Eurographics Symposium on Parallel Graphics and Visualization (2006)
Alan Heirich, Bruno Raffin, and Luis Paulo dos Santos (Editors)

Parallelization of Inverse Design of Luminaire Reflectors

J.A. Magallon[1], G. Patow[2], F.J. Seron[1] and X. Pueyo[2]

[1] Grupo de Informatica Grafica Avanzada (GIGA)
Dept. de Informatica e Ingenieria de Sistemas, Universidad de Zaragoza, Spain
[2] Grup de Grafics de Girona (GGG)
Institut d'Informatica i Aplicacions, Universitat de Girona, Spain

Abstract

This paper presents the parallelization of techniques for the design of reflector shapes from prescribed optical properties (far-field radiance distribution), geometrical constraints and, if available, a user-given initial guess. This is a problem of high importance in the field of Lighting Engineering, more specifically for Luminaire Design. Light propagation inside and outside the optical set must be computed and the resulting radiance distribution compared to the desired one in an iterative process. Constraints on the shape imposed by industry needs must be taken into account, bounding the set of possible shape definitions. A general approach is based on a minimization procedure on the space of possible reflector shapes, starting from a generic or a user-provided shape.
This minimization techniques are usually known also as inverse problems*, and are very expensive in computational power, requiring a long time to reach a good solution. To reduce this high resource needs we propose a parallel approach, based on SMP and clustering, that can bring the simulation times to a more feasible level.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Physically based modeling, D.1.3 [Programming Techniques]: Parallel Programming, I.6.8 [Simulation and Modeling]: Parallel, Monte Carlo

1. Introduction

A reflector is just a part of what is called, in Lighting Engineering, an *optical set*, which consists of a light bulb, the reflector itself (whose shape has to be designed) and the diffusor (figure 1). The reflector has a border, contained in a plane, that limits its shape. In general, a reflector must fit inside a holding case, so its shape cannot be lower at any point than the plane defined by the border nor higher than a certain threshold defined by the case. General BRDFs for the reflector surface must be taken into account. This is a common configuration for illumination settings at streets and general open spaces, besides other cases.

In this paper we will focus on the parallelization of the following simplified problem: in view of the far-field outgoing radiance distribution of a light bulb and a reflector border, find the shape for the reflector whose resulting illumination matches a given optical set outgoing radiance distribution. Do this below an adequate user-defined threshold, and taking into account the suggested initial shape plus its confidence values. The algorithm moves towards minimizing the distance, in the l^2 metric, between the resulting illumination from the reflector and a prescribed, ideal optical radiance distribution specified by the user.

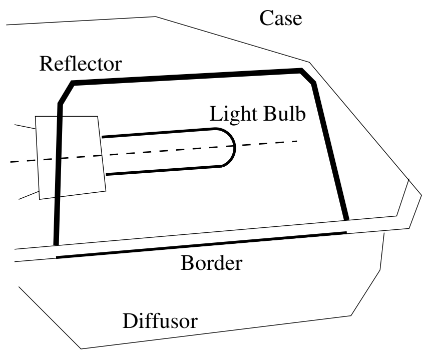

Figure 1: *An Optical Set.*

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

1. The shape must satisfy certain 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.
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 (it must fit inside the case).

Simulation problems are usually very hard with respect to numerical computations. And inverse problems are even more hungry of computing power, as numerical methods to solve an inverse problem involve the solution of many direct simulation problems. Even with the high performance of modern computing hardware, the traditional approach of sequential processing on a computer is not enough to bring inverse design to a daily usage. The only way to surpass this limitation is to spread the calculations over several processors running in parallel.

The paper is organized as follows:

- Section 2 presents a short state of the art for the solution of inverse problems in light simulation.
- Section 3 will present an overview of the inverse problem this paper focus on, the design of reflectors. This section will also describe the implementation of a sequential algorithm to solve the problem.
- Sections 4 and 5 will detail respectively the design and implementation of the parallel solution that is proposed.
- And finally section 6 states the conclusions that have been derived from this work and proposes future lines for further investigation and development.

