D_{ensity}$$

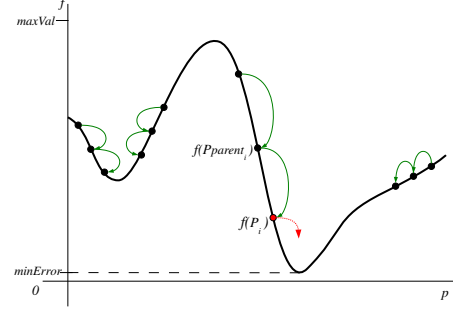


Figure 3: The w_{diff} weight allows the selection of nodes that are approaching faster to a minimum

where n_{ear_i} is the number of nodes in a neighborhood of node i (we get the nodes using an Euclidean measure distance and a user specified maximum distance), $N_{TreeNodes}$ is the number of current tree nodes, and $w_{eight}D_{ensity}$ is a factor for controlling the importance of this density. If the density is high, it means that the current parameter range contains a lot of evaluated nodes, so $w_{density}$ should be low because we consider that the region has been evaluated enough (see Figure 4).

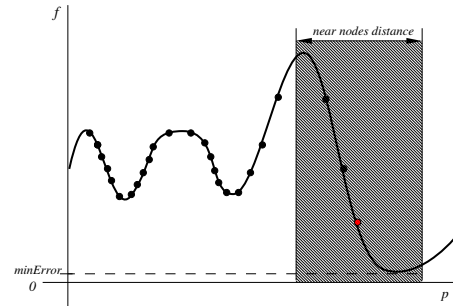


Figure 4: The nodes with less near nodes have higher $w_{density}$ values.

The third weight, named w_{stdv} , represents the variance on evaluation values of nodes

around current one, so

$$w_{stdev_i} = \left(\frac{stdev_i}{n_{ear_i} \cdot (m_{ax} V_{al} \cdot m_{ean_i})^2} \right) \cdot w_{ight} S_{tDev}$$

where $stdev_i$ is the standard deviation of the evaluation values of the near nodes, m_{ean_i} is the mean of evaluation values of the near nodes, and $w_{ight} S_{tDev}$ is the factor to control the importance of this variance. If the current parameter domain region has low variance, it probably means that the function is nearly flat in this region, so it might imply there is not any minimum. In this case, w_{stdev} should to be low (see Figure 5).

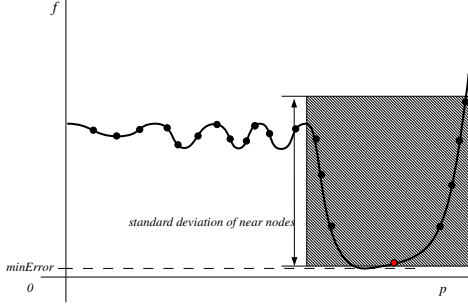


Figure 5: The nodes with greater standard deviation on near nodes have higher w_{stdev} values.

The final composed value is normalized to avoid biased data, so the weighted evaluated value v_i for node i is

$$v_i = \left(1 - \frac{f(P_i)}{m_{ax} V_{al}} \right) \cdot w_i \quad (1)$$

$$w_i = \frac{w_{diff_i} + w_{density_i} + w_{stdev_i}}{w_{ight} D_{iff} + w_{ight} D_{ensity} + w_{ight} S_{tDev}}$$

The node selection is performed stochastically by importance sampling. Note that this random selection avoids focusing the optimization progress only on one tree branch, so the optimization always considers a set of possible minima at the same time.

5.2 Node param space splitting

For each parameter, a range is defined and stored in each node. Initially, these ranges are

the domain of each parameter space, that are taken from the initial constraints for the $\mathbb{R}^N$ space. Since we need a $\vec{P} \in \mathbb{R}^N$ to evaluate f , the central values of each parameter range are used.

$$\vec{P} = \{p_0, \dots, p_i, \dots, p_N\}$$

$$p_i = \frac{d_{i_{min}} + d_{i_{max}}}{2} \quad p_i \in [d_{i_{min}}, d_{i_{max}}]$$

where $d_{i_{min}}$ and $d_{i_{max}}$ define the range of parameter p_i .

