

Technical Section

User-guided inverse reflector design

Gustavo Patow^{a,*}, Xavier Pueyo^a, Alvar Vinacua^b

^aGrup de Gràfics de Girona, Institut d'Informàtica i Aplicacions, Universitat de Girona, Campus de Montilivi, E-17003 Girona, Spain

^bLlenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, E-08035, Barcelona, Spain

Received 12 December 2005; received in revised form 31 October 2006; accepted 18 December 2006

Abstract

This paper proposes a technique for the design of luminaire reflector shapes from prescribed optical properties (far-field radiance distribution), geometrical constraints and user's knowledge. This is an important problem in the field of Lighting Engineering, more specifically for Luminaire Design. The reflector's shape to be found is just a part of a set of pieces called in Lighting Engineering an optical set. This is composed of a light bulb (the source), the reflector and usually a glass that acts as a diffusor for the light, and protects the system from dust and other environmental phenomena. Thus, we aim at the design and development of a system capable of generating automatically a reflector shape in a way such that the optical set emits a given, user-defined, far-field radiance distribution for a known bulb.

In order to do so, light propagation inside and outside the optical set must be simulated and the resulting radiance distribution compared to the desired one. Constraints on the shape imposed by industry needs and expert's knowledge must be taken into account, bounding the set of possible shapes. The general approach taken is based on a minimization procedure on the space of possible reflector shapes, starting from a user-provided starting shape. The algorithm moves towards minimizing the distance, in the l^2 metric, between the resulting illumination from the reflector and the prescribed, ideal optical radiance distribution specified by the user. The initial shape and a provided confidence value are used during the whole process as a boundary for the space of spanned reflectors used during the simulation.

© 2007 Elsevier Ltd. All rights reserved.

Keywords: Inverse problems; Surface modelling; Reflector design

1. Introduction

Finding the shape of a reflector from a given set of required optical properties and geometrical constraints is of high importance in the field of Lighting Engineering (see [1–3] for an introduction to the subject). In this field, most of the effort is spent in building a reflector and testing it, perhaps by simulating its optical properties as a whole, but more often by building a physical prototype of the reflector where measurements are carried out, discarding it if it does not match the specifications, building another, testing it again, and so on. This process is extremely expensive and time-consuming.

Some tools have been developed to face the problem of reflector shape design for Lighting Engineering, but, in general, they follow the build-test-restart principle used by traditional engineering [4–6], which is an extremely costly and painful process, even when the designers already have a reasonably good idea of what the final reflector shape will be.

The problem of Inverse Reflector Design, can be stated as follows (see Fig. 1): given a light source (bulb) with a known luminous intensity distribution, a reflector surface should be constructed in such a way that a prescribed illumination intensity is obtained in certain region in space, after reflection of the emitted light on the surface. In Fig. 1 we can see the desired light distribution shown at the left, and the algorithm should produce the surface shown in the middle, which generates (with the given bulb) the light distribution shown on the right. This prescribed distribution is given as a far-field distribution, which means that it is given in purely directional terms.

*Corresponding author. Tel.: +34 972 418832; fax: +34 972 418792.

E-mail addresses: dagush@ima.udg.edu (G. Patow), xavier@ima.udg.edu (X. Pueyo), alvar@lsi.upc.edu (A. Vinacua).

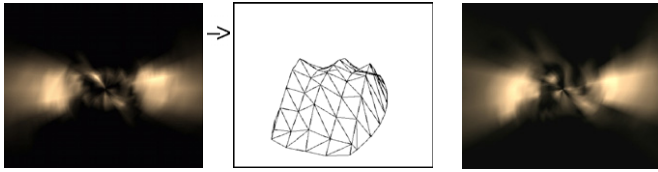


Fig. 1. Example of an inverse surface design problem.

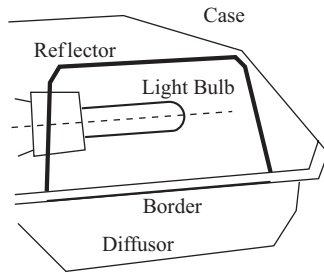


Fig. 2. An optical set.

The reflector shape sought is a part of what is called in Lighting Engineering an *optical set*, which consists of a light bulb, the reflector and the diffuser (Fig. 2). The reflector has a border, contained in a plane, that limits its shape. In general, a reflector must fit inside a holding bounding box, so its shape cannot be lower at any point than the plane defined by the border nor higher than this bounding box.

In general, luminaries incorporate reflectors and refractive diffusers that require physical BRDF (Bi-directional Reflectance Distribution Function) and BTDF (Bi-directional Transmittance Distribution Function) measurements and advanced ray tracing support for repetitively structured optics such as prismatic diffusers, but our objective in the present work is aimed towards the simplest configuration for an optical set, consisting of a light bulb and a reflector surface, which is a common configuration for illumination settings, for instance, at streets and general open spaces. This also includes the case of non-refractive nor scattering diffusers.

In this paper we will focus on the following problem: In view of the outgoing radiance distribution of a light bulb and a reflector border, find the shape for the reflector whose resulting illumination matches a given outgoing radiance distribution. Do this below a user-defined threshold of the maximum allowable error, and taking into account a suggested initial shape plus its confidence values (Fig. 3).

The following constraints are imposed on the shape of the surface to be built:

- (1) The shape must satisfy constructive constraints that require that the shape of the reflector be the graph of a function with respect to the plane of the reflector's border.

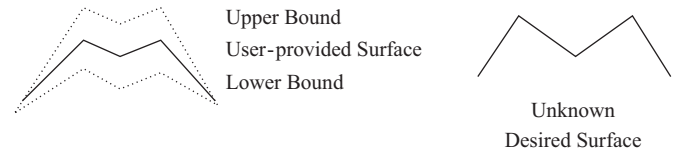


Fig. 3. The user-provided shape plus its confidence values, and the desired shape.

- (2) The resulting shape must exactly fit the given border.
- (3) The shape cannot be lower than the border plane ($z = 0$), or higher than a certain maximum height.
- (4) The user-suggested initial shape plus its confidence values must be used as a bounding region for the space of possible reflectors.

The paper is organized as follows: In Section 2 we present the previous work done in the field so far, and compare it against our solution. In Section 2.2 we give the problem formulation, and in Section 3 we make an overview of the proposed solution. In Section 4 we explain this solution in detail, and in Section 5 we discuss the results obtained. Finally, in Section 6 we present our conclusions and outline approaches for future work.

2. Previous work

2.1. Inverse problems

The central problem of the paper can be put in the context of inverse problems. In illumination simulation, inverse problems are those where we know the effect of the illumination and we have to compute some of the parameters that produce this effect. These include problems such as:

- light source (bulb) positions and their orientations [7,8], where the objective is to find the locations of known light sources in order to achieve a desired illumination.
- luminaire emittance [9–13], where the emittances of already placed light sources are computed to obtain a desired illumination on some surfaces that define the scene.
- surface characteristics of some relevant scene elements [14], that is, finding the parameters of BRDF on prescribed surfaces of the scene [11,12,15,16] to produce the desired effects in the environment.
- the shape or position and orientation of the reflectors in the scene [17–21].

The common characteristic of this kind of problems is that, in general, we know in advance the desired effect of the illumination at some regions of the scene (their final radiance distribution). Then, the algorithm has to work backwards to establish the missing parameters. For a survey on inverse problems in rendering, refer to [22,23].

Fortunately, it has been shown [18] that the general problem of finding the reflector shape, given the illumination distribution in a general scene, can be simplified. This simplification proceeds in a first step by reformulating the problem by the inverse propagation of requirements, from the specified surfaces to an enclosure surrounding the reflector. As a result, an outgoing spatial radiance distribution for the sources is obtained. In this way, the problem of finding the reflector shape from the light distribution in the surfaces of a scene can be simplified to finding the shape from the required outgoing light distribution from the optical set (reflector, bulb and diffusor).

In this paper we present a solution to the problem of finding the shape of a reflector given the outgoing radiance distribution that should emanate from the resulting optical set, without diffusor, as seen at the far field region, i.e.: at large distances from the optical set. Our solution is based on our previous work [24] and differs from other previous approaches in:

- the type of surface used to define the reflector shape (a regular grid of heights instead of a bicubic b-spline [25,26]) that gives more flexibility in the range of achievable surfaces (although introducing C^0 continuity on the edges joining triangles).
- the generality of the light propagation simulation step which, in our case, is based on the well-known Monte Carlo Light Tracing algorithm that can handle all sorts of BRDF.
- handling inter-reflections in an efficient and natural way. Traditional approaches work without taking inter-reflections or general BRDFs into account [25–27].
- the global strategy used for obtaining the desired reflector surface that takes into account the user's knowledge/experience.