2. Previous Work on Inverse Problem Solution

The central problem of the paper can be put in the context of inverse illumination problems. These include topics such as:

- Source position/orientation [PRJ97].
- Luminaire emittance [SDS*93] [KPC93] [HMH95] [Mar98] [RH01].
- Surface characteristics of some relevant surface elements [DHT*00] [BG01].
- Shape or position and orientation of the reflectors in the scene [CSF99] [DCC99].

One 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 recent surveys on inverse problems in rendering, refer to [PP03] and for a survey on the sort of problems studied here, refer to [PP05].

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.: large distances from the optical set. Its main distinctive features are:

- the type of surface used to define the reflector shape (a regular grid of heights instead of a bicubic b-spline [Neu94]) 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 Bidirectional Reflection Distribution Functions
- handling interreflections in an efficient and natural way. Traditional approaches work without taking interreflections (local illumination model) or general BRDFs into account [KO98] [Neu94].
- the global strategy used for obtaining the desired reflector surface, taking into account user's knowledge/experience.

To the best of our knowledge, no attempts were made to build a parallel version of any inverse problem of any type.

3. The inverse problem: Overview

The basic solution approach was introduced in [PPV04a] and starts by reformulating the problem in the following manner: Starting from an initial surface (which might be user-provided), iteratively minimize the distance between the outgoing radiance distribution of the current reflector, $L_{out}(\eta)$, and the desired (user specified) outgoing radiance distribution, $\widehat{L}_{out}$, where η is a vector of dimension n and defines the shape of the reflector.

We aim at the optimization of a function $f(\eta) :: \mathbb{R}^n \to \mathbb{R}$ of the form $f(\eta) = dist(L_{out}(\eta), \widehat{L}_{out})$. For the general case, the algorithm basically 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 by the usage of a Monte Carlo light ray tracing algorithm (in order to be able to accomodate general non–specular BRDFs), 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 superposes with the error and the variance of the best one. Once this new 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.

In the case when the user provides an initial shape, the process is done in two basic steps (figure 2). First, the user-provided shape is scaled and translated in order 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

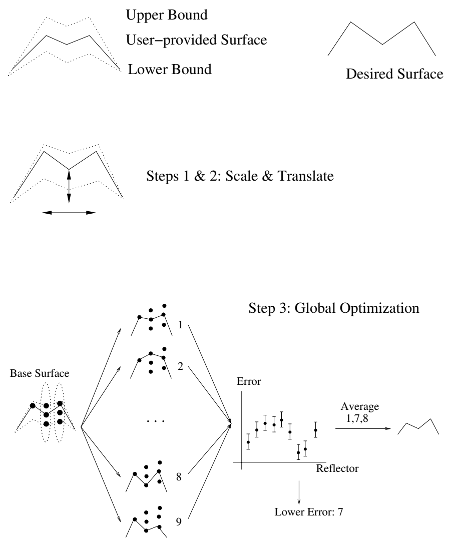

Figure 2: *The Optimization Algorithm: an overview.*

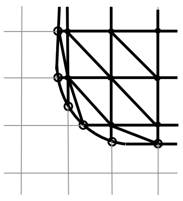

Figure 3: *Close up of tessellated lattice border.*

shape is modified in its height; and *horizontal accommodation*, where the surface is scaled and translated in its other two dimensions. Then, in the second step, a global optimization procedure as described above is applied, but restricted to respect the user's provided constraints to the desired shape.