When a node is selected from the “tree section”, two new children nodes are created. Then, a parameter p_j is chosen stochastically from $\vec{P}$, and his range is splitted in two equal parts, creating two new vectors named $\vec{P}_{left}$ and $\vec{P}_{right}$, that are assigned to the created children nodes.

$$\vec{P}_{left} = \{p_0, \dots, p_{j_{left}}, \dots, p_N\}$$

$$\vec{P}_{right} = \{p_0, \dots, p_{j_{right}}, \dots, p_N\}$$

$$p_{j_{left}} = \frac{d_{j_{min}} + p_j}{2} \quad p_{j_{left}} \in [d_{j_{min}}, p_j]$$

$$p_{j_{right}} = \frac{p_j + d_{j_{max}}}{2} \quad p_{j_{right}} \in [p_j, d_{j_{max}}]$$

We use an heuristic based on node ancestors to choose p_j . For each node we store a vector $\vec{H}$, named “history vector” of size N , where each entry contains information about one parameter. This vector provides the information of the minimization progress in function of previously chosen parameters in the same tree branch. Initially, a random vector index is chosen from the root node history vector to split the parameter range of the first children nodes. Next, when new children nodes are created and evaluated, the history vectors of each one are cloned from its parent node. Then, the vector entry with same index than the previous chosen one on the parent node is updated with the difference between the current and parent node weighted evaluation value $v(\vec{P})$ (see Equation 1) :

$$\vec{H}_{par} = \{h_0, \dots, h_k, \dots, h_N\}$$

$$\vec{H}_{left} = \{h_0, \dots, h'_{k_l}, \dots, h_N\}$$

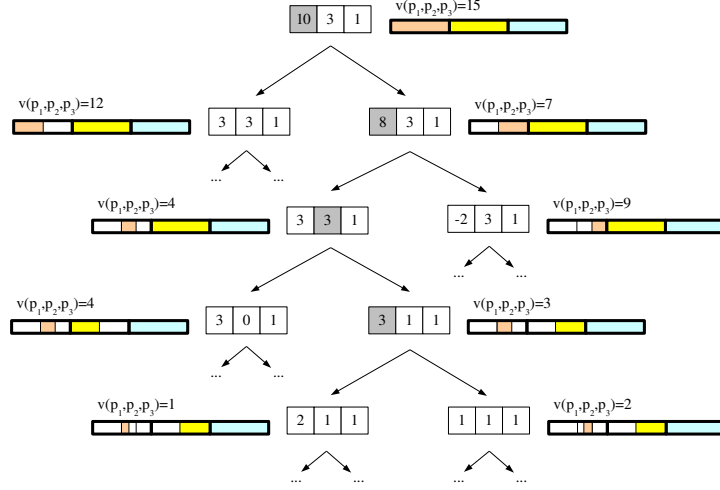


Figure 6: Example of parameter selection with 3 parameters. The color cells are a representation of the parameter ranges. The function $v(p_1, p_2, p_3)$ is the node weighted evaluation on the three parameters. The gray vector cells are the chosen parameters for each node.

$$\overrightarrow{H_{right}} = \{h_0, \dots, h'_{k_r}, \dots, h_N\}$$

with

$$h'_{k_l} = v(\overrightarrow{P_{par}}) - v(\overrightarrow{P_{left}})$$

$$h'_{k_r} = v(\overrightarrow{P_{par}}) - v(\overrightarrow{P_{right}})$$

where $\overrightarrow{H_{par}}$ and $\overrightarrow{P_{par}}$ are the history and parameter vectors of parent node, $\overrightarrow{H_{left}}$, $\overrightarrow{P_{left}}$, $\overrightarrow{H_{right}}$ and $\overrightarrow{P_{right}}$ are the history and parameter vectors of the left and right child nodes respectively, and k is the previous selected parameter index of parent node.

To choose the parameter range to split, we use the history vector to perform a stochastic selection by importance sampling, giving more priority to parameters with greater importance values. Therefore, we give more importance to parameters that previously have had greater differences with their parents, which means that the optimization progress with these parameters approaches faster to the minimum.