2.2. Problem formulation

The usual approach starts by reformulating the problem in the following manner: In an iterative procedure, minimize the distance between the outgoing radiance distribution generated by the current reflector, $OutRad(\eta)$ (η is a n -dimensional vector of components $\eta_1 \dots \eta_n$ consisting of the parameters that control the shape of the reflector), and the desired outgoing radiance distribution, $OutRad_{desired}$, with respect to the η_i components of the η vector (i.e.: $\eta_i \in \mathbb{R}$, $i \in [0, n]$) that define the shape of the reflector.

This allows to define a function $f(\eta) : \mathbb{R}^n \rightarrow \mathbb{R}$ of the form

$$f(\eta) = dist(OutRad(\eta), OutRad_{desired}).$$

A distance can be naturally defined (see next subsection) in terms of a norm in the space of outgoing radiance

distributions, as $dist(X, Y) \equiv \|X - Y\|$, where $\|\cdot\|$ is some properly defined form. Our objective will be to minimize function f .

Inside the $OutRad(\eta)$ function, the shape of the optical set is defined by a set of parameters/control points, contained in η . Each time $OutRad(\eta)$ is evaluated, that is, each time an outgoing radiance representation needs to be generated for the shape (defined by the parameter vector η), two steps are taken. First, generate a surface with the shape specified by η . Then, invoke a routine called “lighting simulation”.

We choose to use the $C\gamma$ coordinate system [1] as our discrete representation for the outgoing radiance distributions, because it represents a standard in the lighting engineering industry. This coordinate system is depicted in Fig. 4. As we can see, the $C\gamma$ representation is an angular representation given by two angles, c and γ ($c \in C[0, 2\pi]$, $\gamma \in \Gamma \subset [0, \pi]$). As our discrete representation uses only a finite number of c and γ angles, we get a discrete set of $C\gamma$ entries, with an intensity value for each pair of c and γ angles chosen. Then we can establish a correspondence between the $C\gamma$ representation and the space of real matrices $s \times t$ where $s = |C|$ and $t = |\Gamma|$. We will use this correspondence to define a distance metric, see next section.

2.2.1. Choice of distance

The distance metric we have used and tested [24] for our outgoing radiance representation is the well-known l^2 norm. Since we are using the $C\gamma$ representation for outgoing radiances, we take the error to be given by

$$Error_2(\eta) = \left(\sum_{ij} \omega^{ij} |OutRad^{ij}(\eta) - OutRad_{desired}^{ij}|^2 \right)^{1/2},$$

i.e. a weighted sum of the squares of the differences between matrix entries, where the sum over indices ij must be understood over all the indices in the $C\gamma$ representation, and ω^{ij} is the solid angle subtended by the ij th entry of the $C\gamma$ matrix. Constructive restrictions were incorporated into the solution in the form of penalizing terms to this error. Thus, in practice, a new was defined accounting for those extra terms.

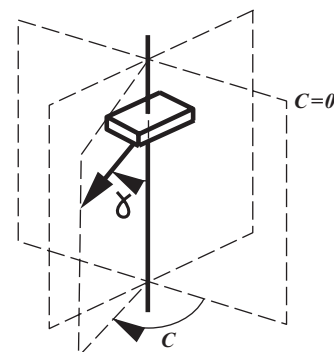


Fig. 4. The $C\gamma$ coordinate system.

2.2.2. Further definitions

It will be useful to introduce a new $C\gamma$ matrix Δ_i whose jk th element is defined by [24]

$$\Delta_i^{jk}(\eta) = |OutRad^{ij}(\eta) - OutRad^{jk}(\eta^i)|, \tag{2}$$

where η is a vector defining a reflector shape and η^i the modified vector whose components are $(\eta^i)_j = (\eta)_j + \delta_j^i \varepsilon$, δ_j^i the Kronecker delta, and ε a small distance. Δ_i represents the i th vertex differential influence on the output $C\gamma$, which shows us the regions on the $C\gamma$ matrix affected by the i th vertex for the current configuration.

Analogously, we can define the $DC\gamma$ distribution whose jk th element is defined by

$$D^{jk}(\eta) = |OutRad^{jk}(\eta) - OutRad^{jk}_{desired}|, \tag{3}$$

which shows the difference between the current $C\gamma$ distribution and the desired one.

We also define a local error measure for each vertex i , called $Status_i$, which is computed as the differential contribution of this vertex to the overall error. Actually, it is a computation of the superposition of the Δ_i and the $DC\gamma$ distributions weighted by the subtended angles. Its elements are defined by (see Fig. 5):

$$Status_i = \sum_{jk} \omega^{jk} \Delta_i^{jk} D^{jk}. \tag{4}$$

It is important to observe that $Status_i \geq 0$, as the Δ_i and D are matrices of positive elements (Fig. 6).

2.3. Overview of the previous solution

In a previous work [24], we have found a solution for the general case when no user-provided information was available. In this section we will overview such work, outlining the basis for the new solution presented here.

That approach starts from a generic initial surface, iteratively minimizing the distance between the outgoing

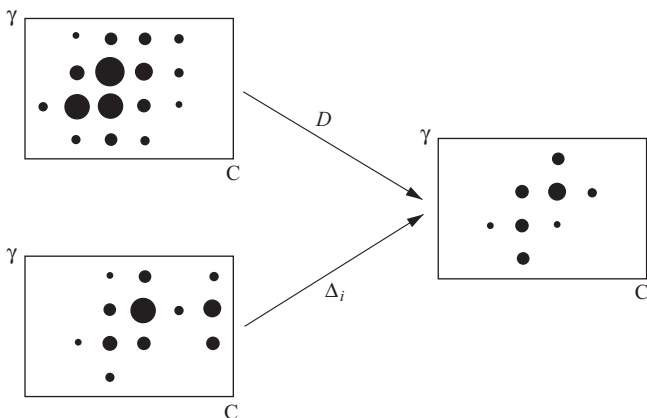


Fig. 5. Drawing of the superposition between D and Δ_i , as defined in Eq. (2) and following.

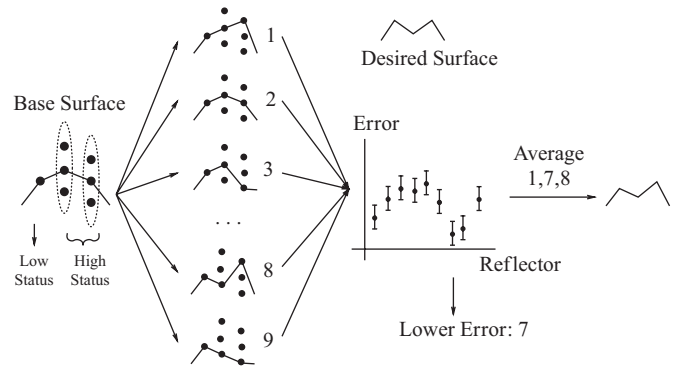


Fig. 6. Overview of the general optimization algorithm.

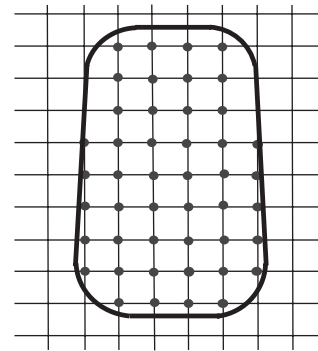


Fig. 7. Lattice border and optimizable points.

radiance distribution of the current reflector, $OutRad(\eta)$, and the desired (user specified) outgoing radiance distribution, $OutRad_{desired}$.

For the general case the algorithm works “around” the chosen starting reflector by generating a family of new ones by iteratively moving each original component of η . Those new reflectors are evaluated using a Monte Carlo light ray tracing algorithm, and the ones with errors close to the one with the best error so far are averaged to obtain the solution of the current iteration. Actually, “close” in this context refers to all reflectors whose evaluation gives a value with a variance that overlaps with the error and the variance of the best one. Once this average reflector is generated, and if the user-defined tolerance has not been achieved, the algorithm proceeds by refining the surface by adding new control parameters, and restarts the optimization loop mentioned above.