3.1. Choice of Distance

The distance metric we have defined and tested for our outgoing radiance representation is the well known l^2 norm. Since we are using the industrial standard (C,γ) representation ([CM97]) for outgoing radiance, we take the error to be given by

$$Error_2(\eta) = \left(\sum_{ij} \omega^{ij} |L_{out}(\eta)^{ij} - \widehat{L}_{out}{}^{ij}|^2 \right)^{1/2}$$

i.e. summing the squared modulus of the difference between matrix entries, where the sum over indices ij must be understood over the two angular indices in the (C,γ) representation, and $i \in [0,s]$ and $j \in [0,t]$ and ω^{ij} is the solid angle subtended by the ij-th entry of the (C,γ) matrix.

Finally, we added penalizing terms to this error to take into account the industry restrictions. The penalty imposed to the objective function is the square of the constraint violation. In this way inequality constraints are handled through a penalty function that "turns on" when the constraint is not satisfied. For example, a penalty term $f_{C_i} = A_i(UpperBound - GridVertex_i)^2$ is added when $GridVertex_i$ is greater than $UpperBound$ and zero otherwise, with A_i being a convenient weighting factor defined as a system constant with a very large number.

3.2. Shape representation

We chose implement a grid representation scheme because:

- It allows us to avoid undesirable striations since images (in the optical sense) are formed behind the reflector [CM97].
- An accurate template or tool to build a physical reflector from the resulting data can easily be made.
- As only points "inside" the boundary are suitable for optimization, this bounds are intersected with the grid lines (figure 3).

3.3. Wrapping the surface

Basically, 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. For example, a wrapper can expose a certain region of the underlying reflector for the optimizer to focus on this area, interpolating the rest in a way transparent to the system. This strategy is applied by iteratively adding only a few vertices to the wrapper until convergence is achieved or a new change in resolution is needed, which occurs when there are no vertices left to add.

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

- The surface wrapper: this component consists of a basic wrapper which is initialized with the vertices optimized at the previous resolution of the reflector, and which interpolates the other vertices. 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, new vertices are added by sorting the already used vertices according to their differential contribution to the overall error (the regions on the $C\gamma$ matrix affected by the vertex for the current configuration [PPV04a]), and choosing the worst N vertices. For each of these, its surrounding vertices are selected from the list of free vertices, and the m with maximum reflector free area coverage are added to the wrapper ([PPV04b]).

We also use this differential vertex contribution (which we will call *delta* from now on) as a measure to adapt the number of steps each vertex spans, making worst vertices span a bigger number of positions.

```
Refl := create a low-res reflector
while       (not converged)
      and  (not userDefinedStop)
    FreeVerts := all verts in Refl
    WrapRefl := wrap(Refl, FreeVerts)
    while      (not converged)
          and (FreeVerts is not empty)
        addVerts(WrapRefl, FreeVerts)
        optimize(WrapRefl)
    if (not converged)
        increaseResolution(Refl)
```

Figure 4: *General optimization algorithm.*

3.4. General Optimization

As stated above, we reformulate the problem as a global optimization one. The algorithm is simply described in figure 4 (see [PPV04a] for additional details).

We start with a low dimensional reflector (generally, a 2 cols × 4 rows reflector), and try to optimize its shape. In order to both alleviate the exponential nature of the change in reflector resolution (number of vertices to optimize), and also to allow a more progressive reduction of the error distance, we introduce a wrapping scheme that allows a progressive introduction of vertices to optimize, see Section 3.3.

At the core of the algorithm, we conducted tests of the performance of each member of a family of reflectors obtained by iteratively adding an increment to each combination of the vertices. For this to be practical, the size of this generated family of reflectors has to be kept manageable. Thus, the vertices are sorted according to their relative degree of wrongness (see below) and we put more effort where a larger error was found.

3.5. User Guided Optimization

In this section we will describe the way optimization can be improved by taking advantage of the user's knowledge of the reflector to be built. This is done in a two-fold manner: on one side, use a user-provided shape as starting point for the optimization procedure, and, on the other hand, use the confidence bounds the user gave with the shape as bounds for the optimization process. Those bounds are intended to be used to focus the optimization on the family of possible reflectors to the ones around the user-provided starting shape (see figure 5 for some examples).