In Figure 6 there is an example of the parameter selection. The example shows a tree parameter selection with $N = 3$. At the root node the first parameter is chosen and

split into two new children nodes named left and right nodes (each one contains $\overrightarrow{P_{left}}$ and $\overrightarrow{P_{right}}$ respectively). Then, the two new nodes are evaluated, and the differences between the weighted values of each one and its parent are stored in the first history vector entry of each new node. When the right node is chosen from the tree section list, the first parameter is selected again and split to create its two new children. When the new left node is chosen in turn, the first history vector entry value is smaller. This means the previously chosen parameter does not carry significant changes with respect its parent, so we are not approaching to a minimum in a fast way. In this case, the algorithm chooses other parameters that previously produced similar or better approaches to a minimum, selecting stochastically the second history vector entry, and splitting the second parameter.

5.3 Local search

As we already mentioned, there are many methods to perform a local search, like the

conjugate gradients algorithms. We have chosen the Hooke and Jeeves method [5] [1] because it is fast, like other conjugate gradient algorithms, and it is not necessary to calculate the derivatives of the function, replacing them by a few function evaluations. Moreover, this method is fast achieving the minimum if starting values are near it. When the selected node error is near to the user defined threshold, the local search process starts using the current minimum error node parameter values. The process stops when it reaches the global minimum, or when the offset is small enough to be able to address that the current local minimum is not the desired minimum. If the desired minimum is not found, the whole tree node is discarded.

6 Results

We have tested our method with three test-cases. The first one, called *Model A*, uses the basis function

$$r(\vec{P}, x, y) = p_0x^2 + p_1y^2 + p_2 \quad (2)$$

to construct the reflector, where $x, y \in [-0.5, 0.5]$ to match the holder boundary. The light source is a sphere of radius 0.05 mm placed at (0, 0, 0) mm. The emittance is a cosine distribution with 1100 lumens. The second test-case, called *Model B*, uses the basis function

$$r(\vec{P}, x, y) = p_0 * (-|\sin(y/\pi) * (p_1 * \sin(x/\pi) - p_3 * (1 - |\cos(p_4 * x/\pi)|))|) \quad (3)$$

to construct the reflector, where $x, y \in [0, 10]$ to match the holder boundary. The light source is a sphere of radius 0.5 mm placed at (5, 5, -0.5). The emittance is a cosine distribution with 2500 lumens. The third test-case, called *Model C*, uses 7 control values to define the geometry shape of a real-case reflector. The light source is a cosine emittance of 28000 lumens on a cylinder of length 65 mm and radius 4 mm, placed along the main reflector axis.

The desired lighting and optimization results for each case are shown in figures 7, 8

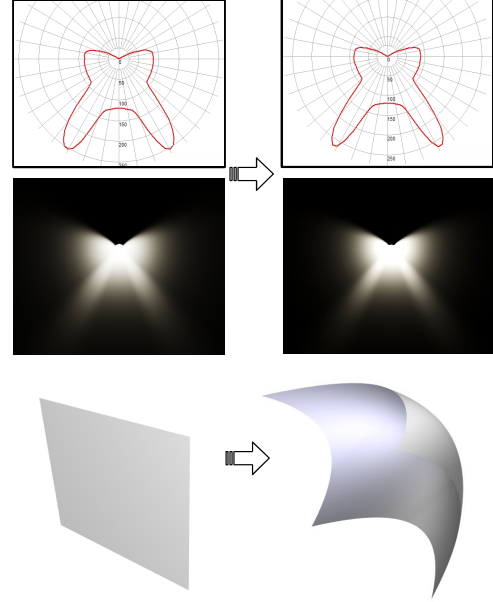


Figure 7: Results for our *Model A*. At the left, the desired lighting. At the right, the optimization result. From the top to the bottom, the IES representations, a render projection of lighting distribution, and the reflector shapes. These shapes are constructed using the parametric function (2).

and 9 respectively. For each one is shown the desired and resulting lighting in IES format. Also, a render of the lighting projection is shown to help to better understand how the lighting is. Finally, we can observe the reflector shape with the initial parameter values and the obtained parameters after the optimization. The differences between desired lighting and obtained lighting are quite small, in all cases below 4% (see Table 1)