To perform our optimization, we build and evaluate each member of a family of reflectors obtained by iteratively displacing each one of the vertices. For this to be practical, the size of this generated family of reflectors has to be kept manageable. Obviously, there is a compromise between the desired accuracy given by these increments added to the vertices (trying to ensure that the family contains at least one reflector close to the desired target), and the size of such family.

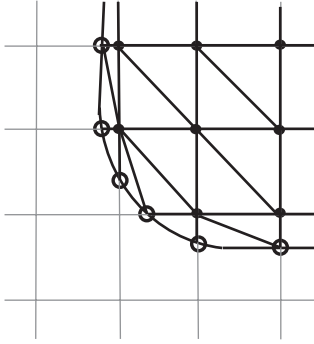


Fig. 8. Close up of tessellated lattice border.

2.3.1. Shape representation

We have chosen to use a grid representation scheme (see Fig. 7). Its features are:

- it allows to avoid undesirable striations since images (in the optical sense) are formed behind the reflector [1].
- an accurate template or tool to build a physical reflector from the resulting data can easily be made [1].
- the routines for boundary matching are almost non-existing: Only the grid points *inside* the boundary are suitable for optimization. When assembling the surface, the polygons that join the grid points that define the surface in the interior of the border with the border itself are created, making a polygonal approximation to the border given originally (see Fig. 8). In order to do this, the border is intersected with the grid lines and the intersection points (marked with circles in Fig. 8) are used to build the polygons. Those polygons are created following a pattern like the one shown in the figure.
- C^0 continuity at *all* the triangle lines in Fig. 8, that are all over the surface.
- to be flexible to adapt to the illumination requirements imposed by the $OutRad_{desired}$ radiance distribution, the surface built this way needs many vertices, which increases computing times.
- a fine grid is needed to achieve the desired smoothness to meet the manufacturing standards of the industry.

2.3.2. Wrapping the surface

A surface wrapper is a data structure that wraps around the basic polygonal surface to optimize, exposing only a few parameters to the optimization algorithm. The way each of that set of shown parameters affects the underlying surface depends on the exact wrapper definition. For example, a wrapper can expose a certain region of the underlying reflector for the optimizer to focus on this area, or it can show a selected set of vertices, interpolating the others in a way transparent to the rest of the system (see Fig. 9).

As mentioned before, a wrapping strategy was introduced in order to address both the exponential growth in the number of vertices at each iteration (see next subsections), and the consequent growth of the search space. Such strategy consists on iteratively exposing only a

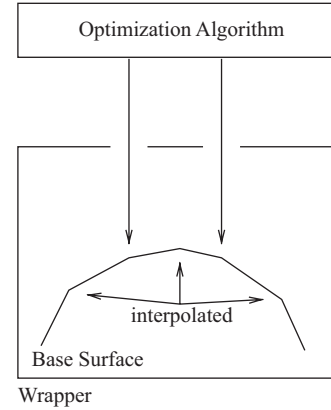


Fig. 9. A wrapper shows a selected set of vertices, interpolating the others.

few vertices until convergence is achieved or a new change in resolution is needed. This last event occurs when there are no vertices left to add.

We can analyze the two components of this wrapping strategy separately:

- *the surface wrapper*: It consists on a basic wrapper which is initialized with the vertices optimized at the previous *resolution* of the reflector, and interpolates the other vertices from the exposed ones. In our current implementation we are using the well-known Akima polynomial interpolator.
- *the addition of new vertices*: Each time the surface needs more flexibility (optimization has not converged with the current set of vertices), new vertices are exposed (added) by sorting the already used vertices according to their relative degree of wrongness $Status_i$ (Eq. (4)), and choosing the N with worst values. Then, for each one chosen, its surrounding vertices are selected and refined, adding them to be exposed to the optimization process.

Using this strategy has a small drawback that must be taken into account: it introduces a limit to the minimum achievable $Error_{used}$ since there is a certain difference between the unwrapped surface and the surface after wrapping is imposed. Of course, this difference diminishes towards zero as the algorithm adds new vertices at each iteration.

3. Overview of the proposed method

Then we start by reformulating the problem in the following manner: Starting from a *user-provided* initial surface, iteratively minimize the distance between the outgoing radiance distribution of the current reflector, $OutRad(\eta)$, and the desired (user specified) outgoing radiance distribution, $OutRad_{desired}$, where η is a vector of dimension n (i.e.: $\eta \in \mathbb{R}^n$) that defines the shape of the reflector. The surface bounds given by its confidence values (see Fig. 10(a)) must be taken into account at all times during this optimization process. This process can be

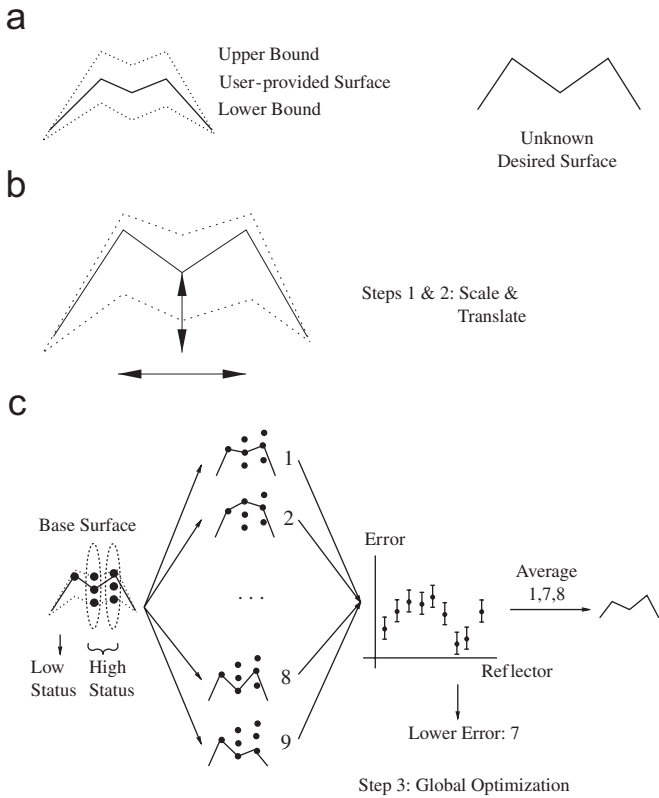


Fig. 10. The optimization algorithm: an overview.

thought of as being done in two basic steps:

- first, the user-provided shape is scaled and translated in order to try to get a better initial estimation of the desired shape to be found. This is done by linearly modifying the surface in each of its three dimensions, in two steps:
 - vertical accommodation, where the shape is modified in its height.
 - horizontal accommodation, where the surface is scaled and translated in its other two dimensions.
- see Fig. 10(b).
- then, a global optimization procedure as described above is applied, but restricted to respect the user-provided constraints to the desired shape (given in the form of initial confidence values). See Fig. 10(c).

4. User-guided optimization

In this section we will describe the way optimization presented in Section 2.3 can be improved by taking advantage of the user’s knowledge of the reflector to be built.

4.1. User input

As mentioned before, knowledge from the user is given by providing a reference, starting surface and a measure of

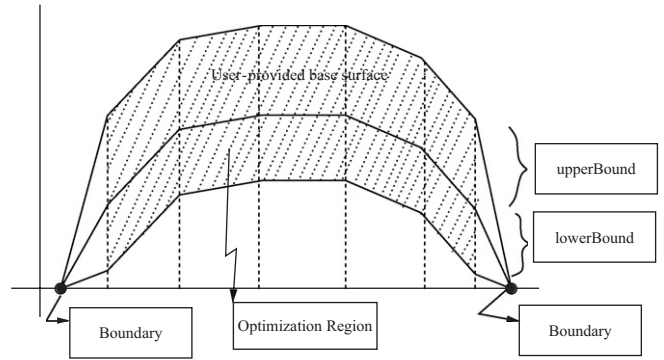


Fig. 11. The user-provided tolerance is used as a bound on the space of attainable shapes.

confidence. That initial shape is provided as a set of heights for the prescribed grid locations. On the other hand, the user’s tolerance is introduced by providing two real numbers, that represent the upper and lower bounds to the heights provided in the file defining the surface shape. This bound determine an upper/lower limit on the heights each vertex could take with respect to its initial position. We can say that these two real numbers represent offsets for the basic surface, generating two new surfaces, a lower and an upper one, which are the new bounding box for the optimization computations. See Fig. 11. Extensions where the user provides confidence levels at a vertex-wise level could easily be incorporated in the system described in this section.