The procedure consists of two nested loops, the outer one responsible of the multi-resolution changes on the shape (See Section 3.3), and the other one consisting of two basic steps that execute until a convergence criterion has been achieved. These two steps are: Overall Accommodation (Section 3.5.1) and Vertex Optimization (Section 3.5.2). Of course, both steps must preserve the user provided bounds.

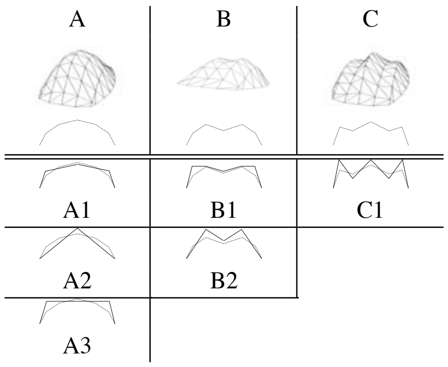

Figure 5: *Sample set of desired objective reflectors, with their respective profiles, and schemas of initial reflectors for user-guided optimization tests.*

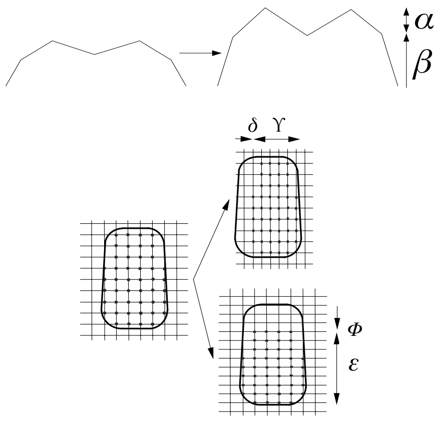

Figure 6: *The usage of the variables* $\alpha, \beta, \gamma, \delta, \varepsilon$ *and* ϕ.

3.5.1. Overall Accommodation

This step consists of trying to modify the original surface while trying to retain its original shape at the same time. This is done because we consider that 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 $z = f(x,y)$ on the (x,y) plane, and changing it by introducing a linear modification in each possible dimension. This way, the surface now becomes a function of $\alpha, \beta, \gamma, \delta, \varepsilon$ and ϕ of the form

$$z = \alpha f(\gamma x + \delta, \varepsilon y + \phi) + \beta$$

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 figure 6).

The optimization process itself is performed by the global optimization algorithm described before, with bounds set in such a way that they preserve the reflector boundary.

3.5.2. Vertex Optimization

Once the overall shape was optimized as much as possible in the previous step, and if the obtained error is not lower than the desired error, the optimization strategy proceeds to perform a minimization of the resulting error on a per-vertex basis. 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 algorithms already described for the non-user guided optimization (see Section 3.4). Here we are in the situation where there is no previous surface to use as a reference, and we don't know a priori which vertices should we start with.

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. This criterion is based on choosing the vertices i that minimize the geometric distance $dist(i,j)$ to all others, while maximizing the distance to those already chosen by the algorithm.

The optimization must respect and preserve the user defined bounds (B_{hi} and B_{lo}) around the original shape. This means that the shape must verify

$$f^{user}(x,y) + B_{lo} \le f(x,y) \le f^{user}(x,y) + B_{hi}$$

for every point (x,y) inside the shape boundary. In most cases, B_{lo} is less than 0.

4. Parallel Algorithm

Designing a parallel algorithm to solve a simulation problem can be a complex task, where subjects as sequential code dependencies, data dependencies, balance between processing and communication and others have to be accounted for.

In this section we present the need for parallelization on inverse design of reflectors, and the algorithm and implementation that have been developed.

4.1. Need for Parallelization

As presented in previous sections, the inverse problem is solved by performing a high number of direct simulations. Each one is a light tracing simulation that can take from seconds to minutes to compute.

For a standard inverse problem, we may need to solve about 20000 direct simulations to reach a good design. So the average run time for an inverse design can go up to tens of hours – or even days – on a single computer. The need for a parallel solution is immediate.