The overall optimization times, number of processed reflectors and relative errors are shown in Table 1. It also shows the number of local searches processed. We have used a 2×10^6 ray set size for *Model A*, and 1×10^6 for the others. The lighting calculation for each reflector lasts, in average, around 200 ms for all cases. The evaluation of l^2 for each final lighting distribution lasts around 70 ms in average. Considering some extra time to com-

Model	Reflector triangles	Parameters	Tested reflectors	Local searches	Optimization time (sec.)	Relative error (%)
A	19796	3	637	1	182	1.08
B	19404	4	3172	2	770	0.32
C	16200	7	1431	2	390	3.7

Table 1: Optimization results for all tested cases.

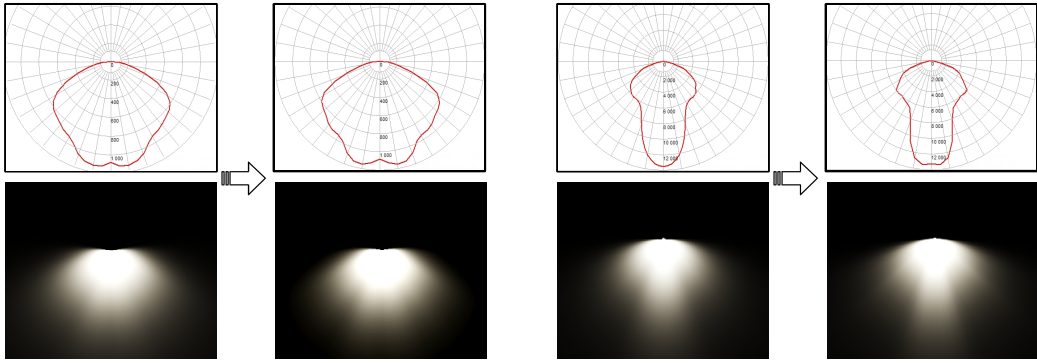


Figure 8: Results for our *Model B*. The image structure is the same than Figure 7. The reflector shapes are constructed using the parametric function (3).

pute the reflector shape, less than 300 ms are needed in average to compute the final $f(\vec{P})$ value.

The same cases are tested using a brute force optimization algorithm [9]. The results are shown in Table 2. The number of tested reflectors and, in consequence, the time to reach a solution are exponentially related with the number of parameters, as the brute force algorithm tests all parameter combinations. Also the relative errors tend to be larger because the local search method used in our algorithm approaches to the minimum with more preci-

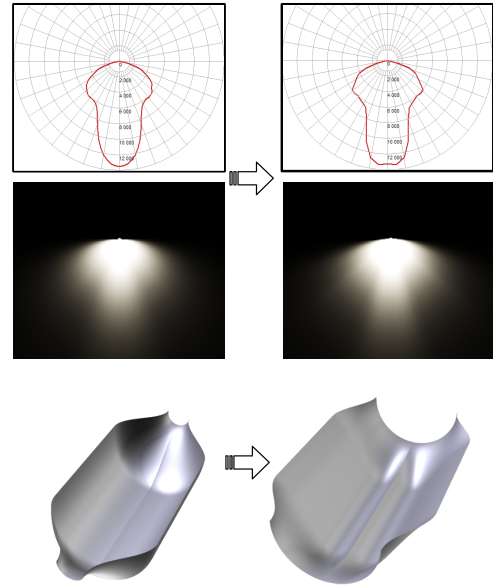


Figure 9: Results for our *Model C*. The image structure is the same than Figure 7. The reflector shapes are constructed modifying 7 control parameters. This kind of reflector is based on a real case.

sion than what is possible with the the brute force algorithm.

Model	Tested reflectors	Optimiz. time	Relative error (%)
A	39965	2h 52m	1.09
B	75486	4h 43m	0.54
C	772677	53h 21m	11.62

Table 2: Optimization comparison with Table 1 using a brute force optimization algorithm.

7 Discussion

As shown in the previous section, the optimization is very fast, and returns reflectors that generate lighting distributions very close to the desired ones. The main bottleneck of the method is the ray tracing calculation for each reflector. On the other hand, the OPTIX system has a very good ray tracing performance in comparison with other methods. Therefore, we have to focus on the optimization tree construction and nodes selection.