4.2. User-bounded optimization

The optimization consists then of two nested loops, the outer one being responsible of the multi-resolution changes on the shape (see Section 2.3.2), and the inner one consisting of two basic steps that execute until a convergence criterion has been achieved. These two steps are:

- overall Accommodation (Section 4.2.1)
- vertex Optimization (Section 4.2.2)

Of course, both steps must preserve the user-provided bounds, and in Section 4.2.4 a detailed description of how this was ensured in the optimization process is presented.

The resulting algorithm looks like this:

```

Reflector := User defined reflector
while (not converged) and (not
tooManyIterations)
while (not converged) and (enough change in
error)
Overall Accommodation(Reflector)
error := Vertex Optimization(Reflector)
if not converged
increaseResolution(Reflector)
    
```

4.2.1. Overall accommodation

This step attempts to improve the original surface while trying to retain its original shape at the same time. This is done because we consider that, given his experience, the user provided a shape that is close to the desired one, so we try to take maximum advantage of this information. This is done by taking the original surface, defined as a function on the x, y plane:

$$z = f(x, y)$$

and changing it by introducing a linear modification in each possible dimension. This way, the surface now becomes a function of $\alpha, \beta, \gamma, \delta, \epsilon$ and ϕ of the form

$$z = \alpha f(\gamma x + \delta, \epsilon y + \phi) + \beta, \tag{5}$$

where α and β introduce linear variation in the z -axis, while the other four new variables serve as lateral/longitudinal accommodation linear correction terms/factors. See Fig. 12.

Unfortunately, this optimization on these 6-degrees of freedom would represent excessive calculation times, so we replaced it with a two-step optimization process:

- first, the optimization of α and β in the expression

$$z = \alpha f(x, y) + \beta, \tag{6}$$

leaving the variables x, y as constants with respect to the optimization process.

- then, the x, y directions are looked at, resulting in the optimization of γ, δ, ϵ and ϕ and in Eq. (5), with and acting now as constants with respect to the optimization process.

In the second step, special care must be taken when setting the bounds for the optimization of these four

variables, as no vertex must be re-located to a place outside the shape boundary.

This is done through a set of classes called Accommodation Wrappers. Each of those classes (see Section 2.3.2) has specific roles during this optimization stage. In particular, we used a “Grid Accommodation wrapper” during the optimization of the 2 + 4 variables mentioned above (α and β in the first round, and γ, δ, ϵ and ϕ for the second one). This wrapper only shows those degrees of freedom, hiding the individual vertices from the rest of the application. The results of this first stage are kept for the remaining of the main optimization loop into a “Fixed Grid Accommodation wrapper”, which modifies the underlying surface with those values, while exhibiting the individual vertex chosen for optimization to the rest of the optimization algorithm. In this way, the optimization proceeds as before, but the resulting surfaces benefit from the accommodation results.

The Accommodation Wrappers are built from a hierarchy of wrappers formed by three classes: a `GenericGridAccommodationWrapper`, and its two subclasses: the `GridAccommodationWrapper` and the `FixedGridAccommodationWrapper` (Fig. 13).

Those wrappers work by altering not the shape data stored, but the location of the vertices generated when the surface is built for testing just before the light simulation step. In this way, we are able to alter the x, y dimensions without really changing the internal data structures. Since this step only affects the way the outgoing light distribution is computed, this change has no side effects on the rest of the optimization process.

The main difference among the two type of Accommodation Wrappers is the degrees of freedom they show to the optimization process: The `GridAccommodationWrapper` works by letting the system optimize the above-mentioned $\alpha, \beta, \delta, \gamma, \epsilon$ and ϕ variables (Fig. 14). On the other hand, the `FixedGridAccommodationWrapper` works in a completely transparent way towards the rest of the optimization process: Once it is initialized with the above mentioned six values, it modifies the way reflectors are built at the light simulation step, passing through the usual degrees of freedom, i.e. the heights (z) for each location on the vertex grid. In this way, the rest of the optimization can proceed in a similar way as was described for the general case (see Section 2.3), while retaining the results of the previous step in the iterative process. See Fig. 15.

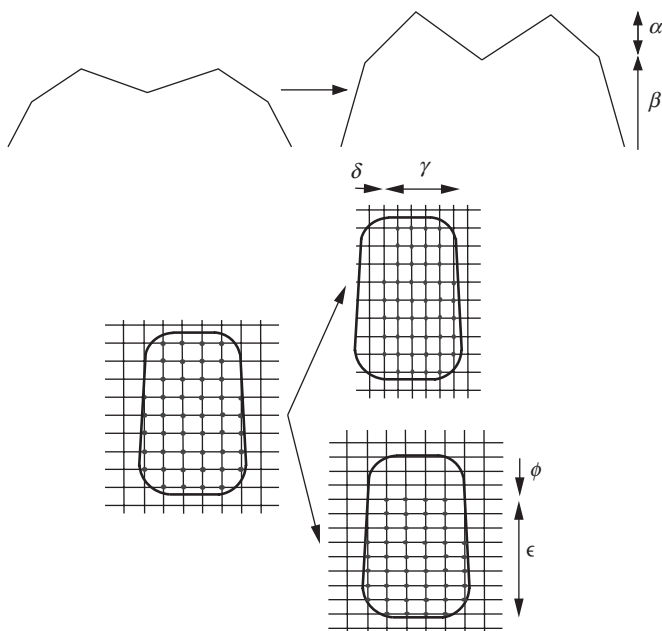


Fig. 12. The usage of the variables $\alpha, \beta, \gamma, \delta, \epsilon$ and ϕ .

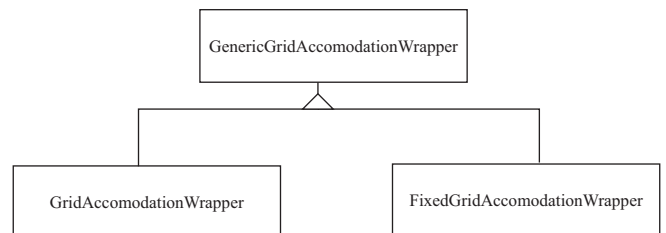


Fig. 13. Accommodation wrappers.

The optimization process itself is performed by the global optimization algorithm described before, with bounds set as will be described in Section 4.2.4.

The pseudocode for the inner loop presented above can actually be rewritten as follows:

```

Refl: = User defined reflector
WrappedRefl: = GridAccomodationWrapper (Refl)
XYZfactors: = optimizeXYZ (WrappedRefl)
WrappedRefl: = GridFixedAccomodationWrapper (Refl, XYZfactors)
While (not converged) and (not tooManyIterations)
    error: = Vertex Optimization (WrappedRefl)
    If not converged
        increaseResolution (Refl)

```

4.2.2. Vertex optimization

Once the overall shape was optimized as much as possible in the previous step, and if the error obtained is not lower than the accepted error, the optimization strategy proceeds to perform a minimization of the resulting error on a per-vertex basis. But, before doing that, the vertices that are most significant for the optimization should be identified, in order to be able to use a modification of the algorithm already described for the non-user guided optimization [24]. These algorithms are based on an optimization procedure based on the Akima interpolation algorithm. The main difference with the algorithms described in the previous section is that there we started with an extremely low-resolution surface, progressively refining it and adding new vertices to the optimization. Here we find ourselves in the situation where the user provides a high-resolution reflector to start with, so there is no previous surface to

4.2.3. Vertex selection

In order to find the set of vertices to be used to control the first iteration of the optimization procedure, we use a purely geometric criterion, trying to select those that ensure the highest control of the underlying surface. Then, in

successive iterations, vertices are added as described in Section 2.3.2.