4.2. Task and Data Dependencies of the Sequential Algorithm

The core of the optimization algorithm described in figure 4 is the `optimize()` function. The skeleton of this function is detailed in figure 7.

```
optimize(base_reflector):
  reflector_set: set of reflectors;
  store:         set of pair(reflector,value);

  generate deltas
  generate reflector_set from base_reflector
  for each r in reflector_set
    v := simulate(r)
    add pair(r,v) to store

  average the best values from store that overlap
          with the current global best one
```

Figure 7: *Schematic of the sequential* `optimize()` *algorithm.*

The analysis of the algorithm detected two main areas where parallel processing could be useful:

- On each step of the optimization (the pass from one known good solution to another better one), several direct simulations are performed. This simulations are completely unrelated to each other, as work on top of different geometries defined by different parameter sets, so they are candidates to be run in parallel.
- Each simulation is solved by light tracing algorithm. This algorithm shoots rays from the light that store illumination information on hit points over the geometry. Tracing each of these rays is also independent of the others, so this is another area suitable for parallel execution.

There are also places where concurrent processing can not be applied due to data dependencies:

- Each step of the simulation starts from the result of the previous one. This makes impossible to parallelize the calculation of two steps, and the global advance of the iterative algorithm is mainly sequential.

And finally, the analysis of relations between the possible parallel tasks detected and the data they use are also important regarding the implementation:

- Parallel direct solutions for each step use the same base reflector, modified by the current parameter set.
- Parallel tracing of rays on a direct simulation interact with exactly the same reflector.

These relations force some data of the algorithm to be shared between tasks, so a decision is needed about to store them on a unique instance (and let the tasks access it remotely) or to replicate them.

4.3. Global Communications and Control

Looking at the data dependencies discovered in the previous section, the communication needs of the parallel algorithm will depend on how much of this data is going to be shared between parallel tasks. The points where data needs to be spread or collected between tasks will mark points of synchronization in the parallel algorithm.

Data to be shared between tasks are:

- The geometry of a reflector in the direct simulation.
- The best solution reached at a point in time, so new solutions generated in parallel can be compared with it.

With this restrictions, and based on the available hardware described below, we designed our algorithm as follows:

- Each direct simulation will be parallelized on a SMP (shared memory multi–processing) approach. All tasks tracing rays have access to a shared geometry database, and store light in mutual exclusion in an also shared table. This makes the need for communications nearly null in this step.
- The simulations needed to get the next best solution may be performed in parallel on a distributed memory parallel computer. The communication needs here vary depending on the chosen distributed scheme.

For the distributed processing, two options appeared at a first glance:

- A master-slave (or farm) scheme, where one process guides the simulation, and several other tasks are just workers that perform direct simulations on behalf of the master.
- A data-parallel approach, where all tasks are equivalent and perform simulations of different data sets, obtaining a set of partial results. Their results are finally merged to get the best global solution.

Initially the first idea looked as the most promising, with respect to optimization. Regarding implementation, one requirement was the use of existing sequential optmization code that had been alredy tested and validated. As we started to build the master–slave approach we realized that, apart from simple data (like global minimum after each step or global state to begin next step), some very complex data structures had to be transferred between tasks (reflector geometry, wrapper, etc.), which required extensive modifications to current code. This was considered harmful with respect to the optimization algorithm reliability so at the end, the implementation chosen was a data-parallel approximation, which required just some syncronisation and data interchange, as can be seen in figure 8.