The key of the optimization method is the node selection and splitting. The weighted evaluations gives us an heuristic way to do it. It is important to cover all the parameter space domain, but focusing on suitable minima. This way, this is not too far from the kd-tree axis subdivision space idea, so we can guarantee that all of parameter domain space is covered. The selection of user thresholds and weights is relevant. We believe that more heuristics could be used to help the user to choose these control values.

8 Conclusions and Future Work

We have presented an optimization method for a inverse reflector design problem. The method is based on stochastic tree, driven by heuristic rules. Each optimization step evaluates the difference between current reflector shape illumination and desired one. This evaluation is done using a fast GPU raytracing method. The minimum that satisfies the user provided requirements is reached in minutes and with less evaluations than with most other classic optimization algorithms. Also, many local minima are avoided.

We consider, as future work, to parallelize the optimization algorithm, using GPU tools such as CUDA, to allow the processing of multiple tree-branches at time.

9 Acknowledgments

This work was done under grant TIN2007-67120 from the Spanish Government, and under grant 7th Framework Programme-Capacities (grant No 222550, project

EcoStreetLight) from the UE.

References

- [1] Oscar Anson, Francisco J. Seron, Diego Gutierrez, *NURBS-based inverse reflector design*, Congreso Español de Informatica Grafica, pp 65-74, 2008
- [2] Bilbro, G.L. and Snyder, W.E., *Optimization of functions with many minima*, IEEE Transactions on Systems, Man and Cybernetics, Vol. 21, Num. 4, pp 840-849, 1991
- [3] Luis A. Caffarelli and Sergey A. Kochengin and Vladimir I. Olier, *On the Numerical Solution of the Problem of Reflector Design with Given Far-Field Scattering Data*, Contemporary Mathematics, Vol. 226, pp 13-32, 1999
- [4] Rachid Chelouah and Patrick Siarry, *Tabu Search applied to global optimization*, European Journal of Operational Research, Vol. 123, Num. 2, pp 256-270, 2000
- [5] Hooke, Robert and Jeeves, T. A., “ *Direct Search*” *Solution of Numerical and Statistical Problems*, J. ACM, Vol. 8, Num. 2, pp 212-229, 1961
- [6] S. Kirkpatrick and C. D. Gelatt and M. P. Vecchi, *Optimization by Simulated Annealing*, Science, New Series, Vol. 220, Num. 4598, pp 671-680, 1983
- [7] A.H. Land and A.G. Doig, *An automatic method for solving discrete programming problems*, Econometrica, Num. 28, pp 497-520, 1960
- [8] Aristidis Likas and Nikos Vlassis and Jakob J. Verbeek, *The global k-means clustering algorithm*, Pattern Recognition, Vol. 36, Num. 2, pp 451-461, 2003
- [9] Albert Mas and Gustavo Patow and Ignacio Martin, *Fast Inverse Reflector Design (FIRD)*, Computer Graphics Forum, Vol. 28, Num. 8, pp 2046-2056, 2009
- [10] Andreas Neubauer, *Design of 3d-reflectors for near field and far field problems*, Springer, 1997
- [11] NVIDIA, *NVIDIA OptiX Ray Tracing Engine - Programming Guide*, <http://developer.nvidia.com/object/optix-home.html>, 2009
- [12] Gustavo Patow and Xavier Pueyo and Àlvar Vinacua, *Reflector Design From Radiance Distributions*, International Journal of Shape Modelling, Vol. 10, Num. 2, pp 211-235, 2004
- [13] Gustavo Patow and Xavier Pueyo, *A Survey of Inverse Surface Design From Light Transport Behavior Specification*, Computer Graphics Forum, Vol. 24, Num. 4, pp 773-789, 2005
- [14] Gustavo Patow and Xavier Pueyo and Àlvar Vinacua, *User-guided inverse reflector design*, Computer and Graphics, Vol. 31, Num. 3, pp 501-515, 2007
- [15] William H. Press, *Numerical recipes : the art of scientific computing*, Cambridge University Press, 2007