The criterion to select the initial vertices is based on an heuristic that chooses the vertices that minimize the distance to all others, while avoiding the evaluation of pairs of neighbors in the list of vertices selected. In order to do that, the distance between to vertices i, j is defined as

$$dist(i, j) = \left((v_x^i - v_x^j)^2 + (v_y^i - v_y^j)^2 \right)^{1/2},$$

where v_x^i and v_y^i are the x, y coordinates of the vertices in the grid (see Section 2.3.1). For any given vertex i , we can define the value

$$weight(i) = \sum_j dist(i, j).$$

Then, the vertices are chosen by executing the following algorithm':

```

sort vertices i by increasing weight (i)
f: = 0
while (more vertices needed) and (not f = endOfVertices)
    while (not f = endOfVertices) and (hasChosenNeighbor (f))
        f++;
    if (not f = endOfVertices)
        put vertex in chosen list (f)
        updateVertexWeights (f)
        sort vertices
        f = 0
if (f = endOfVertices)
    haltWithError (NOT_ENOUGH_VERTICES_AVAILABLE)

```

use as a reference, and we do not know a priori which vertices should we start with for the initial optimization process.

Here, the outer loop tries to find as many vertices as requested, and the inner loop is the one that chooses the needed vertices from the available ones. In the code above,

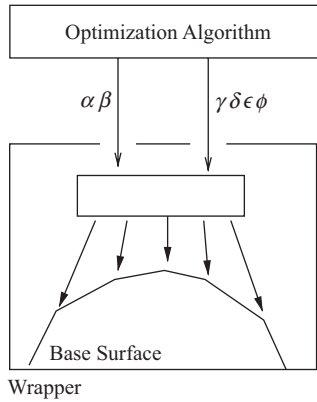


Fig. 14. Working of the grid accommodation wrapper.

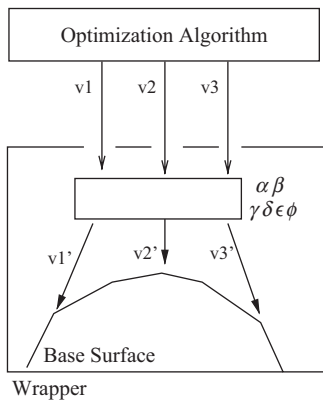


Fig. 15. Working of the fixed grid accommodation wrapper.

the procedure “updateVertexWeights(f)” subtracts to all vertex-associated weights a term proportional to the distance from each vertex to the new chosen one. We used the expression $-\omega \sum_j \text{dist}(\text{newChosenIndex}, j)$ as the subtracting term, where ω is a positive weighting factor, and *newChosenIndex* is the index of the vertex recently chosen to act as an optimization vertex. The condition hasChosenNeighbor(f) is used to ensure that no two neighboring vertices are chosen, since it was found to be a bad practice to use this type of vertices in order to control the surface shape in the interpolation process, as a concentration of optimizable vertices in a region reduces the flexibility of the achievable surface by the vertex interpolator used later.

4.2.4. Bounded optimizations

As mentioned above, the optimization must respect and preserve the user-defined bounds around the original shape. This means that the shape must verify

$$f^{user}(x, y) + \text{lowerBound} \leq f^{user}(x, y) + \text{upperBound}$$

for every point (x, y) inside the shape boundary. In general, *lowerBound* will be less than 0. Remember that in our current implementation, we are dealing basically with a discrete grid of heights, so this condition can be controlled only on those grid locations. Fortunately, since both the

user surface and the one the optimization algorithm works with are defined over the same grid locations, we will not introduce new problems: since intermediate points are linearly interpolated from the surface vertices, they cannot get higher than the user-defined surface (plus bounds) the corresponding vertices do not do so.

The bounds for the optimization process are controlled in a few steps:

- first, in the Overall Accommodation step (described in Section 4.2.1), α and β in Eq. (6) are controlled by finding the maximum (and minimum) possible values. Then, they are used as bounds for the global optimization procedure described before. This is done in a conservative way, by computing the minimum and maximum α and β values for each vertex and keeping the ones with smaller absolute value. Care must be taken as α is a multiplicative factor that should remain in the vicinity of 1. This way we find an upper and a lower bound a_l, a^u for α among all evaluated values. Similar steps should be taken with β , but this time performing just a minimization of absolute values, as β varies around 0.
- at the Vertex Optimization stage, described in Section 4.2.2, once the Grid-FixedAccommodationWrapper is used, the α and β values optimized in the previous step are set. Then, for the rest of the iteration, each vertex is given a bound that is the corresponding value given by the user input plus its bounds, taking into account the vertex current position and the already mentioned α and β factors. In this way, each vertex has a lower/upper bound given by the corresponding user-provided vertex plus its original lower/upper bound minus its current value at the present iteration, always controlled by the α and β previously computed.

4.2.5. Control of vertex displacements

The algorithm in [24] used a fixed spacing for the vertex movements, but it was found that using the relative error $Status_i$ (see Eq. (4)) not only to establish the ordering, but also to adapt the number of steps each vertex spans was extremely beneficial for the execution time of the optimization process. This improvement comes from the fact that the relative error described is a measure of the local error in shape each vertex contributes to the overall error, and it seems logical to use it as a guide of how much effort is needed for each vertex. The new algorithm for choosing the number of steps for the vertices starts by choosing an approximate budget of reflectors we are willing to evaluate (about 18000 in the tests performed). It is easy to realize that the product of the number of samples over the N vertices to optimize must equal the given *budget*.

$$\text{budget} \approx \prod_i n_i.$$

If we assume that each vertex i will span a number n_i of positions which is approximately proportional to its

relative error (called status in the following):

$$n_i \approx k * Status_i,$$

with k a proportionality constant, we have that the budget would be given by

$$budget \approx k * Status_1 * k * Status_2 * \dots * k * Status_N,$$

$$\ln(k) \approx \left(\ln(budget) - \sum_1^N \ln(Status_i) / N \right).$$

From there, it is easy to compute the number of steps for each vertex as

$$n_i = \text{round}(k * Status_i).$$

Thus, in practice, the real number of reflectors tested is “around” the predefined budget, instead of exactly the number chosen as budget. For example, in our experiments the original budget was set to 18 000 reflectors, and a number ranging from 13 000 to 21 000 reflectors were tested. Of course, if the *budget* is to be taken as an absolute limit for our number of tested reflectors, the function *truncate()* should be used instead of *round()* in the last expression.

4.2.6. Final vertex optimization algorithm

The algorithm presented in Section 2.3 makes too many unnecessary computations, so a more intelligent version was developed, which sorted the vertices according to their *Status_i* and put more effort where a larger error was found. This is possible as *Status_i* can be seen as the vertex differential influence on the difference between the current and desired output. It must be emphasized that this local measure of the error is only used to give a hint to the optimization algorithm on where to put more effort and by reducing the search space of possible reflectors to test, but the global optimization is still guided by a global criteria (the *Error_{used}* mentioned before). The vertices were grouped in sets and the subset of vertices with worst behavior was more finely sampled. This can be done because *Status_i* gives hints on the overall shape error in a vertex neighborhood, so we bias the importance of the optimization towards those regions that show the biggest deviations from the desired shape.

The final search space was built from the set of η vectors $\{\eta^{base} + \sum_{i=1}^N d^i\}$, with d^i the vector whose components are $(d^i)_j = \delta_j^i k^j StepSize$. The values for k^j are taken from $k^j \in [-k_{MAX}^j, k_{MAX}^j]$, and k_{MAX}^j is the number of steps for each degree of freedom resulting from the *Status_i* sorting ($k_{MAX}^i \geq k_{MAX}^j \equiv Status_i \geq Status_j$). In this equation, η^{base} is the resulting parameters from the previous iteration.

The brute force procedure described above is not enough when taking into account the noisy nature of the Monte Carlo algorithms used in the light simulation step, since a certain variance is associated with each evaluation of the *OutRad* function. Thus, performing the brute force search and retaining only the best value is not right, since each evaluation has an error associated that cannot be disregarded. This error, which we call the *AccuracyError*, is precomputed as the deviation from zero when computing the difference of the distribution resulting from a reference reflector and itself when computed with a different number of traced rays. So, a final version of the brute force algorithm was developed which deals with this situation: each time a value is computed, its error is taken into account by comparing its value minus its error with respect to the best so far plus its respective error (remember we are minimizing towards zero). That is, if $Error_{used}(\eta) - Current's AccuracyError \leq Error_{used}(best) + Best AccuracyError$, the value is kept in a list that represent all values optimally with respect to the best so far (with its respective error bar). The mentioned errors in the function evaluation are computed as the tabulated standard deviation for each accuracy, indexed by the number of rays fired in the light simulation. Finally, at the end of a brute force pass, the list retains a certain number of candidates whose error bars overlap with the best one. To choose one, the list is progressively filtered at higher accuracy by increasing the number of fired rays at the light propagation computations, until only one value remains in the list or until we have reached our maximum tolerable accuracy. In the latter case, we can consider the values as samples of the same ideal reflector, and average them to get the final value.