In our case, the solution is just a *floating point number*, that measures the error with respect to the desired final light field. So the communication needs are really low. The reflector and value sets used in the algorithm are not fully built, but created on demand depending on the number of computers in the parallel machine. So, if we run the simulation on a N box cluster, reflectors and values are generated and stored in chunks of N elements. This makes the system use a static (but interleaved) load balance.

```
optimize(base_reflector):
  self:          task-id;
  reflector_set: set of reflectors;
  value_set:     set of values;
  store:         set of pair(reflector,value);

  generate deltas
  generate reflector_set from base_reflector

  self := who-am-i
  for each r in reflector_set
    if (r belongs to self) // mine
      v := simulate(r)
    else
      v := <empty>
    store v in value_set

  for each v in value_set
    if (v is not <empty>) // I have the solution
      send(v,every-other-task)
    else
      receive(v) // someone will send it to me
   for each r in reflector_set,
            v in value_set
     add pair(r,v) to store
  // here all tasks have all values

  average the best values from store that overlap
          with the current global best one
```

Figure 8: *Schematic of the parallel* `optimize()` *algorithm.*

4.4. Scalability

The static load balance can look not so good in principle, but because of the relative sizes of the reflector set to simulate and the number of processors in the parallel computer, the misbalancing is minimal. So this approach should work fine for the ranges involved in our problem.

There are two parts in the algorithm that are run in parallel, and present different scalability.

For the calculation of deltas, the number of problems to solve is in the order of the parameter space dimension (usually below 10), but the number of rays used is much higher to obtain an accurate value for the deltas. So in this case only a small number of processors can be used to build one delta on each one. Anyway, the time for the delta calculation is a small fraction of the total run time.

The simulation of the reflectors in the set is much a scalable problem. Only if the number of processors grows to a level similar to the number of reflectors some scalability problems will arise. As stated previously, the number of processors is in the range $1-100$ and the number of reflectors to be simulated can be up to 15000.

5. Implementation and Results

In this section we present some relevant details of the implementation and the results we obtained.

5.1. Test environment: hardware and software

The parallel computer used to perform our tests was the *hermes* cluster of the Instituto de Investigacion en Ingenieria de Aragon ([I3A]).

It is a MIMD cluster built with 50 nodes. Each node has

- 2 Pentium4 Xeon processors at 2.8 GHz
- 2 Gb of RAM
- An internal 40Gb hard drive

The cluster is accessed via a front–end node, similar to the computation nodes but with 8Gb of RAM. The system has a central SAN/NAS storage system, built with two SunFire v240 servers and some StoreEdge boxes filled with SATA disks, offering a total space of 6 Tb. The interconnection of the nodes and the storage system is done via two parallel GigaBit Ethernet networks and several switches.

All the nodes run Linux, actually the FedoraCore 2 distribution. The system runs some batching and parallelization software, like GRID and Condor, but for our needs we focused on the also available LAM–7.1.1 implementation of MPI.

The LAM–MPI [Ind] software offers a complete implementation of the MPI v2 standard [MPI], with the corresponding libraries for development of MPI applications and the utilities for management of the parallel execution environment.

5.2. Mapping Tasks and Processors

As the nodes were also capable of SMP parallel processing, we initially tested an SMP version of the code, where each light tracing simulation was programmed on top of the POSIX Thread library. So our full algorithm has two levels of parallel execution: one at the simulation level, with shared memory multiprocessing, and one between simulations, in a distributed memory schema.

But due to restrictions in the use of the cluster (the batch queues should also be able to execute their jobs), this code was not used in the MPI version. Anyway, the performance of the threaded version of the simulation was nearly perfect, with relative efficiencies higher than 90%.

LAM-MPI allows to build a virtual parallel machine selecting which nodes the user wants to include in it. For our probes we used sets from 10 to 50 boxes, and the final test was executed on 40 of the 50 available nodes.

5.3. Performance

To evaluate the performance of the algorithm, we ran several sequential and parallel simulations, to average timings and diminish the effects of other processes running in the cluster.

To compare both simulations, we used the total running time, but as is explained in previous sections, scalability is different in two sections of the program. The delta calculation can not go faster than a certain speedup, because it can use only a limited number of nodes.