Here is the pseudocode for the algorithm:

```
{brute force pass}
ErrorBar := Error Bar at Current Accuracy
For each reflector R in the restricted SearchSpace
  evaluate R
  if (Error_used(R) - ErrorBar <=
    Error_used(BestAtList) + ErrorBarForBest)
    add (R, ErrorBar) to List
{Filtering Pass}
while not at maximum accuracy
  increase accuracy
  filter List at new accuracy
return (Average reflectors in List)
```

5. Results

In this section we will present the results obtained with the new algorithm described in this paper, and compare them to the results from our previous technique [24]. In Fig. 16, the different desired reflectors are shown, and in Fig. 17, a table showing the desired (1st row) and the user-provided reflectors used as test cases are drawn. Notice that those provided by the user show shape similarities with the desired results, as expected from a user with a good knowledge of how to fulfill the given requirements. For later analysis, the respective average path lengths of the traced rays for the reflectors in Fig. 16 are presented in Table 1.

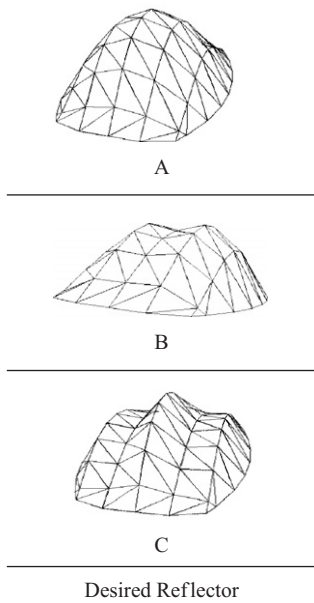


Fig. 16. Our set of desired objective reflectors.

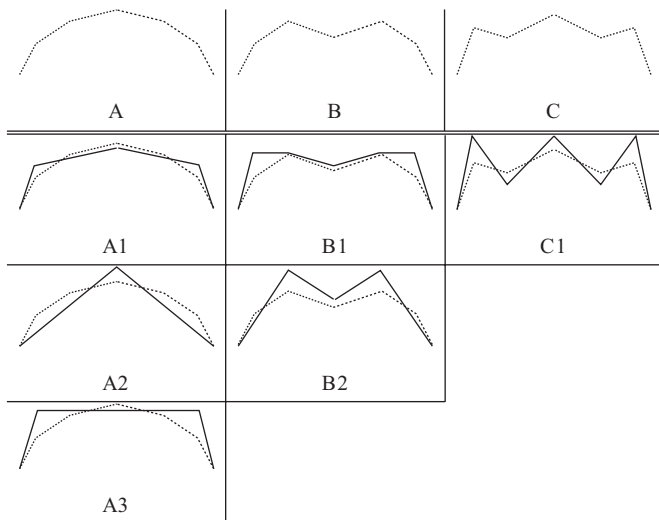


Fig. 17. First row shows the desired reflector, and the lower rows show user-provided starting reflectors.

Table 1

Average path lengths found for the reflectors used (see Fig. 16).

Refl BRDF	0	0.05	0.44	1
A	2.68	2.68	2.68	2.78
B	2.57	2.57	2.57	2.65
C	2.76	2.76	2.78	2.80

Table 2

Comparison of the results without (old algorithm) and with user help

BRDF	Test	Initial	Final	Time
0	Old	0.25±0.1	0.09±0.09	≈ 10 days
	A1	0.06±0.03	0.05±0.03	20 h 41' 46.5"
	A2	0.05±0.03	0.040±0.03	1 h 24' 39.4"
	A3	0.1±0.03	0.05±0.03	31 h 35' 45.4"
0.05	Old	0.3±0.1	0.2±0.1	10 days
	A1	0.17±0.04	0.16±0.04	26 h 37' 50.2"
	A2	0.22±0.04	0.17±0.04	40 h 08' 34.1"
	A3	0.23±0.04	0.16±0.04	22 h 17' 00.4"
0.44	Old	0.55±0.4	0.4±0.4	10 days
	A1	0.35±0.13	0.27±0.13	34 h 05' 36.0"
	A2	1.02±0.13	0.31±0.13	20 h 53' 22.9"
	A3	0.56±0.13	0.28±0.13	28 h 19' 54.7"
1	Old	0.8±0.6	0.6±0.6	10 days
	A1	0.5±0.2	0.4±0.2	24 h 29' 28.5"
	A2	1±0.2	0.5±0.2	25 h 38' 46.0"
	A3	0.9±0.2	0.4±0.2	20 h 44' 28.0"

The value in the first column, the diffuse to specular coefficient is the ratio between those coefficients in a regular Phong BRDF formula. The second column shows the results obtained. The experiments are labelled following the names given in Fig. 16.

Table 3

Comparison of the variances for the results of our algorithms, the old one presented in [23] and the new one presented here

BRDF	Variances	
	Old	New
0	0.1	0.03
0.05	0.1	0.04
0.44	0.4	0.13
1	0.6	0.2

In Table 2 a comparison of the results with and without user help is presented for case A for varying BRDFs, ranging from a pure specular to an almost diffuse surface (see first column in Table 2). The used BRDF is the Phong Model widely used in the computer graphics rendering community, and the variations were performed by changing the diffuse and specular weights in the model. The first thing to notice is the rise in the variance (see Table 3) in the function evaluations as the BRDF goes from more and more diffuse, something that was expected. The consequence is that the more diffuse the surface, the more

rays are needed to achieve a small increase in accuracy, which leads to forbiddingly high computing times for a very diffuse surface. Also, for very diffuse reflectors, it makes the results of our algorithm untrustworthy, since the variance approaches the magnitude of the function value.

It is important to observe, in Table 2, that the variance for the new algorithm is much lower than that for the previous one, because 10 times more rays were used. It was not possible to use the same number of rays as before, as the initial value for the user-provided reflector was small enough to be comparable to the associated variance for that number of rays. So, the algorithm uses a higher number of rays for the optimization in order to be reliable (it was found in practice that the error values used at any time should not be higher than about one third of the current error value). As we can see, the results on the new algorithm are quite promising, as computation times (last column) reduced from the order of 10 days to only one. So, what is most important about this table is the fact that the new algorithm started with an error which is, more or less, the same as the one for the previous version of the algorithm, giving much better results at the end due to the better, user-provided, starting point. This provides a measure of the quality of the results achieved by the optimization.

In order to be able to perform a visual assessment of the performance of our algorithm, in Fig. 18 we have a

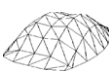
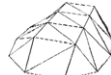
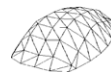
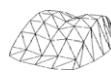
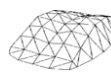
Reference Reflector	Starting/Obtained Reflector
	Previous Method [PPV04]  initial Reflector
	 Final Reflector
	User-guided optimization  user-provided starting reflector
	 obtained final Reflector

Fig. 18. First column: Reflector that generated the desired C_γ distribution, second column: starting reflector for previous version of the algorithm [24]. The example corresponds to the original Example B in Fig. 16 (with its initial and final reflectors), and the B1 reflector in Fig. 17 used as the starting point for the user-guided optimization

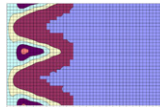
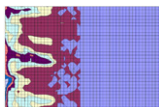
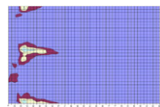
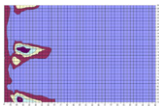
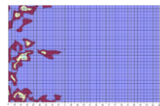
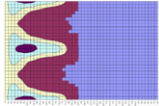
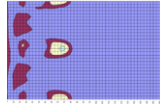
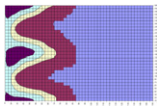
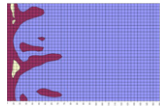
Desired Distribution	Starting/Obtained Distribution	Error Distribution
	Previous Method [PPV04]  initial reflector	
	 Distribution	 Error
	User-guided optimization  user-provided starting reflector	
	 Distribution	 Error

Fig. 19. First column: Desired C_γ distributions, second column: Distributions of starting reflectors for the previous method [24] and the new version of the algorithm, and third column, the error distribution computed with respect to the reference one. The example corresponds to the one in Fig. 18.

comparison of the results of the previous (right top) and new (right bottom) algorithms.

Fig. 19 further illustrates those results by plotting the desired light distribution (first column), the starting distribution for the previous algorithm (second column, first row), the finally obtained distribution for that algorithm (second column, second row), the distribution for the starting reflector for the new method (second column, third row) and the resulting distribution after the optimization of the reflector shape (second column, fourth row), as well as their respective error distributions with respect to the reference (third column). It is important to notice that, although the old algorithm gave a reasonably good improvement on the generated distributions, the new algorithm outperforms the previous one by generating a distribution with much smaller errors than the ones obtained before.

We could analyze the case where the previous algorithm is provided with a starting shape like the ones used to feed the new algorithm. As the previous algorithm starts from

Table 4
Results of the performed tests with different Phong BRDFs

BRDF	Test	Initial	Final	Time
0	A1	0.06±0.03	0.05±0.03	20 h 41' 46.5"
	A2	0.05±0.03	0.04±0.03	21 h 24' 39.4"
	A3	0.1±0.03	0.05±0.03	31 h 35' 45.4"
	B1	0.1±0.03	0.036±0.009	28 h 35' 45.4"
	B2	0.1±0.03	0.058±0.009	25 h 08' 42.0"
	C1	0.09±0.03	0.06±0.03	30 h 36' 47.3"
0.05	A1	0.17±0.04	0.16±0.04	26 h 37' 50.2"
	A2	0.22±0.04	0.17±0.04	40 h 08' 34.1"
	A3	0.23±0.04	0.16±0.04	22 h 17' 00.4"
	B1	0.20±0.04	0.12±0.04	39 h 07' 57.0"
	B2	0.22±0.04	0.16±0.04	30 h 31' 44.6"
	C1	0.21±0.04	0.16±0.04	30 h 43' 22.0"
0.44	A1	0.35±0.13	0.27±0.13	34 h 05' 36.0"
	A2	1.02±0.13	0.31±0.13	20 h 53' 22.9"
	A3	0.56±0.13	0.28±0.13	28 h 19' 54.7"
	B1	0.48±0.13	0.29±0.13	22 h 53' 08.0"
	B2	0.52±0.13	0.33±0.13	23 h 09' 57.0"
	C1	0.82±0.13	0.37±0.13	43 h 30' 55.1"
1	A1	0.5±0.2	0.4±0.2	24 h 29' 28.5"
	A2	1±0.2	0.5±0.2	25 h 38' 46.0"
	A3	0.9±0.2	0.4±0.2	20 h 44' 28.0"
	B1	0.7±0.2	0.4±0.2	28 h 58' 05.0"
	B2	0.8±0.2	0.5±0.2	22 h 55' 41.0"
	C1	2.3±0.2	0.6±0.2	52 h 40' 19.0"

the supposition that we are not necessarily close to the desired surface, it takes large steps in order to cover evenly the space of possible reflectors, thus ignoring the potential information stored in the user-provided surface. It would therefore test exactly the same number of possible solutions as before, disregarding the quality of the provided solution and thus making many unnecessary calculations in regions of the search space far from the region of interest. We can conclude that convergence will not be improved in the previous algorithm by choosing a smarter starting point, while the new algorithm takes full advantage of the new information.

And, finally, Table 4 shows the experiments performed to test our algorithm with different BRDFs and different starting and target points. The table shows that the new algorithm behaves quite satisfactorily under those examples as well, with running times of about 1 day, which is significantly less than the previous version, non-user guided algorithm, which took about 10 days for the optimization process.

It is important to mention that, in those user-guided experiments, the usage of a local illumination-based rendering method is unfeasible. This is because of the higher number of bounces needed to accurately compute the outgoing radiance distributions, which requires careful computations by using the more general Monte Carlo method. This is confirmed by the average path length presented in Table 5, which clearly shows that local illumination algorithms are inadequate for this sort of

Table 5
Average path lengths found for the optimization with user-defined starting reflectors (see Fig. 16) and Table 1

Refl BRDF	0	0.05	0.44	1
A1	2.71	2.73	2.75	2.78
A2	2.75	2.76	2.81	2.84
A3	2.81	2.80	2.98	3.01
B1	2.81	2.80	2.85	2.89
B2	2.88	2.86	2.87	2.89
C1	3.09	3.07	3.10	3.14

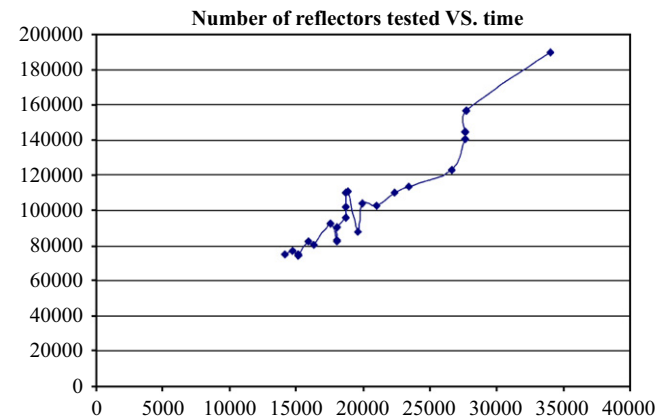


Fig. 20. General timing (in seconds) of the experiments with respect to the number of tested reflectors.

problems. It is important to note that the number of bounces has a small but clear increase with the diffusiveness of the surface.

We can perform a final analysis of the results by drawing the time used by the optimization algorithm vs. the number of reflectors computed in each case. In Fig. 20, the general linear behavior can be found, but a deeper analysis, grouping the different experiments by the BRDF or the shape of the reflectors will shed more light on the reasons for the different timings. If we group the experiments mainly by the range of shape they are spanning, we would observe that this is a very important factor to explain the small deviations from the linear behavior shown in Fig. 20. In Fig. 21 we can see the experiments grouped by the starting user-provided reflectors. From those figures, we can conclude that certain shapes are prone to generate larger average path lengths, like the one presented in case C of Fig. 16, as rays tend to need more bounces until they get out of the optical set. Also, BRDFs have an important role, as the more diffusive the BRDF, the more length has the average path length. This is confirmed by the average path lengths in Table 5.

Finally, we can consider the case where the user provides an incorrect shape or, at least, a correct shape but distorted with respect to the “ideal” one. As long as the “ideal” shape to be found is inside the boundaries defined by the user-provided shape plus its confidence values, convergence

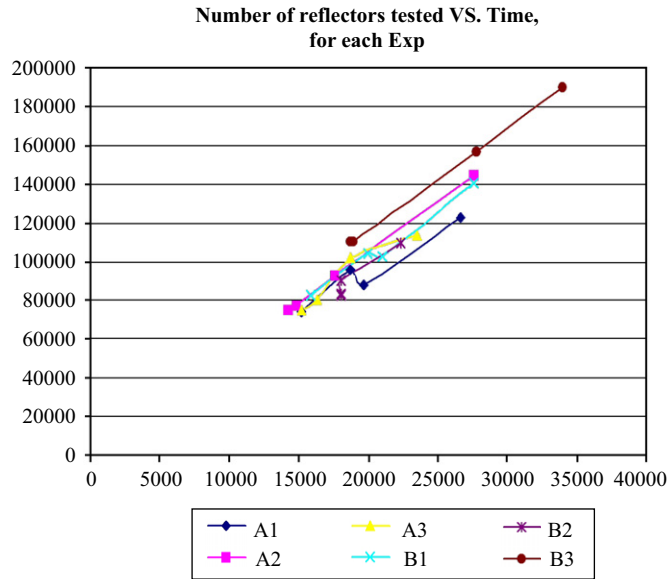


Fig. 21. General timing (in seconds) of the experiments with respect to the number of tested reflectors, grouped by the starting reflector given by the user.

would be a little bit slower, satisfactory. But, if the shape is partially or totally outside this optimization region, convergence would be impossible and the algorithm would be stuck in a local minimum. Finally, we can consider the case where the user provides an incorrect shape or, at least, a correct shape but distorted with respect to the “ideal” one. As long as the “ideal” shape to be found is inside the boundaries defined by the user-provided shape plus its confidence values, convergence would be a little bit slower, satisfactory. But, if the shape is partially or totally outside this optimization region, convergence would be impossible and the algorithm would be stuck in a local minimum.