In table 1 we present the results for the two simulations of one step of the algorithms (one call to `optimize()`), with 100.000 rays per simulation, and a parameter space dimension of 7 (so there are seven deltas to calculate). The number of reflectors was 13824, and the number of processors included in the parallel machine was 40. In the table we show the run time (sequential and parallel), the spedup and the relative efficiency for the two main steps of the algorith (the delta calculation and the reflector simulation). Measures for each step are evaluated with respect to its own use of the processors (the delta part only runs on 7 nodes and the rest on 40) which is also shown in the table. The last row shows global data, looking at the program as a whole and neglecting the fact that the two parts are very different.

	Sequential	Parallel	N	S	E_r
Deltas	00:31:19	00:04:48	7	6.5	0.93
Reflectors	15:32:30	00:40:50	40	22.8	0.57
Total	16:08:28	00:52:35	40	18.4	0.46

Table 1: *Running time for sequential and parallel algorithms. Also, the number of processors used N, the speedup S and the relative efficiency E_r are shown.*

The delta part was calculated with a $\times 100$ factor in the number of rays used for the simulation, so each of them is noticeably longer than the ones in the reflector simulation part. Here communication times are not very important and the efficiency obtained is very good, about the 93%. The reflectors part uses lower precission simulations, so the times are very short (as we will show below, the system simulates about 400 reflectors per minute) and the times for computation and communication become comparable. Hereof the lower effciencies obtained, about 57%. But the speedup obtained is still good, higher than 20.

Our parallel algorithm is not focused on low-latency (i.e. calculating a direct simulation in the least possible time), but on throughput: getting a huge number of simulations in the lowest total time. To measure the throughput of the parallel machine, we also stored the total running time to simulate the reflectors at several points in time (for example, after every 100 simulations) so we could define some kind of 'reflectors per minute' throughput metric. The values of this metric are shown in table 2 (average) and figure 9 (evolution over time). The system runs at an average of 338 reflectors per minute, and the speedup matches the one obtained for run times.

	Sequential	Parallel	Speedup
RPM	14.8	338.5	22.8

Table 2: *Reflectors per minute, average.*

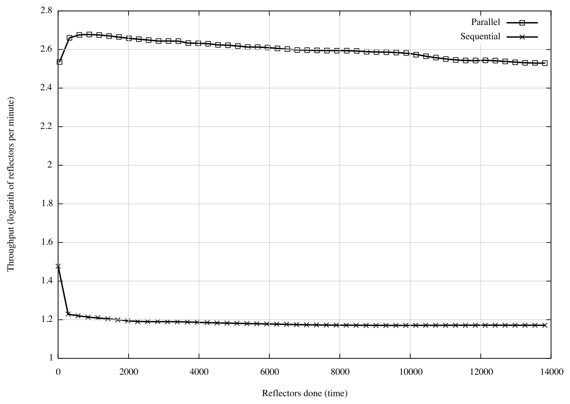

Figure 9: *Evolution over time of the average number of reflectors simulated per minute. Horizontal axis shows time with an arbitrary scale (the number of reflectors simulated in total). Vertical axis shows the rate of reflectors simulated per minute (in logaritmic scale).*

As can be seen in the graphs, throughput stabilizes over time, as the average over a long running time diminishes the runtime variance due to other CPU uses. The speedup of the algorithm is not perfect (only an efficiency about 50%); this can be caused by two main reasons:

- Other loads in the cluster. While running our simulations, there were other batch processes running in the cluster (a permanent load about 40%-50% was observed).
- Relative time of communications compared with simulations.

6. Conclusions and Future Work

In this work we have presented both:

- A new algorithm for the design of reflectors in luminaire optical sets.
- A parallel design and implementation of this algorithm on a MIMD clustered computer.

The behaviour of this implementation has been evaluated and compared with the sequential one, showing a big improvement in simulation times. Obtained speedups are higher than 20 and efficiency can be even more than 90% in some parts of the code.