6. Conclusions and future work

From our experiments we can conclude that the algorithm presented is stable and robust for pure specular and highly glossy reflector surfaces, while its convergence diminishes as the BRDF goes diffuse. This happens because diffuse BRDFs blur the information out, making it impossible for the optimization algorithms to distinguish the dominant features they are trying to determine. This phenomenon can easily be predicted by studying the behavior of the variances for a given BRDF with respect to the number of fired rays. The main drawback of our previous solution [24], its large time costs, has been overcome by using the user’s (a Lighting Engineer) knowledge in the form of a starting shape with confidence values. We have also found ways to understand and qualitatively predict the timing for a certain simulation given the starting reflector shape and the diffusiveness of the surface BRDF.

From the above, we can conclude that our objective of presenting a feasible algorithm able to treat the full inverse

reflector design problem (general BRDFs, inter-reflections, general light bulbs, etc), taking into account industry restrictions, has been achieved.

Accelerating the current algorithm is of extreme importance for the industrial usage of our algorithms. One of the key modules that directly impinge onto the performance of the system is the light simulation step. Thus, parallelization arises as a logical alternative to study. This parallelization can be introduced in two different ways:

- parallelization of the presented algorithms over several CPUs, which is sensible as the optimization step has a structure that makes it strongly parallelizable among several processors.
- another good possibility of doing this is by taking advantage of modern graphics hardware for the light simulations, as it is able to process more than 300 millions untextured polygons every second. As our current reflectors have about 200 polygons, and reflectors for more complex industrial treatment probably would need about 100 times more polygons, we can arrive at the conclusion that, with the new graphics hardware available at consumer level, we probably will be able to process more than 30 000 reflectors per second. Unfortunately, to get those numbers by using hardware, we probably would need to make some sacrifices, as the type of BRDF used or the calculation of inter-reflections. This is not necessarily a bad thing, as we can use the resulting reflector as a good starting point for a more accurate Monte Carlo simulation.

Although improving the speed of the calculation for the illumination step would drastically reduce calculation times, the core of the algorithm grows exponentially with the number of vertices computed (remember that a combination of all sensible positions for each vertex must be calculated). Thus, finding an algorithm that behaves polynomially with the number of degrees of freedom to optimize is a crucial point to take into account as future work. For example, we have recently been acquainted with [28,29] which present new results on the solution of a partial differential equation that arises under the mathematical formulation of the problem with some extra constraints (pure specular BRDF, no inter-reflections, no visibility calculations, the shape must be defined as a radial function on the sphere, linear constraints). Those studies show that this restricted problem can be reduced to a linear optimization problem and that algorithms in linear programming apply. Taking advantage of that solution as a starting point for our, more general, problem is an interesting option to explore.

Acknowledgements

This work was supported by grants TIN2004-07672-C03 and TIN2004-08065-C02 of MCyT (Spain), and SGR2005GRC of Generalitat de Catalunya—DGR.

References

- [1] Coaton JR, Marsden AM. Lamps and lighting. London: Arnold; 1997.
- [2] Elmer WB. The optical design of reflectors. New York/Chicester/Brisbane/Toronto: Wiley; 1978.
- [3] Keitz HAE. Cálculos y Medidas en Luminotecnia. Biblioteca Técnica Philips, Ed. Paraninfo, 1993 [In Spanish].
- [4] Deville PM. Modélisation et Simulation des Propriétés Radiatives des Sources Lumineuses. PhD thesis, Universte Henry Poincare, Nancy I, France, 1996 [In french].
- [5] Deville PM, Merzuk S, Cazier D, Paul J-C. Spectral data modeling for lighting applications. Computer Graphics Forum (Eurographics '94) 1994;13(3):97–106.
- [6] Deville PM, Paul J-C. Modeling the spatial energy distribution of complex light sources for lighting engineering. In: Hanrahan PM, Purgathofer W, editors. Rendering techniques '95 (Proceedings of the sixth eurographics workshop on rendering). New York: Springer; 1995. p. 147–59.
- [7] Poulin P, Fournier A. Lights from highlights and shadows. Computer Graphics 1992;25(2):31–8.
- [8] Poulin P, Ratib K, Jacques M. Sketching shadows and highlights to position lights. In: Proceedings of computer graphics international 97. Silver Spring, MD: IEEE Computer Society; 1997. p. 56–63.
- [9] Harutunian V, Morales JC, Howell JR. Radiation exchange within an enclosure of diffuse-gray surfaces: the inverse problem. In: Inverse problems in heat transfer, ASME/AIChE national heat transfer conference, Portland, August 1995.
- [10] Kawai JK, Painter JS, Cohen MF. Radiooptimization—goal based rendering. In: Computer graphics proceedings, annual conference series, 1993 (ACM SIGGRAPH '93 Proceedings), 1993. p. 147–54.
- [11] Marschner SR. Inverse rendering in computer graphics. PhD thesis, Program of Computer Graphics, Cornell University, Ithaca, NY, 1998.
- [12] Ramamoorthi R, Hanrahan P. A signal-processing framework for inverse rendering. In: Computer graphics proceedings, annual conference series (SIGGRAPH 2001), August 2001. p. 117–28.
- [13] Schoeneman C, Dorsey J, Smits B, Arvo J, Greenberg D. Painting with light. In: Computer graphics proceedings, annual conference series, 1993 (ACM SIGGRAPH '93 Proceedings), 1993. p. 143–6.
- [14] Poulin P, Fournier A. Painting surface characteristics. In: Hanrahan PM, Purgathofer W, editors. Rendering techniques '95 (Proceedings of the sixth eurographics workshop on rendering). New York: Springer; 1995. p. 119–29.
- [15] Boivin S, Gagalowicz A. Image-based rendering of diffuse, specular and glossy surfaces from a single image. In: Computer graphics proceedings, annual conference series (SIGGRAPH 2001), p. 107–16, August 2001.
- [16] Debevec P, Hawkins T, Tchou C, Duiker H-P, Sarokin W, Sagar M. Acquiring the reflectance field of a human face. In: Computer graphics proceedings, annual conference series (SIGGRAPH 2000), 2000. p. 145–56.
- [17] Ashdown I. Non-imaging optics design using genetic algorithms. Journal of the Illuminating Engineering Society 1994;23(1):12–21.
- [18] Cardoso Costa A, Augusto Sousa A, Nunes Ferreira F. Lighting design: a goal based approach using optimization. In: Lichinski D, Ward Larson G, editors. Rendering techniques '99 (Proceedings of the 10th eurographics workshop on rendering). New York: Springer; June 1999. p. 317–28.
- [19] Cardoso Costa A, Augusto Sousa A, Nunes Ferreira F. Optimisation and lighting design. In: WSCG '99 (Seventh international conference in central Europe on computer graphics, visualization and interactive digital media), Plzen-Borey, Czech Republic: University of West Bohemia, 1999. p. SP/29–36.
- [20] Doyle S, Corcoran D, Connell J. Automated mirror design for an extended light source. Proceedings of SPIE 1999;3781:94.
- [21] Doyle S, Corcoran D, Connell J. Automated mirror design using an evolution strategy. Optical Engineering 1999;38(2):323–33.
- [22] Patow G, Pueyo X. A survey on inverse rendering problems. Computer Graphics Forum 2003;22(4):663–87.
- [23] Patow G, Pueyo X. A survey of inverse surface design from light transport behavior specification. Computer Graphics Forum 2005;24(4):773–89.
- [24] Patow G, Pueyo X, Vinacua A. Reflector design from radiance distributions. International Journal of Shape Modelling 2004; 10(2):211–35.
- [25] Engl HW, Neubauer A. Reflector design as an inverse problem. In: Heiliö M, editor. Proceedings of the fifth European conference on mathematics in industry. Stuttgart: Teubner; 1991. p. 13–24.
- [26] Neubauer A. The iterative solution of a nonlinear inverse problem from industry: design of reflectors. In: Laurent PJ, Le Méhauté A, Schumaker LL, editors. Curves and surfaces in geometric design. Boston: A.K. Peters; 1994. p. 335–42.
- [27] Kochengin SA, Olikier VI. Determination of reflector surfaces from near-field scattering data II. Numerical solution. Numerical Mathematics 1998;79(4):553–68.
- [28] Glimm T, Olikier V. Optical design of one-reflector systems and the Monge–Kantorovich mass transfer problem. Journal of Mathematical Science (NY) 2003;117(3):4096–108.
- [29] Wang X-J. On the design of a reflector antenna II. Calc Var PDE 2004;20(3):329–41.