There are some areas where analysis is still pending and further work will be needed:

- Estimation of the balance between communication and calculation times, and study of possible enhancements.
- Execution of the tests in a controlled environment, even with less processing nodes, to have more precise and undistorted timing information.

7. Acknowledgements

We would like to faithfully thank the Instituto de Investigacion en Ingenieria de Aragon ([I3A]) for granting us access to *hermes*, its computation cluster, and for allowing to 'overburn' the cluster for a short time.

This work has been developed under the project TIN2004-07672-C03 of the Ministerio de Ciencia y Tecnologia of Spain.

References

[BG01] 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)* (August 2001), pp. 107–116.

[CM97] COATON J. R., MARSDEN A.: *Lamps and Lighting*. Ed. Arnold, London, 1997.

[CSF99] COSTA A. C., SOUSA A. A., FERREIRA F. N.: Lighting design: A goal based approach using optimization. In *Rendering Techniques '99 (Proceedings of the 10th Eurographics Workshop on Rendering)* (New York, NY, June 1999), Lichinski D., Larson G. W., (Eds.), Springer-Verlag, pp. 317–328.

[DCC99] DOYLE S., CORCORAN D., CONNELL J.: Automated mirror design using an evolution strategy. *Optical Engineering 38*, 2 (1999), 323–333.

[DHT*00] 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), pp. 145–156.

[HMH95] HARUTUNIAN V., MORALES J. C., HOWELL J. R.: 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).

[I3A] I3A: INSTITUTO DE INVESTIGACIÓN EN INGENIERÍA DE ARAGÓN: *I3A Home Page*. http://i3a.unizar.es.

[Ind] INDIANA UNIVERSITY: *LAM-MPI Home Page*. http://www.lam-mpi.org/.

[KO98] KOCHENGIN S. A., OLIKER V. I.: Determination of reflector surfaces from near-field scattering data ii. numerical solution. *Numer. Math. 79*, 4 (1998), 553–568.

[KPC93] KAWAI J. K., PAINTER J. S., COHEN M. F.: Radiooptimization - Goal Based Rendering. In *Computer Graphics Proceedings, Annual Conference Series, 1993 (ACM SIGGRAPH '93 Proceedings)* (1993), pp. 147–154.

[Mar98] MARSCHNER S. R.: *Inverse Rendering in Computer Graphics*. PhD thesis, Program of Computer Graphics, Cornell University, Ithaca, NY, 1998.

[MPI] MPI FORUM: *MPI Standard Home Page*. http://www-unix.mcs.anl.gov/mpi/.

[Neu94] NEUBAUER A.: The iterative solution of a non-linear inverse problem from industry: Design of reflectors. In *Curves and Surfaces in Geometric Design* (Boston, 1994), Laurent P. J., Méhauté A. L., Schumaker L. L., (Eds.), A. K. Peters, pp. 335–342.

[PP03] PATOW G., PUEYO X.: A survey on inverse rendering problems. *Computer Graphics Forum 22*, 4 (2003), 663–687.

[PP05] PATOW G., PUEYO X.: A survey of inverse surface design from light transport behavior specification. *accepted for publication at Computer Graphics Forum 24*, 4 (2005), 773–789.

[PPV04a] PATOW G., PUEYO X., VINACUA A.: Reflector design from radiance distributions. *International Journal of Shape Modelling 10*, 2 (2004), 211–235.

[PPV04b] PATOW G., PUEYO X., VINACUA A.: *User-Guided Inverse Reflector Design*. Technical Report TR-IIiA 04-07-RR, Universitat de Girona, 2004.

[PRJ97] POULIN P., RATIB K., JACQUES M.: Sketching shadows and highlights to position lights. In *Proceedings of Computer Graphics International 97* (June 1997), IEEE Computer Society, pp. 56–63.

[RH01] RAMAMOORTHI R., HANRAHAN P.: A signal-processing framework for inverse rendering. In *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 2001)* (August 2001), pp. 117–128.

[SDS*93] 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), pp. 143–146.