Automated Lighting Design For Photorealistic Rendering

Silvana Podaras*
Supervised by: Károly Zsolnai†

Institute of Computer Graphics
Vienna University of Technology
Vienna / Austria

Abstract

We present a novel technique to minimize the number of light sources in a virtual 3D scene without introducing significant perceptible changes to it. The implementation is done as an extension of *LuxRender*, a state-of-the-art, physically based and open-source renderer. The algorithm adjusts the intensities of the light sources in a way that a set of light sources can be substituted by a smaller set, thus enabling to render a similar image with significantly less number of light sources, introducing a remarkable reduction to the execution time of scenes where many light sources are used.

Keywords: radiosity, global illumination, constant time

1 Introduction

With the advance of technology, the use of computer graphics has become an everyday routine in motion picture production. Since the making of "Toy Story", the first feature-length film that was entirely computer-animated, there have been a lot of improvements in technology. Nowadays, it is possible to create images that are almost indistinguishable from reality. One of the key elements in order to make a film look "real" is to simulate physically correct light transport when computing an image.

One of first attempts for such a simulation was made by Appel [1], who introduced the ray tracing algorithm. Although the images rendered with that method were far from photorealistic, the fundamental concept has become the basis of state-of-the-art algorithms. A major problem of photorealistic rendering is the time needed to gain high-quality images, because rigorous mathematical and statistical methods are used to simulate realistic effects. Depending on the desired effects and used hardware, render times can be prohibitively long - it can take up to hours or days to obtain images with satisfying quality [10].

Speeding up the rendering process has thus been a hot topic in science and industry for years, and was usually involving the creation of more efficient sampling strategies or better rendering algorithms. The problem can also be addressed from a different angle to speed up this process by means of reducing the number of light sources used in a scene. This idea came up when considering the way how industry giants like PIXAR use physically based ray-tracing systems in their daily work [4]. In order to achieve not only a physically plausible image, but also adhere to a certain look and feel desired by the artist, many light sources are placed in a scene - even up to hundreds of them [2]. Such vast number of light sources directly affects render time, because more of them have to be sampled in order to get a smooth and converged output. When having so many light sources in a scene, would it be possible to render an (almost) identical image with fewer light sources? For a large number of sources, the answer to this question can hardly be given in a reasonable amount of time when letting a user try out all different settings. Instead, the idea came up to let an algorithm perform the adjustment of the light intensities in order to find a similar result image which uses less lights than the initial setup. The overall concept how such an algorithm could be designed is explained in more detail in Section 3.

This work is founded on the assumption that using less light sources results in faster execution times. In order to show the validity of this assumption, an empirical evaluation on several scenes was done. Those measurements were made by means of testing on five carefully chosen scenes with a varying number of light sources, light sampling strategies and a comparison to the ground truth image. The results are presented in Section 5.

2 Related work

Although various efforts were made to speed up the rendering process, little research has been made to reduce the number of lights used in a scene. The most noticeable work is 'Lightcuts' by Walter et al. [13]. In this approach, lights in a scene are first clustered by spatial proximity and similar orientation. Those clusters get hierarchically organized in form of a binary tree. Every cluster is represented by one of the lights it consists of, and can be further refined to smaller clusters, which also use one light as representative, and so on. To reduce light sources, a "cut"

*spodaras@cg.tuwien.ac.at
†zsolnai@cg.tuwien.ac.at

through the tree is made and only the representative lights of the clusters in the cut are used to illuminate the scene. The representative light in a cluster gets modified in order to approximate the resulting illumination if all the lights in the cluster would be used. If the error of such an approximation is so small that it is not perceivable, only the representative light is used when rendering the scene, else the cluster is refined and more lights are used.

Apart from that approach, little prior work has been done to automatically reduce the number of light sources in a scene. Instead, the more general field of automated lighting design has been a popular topic. This field focuses on computationally adjusting parameters like for example light intensities and emission colors, instead of letting the user handle the fine-tuning.

The majority of methods let the user define a "desired" image as input, and the computer tries to calculate the according parameters to achieve this effect. Schoeneman et al. [11] for example assume that the designers of a virtual scene know where to place light sources, but fail in choosing the right intensities or colors. After having the designer "paint" effects such as shadows and spots of light on a target image, an optimization process is started to determine the settings that match the painted image best. Similar work has been done by Costa et al. [3] and Kawai et al. [6].

A contrary approach is "interactive evolution", which lets the user explore a set of possible solutions that the computer creates. Sims [12] used evolutionary algorithms in combination with user input to generate different sets of plant structures, procedural textures and animations. The quality for each solution is determined by the subjective judgment of the user, before the next evolution step is applied. Thus the user can "guide" the results in a specific direction without having to know about the underlying mechanisms for calculating the parameters.
The design galleries approach of Marks et al. [8] lets the computer set up many different light settings and present them to the user, who can choose the setup that seems most appealing to him.

3 Concept

When rendering a scene which makes use of many lights, eventually the same result image could be achieved by using fewer lights when turning some of them off or when changing their intensities. Trying out all variations manually is not an option, because the vast amount of possible settings would make this a time-consuming and cumbersome task. On the other hand, automatizing this process is relatively simple. An algorithm for this task would have to try out different light settings for a scene and compare the resulting images to an initial image the user wants to achieve. The more similar a new image is to the desired one, and the less light sources are used, the better the solution is.

Despite the idea itself is very simple, it is crucial to understand that the problem space is remarkably high dimensional. When trying to reduce the number of lights by only turning them on and off, a binary integer programming problem has to be solved, which is known to be in the NP-hard complexity class. Although the number of solutions increases with the number of lights used, the total amount is finite (up to $2^{amount\ of\ lights}$). This method would be sufficient for simple scenarios, where several light sources clearly have no contribution to the scene. This could be the case if an artist has created a light source with almost no intensity in the scene, or when a light source gets occluded by some object and suddenly has no contribution to the image. To decide if a proposed solution is close enough to the original image, a user-defined threshold could be set.

In practical applications, such easy scenarios will be the minority of cases. To reduce the overall amount of lights, changing the intensity of some of them before turning others off will be necessary. The solution space in this case varies drastically from the first one: an infinite amount of solutions exists, and each one is a valid image gained with certain light settings. For an arbitrary scene, it is unclear which light intensities have to be changed in what way. Exploring this search space without making more constraining assumptions is a non-trivial task. This problem can be addressed by using classical constrained optimization algorithms. In our work, we have used a genetic algorithm due to its capability to explore such vast search spaces. How such an algorithm was designed for our problem is described in the following paragraphs.

Genetic Algorithms (from now on referred to as "GA") are search heuristics inspired by the processes and mechanisms of natural evolution and can be applied to a large class of practical problems. The only requirement is that solutions for the problem can be encoded in a form that they can be processed by genetic operators such as mutation, crossover and fitness. In a GA, those encoded solutions are also called "chromosomes" [9], [5]. In our case, a possible solution can be represented by a vector of values which stores the intensities of each light. Those values are binary if the lights should simply be turned off and on, or they can encode continuous intensity values if the optimization routine should change the light intensities as well. A collection of initial solutions is generated by assigning the initial light settings to several chromosomes. Then, to gain new solutions, each chromosome gets randomly modified by either mutation or crossover operations. Mutation chooses one index of the vector at random and either flips the bit in binary-mode or adds or subtracts a small value for continuous optimization. The crossover operator simply chooses two chromosomes at random and recombines them at a randomly chosen index to form a new chromosome.

After the initialization, an evaluation step has to take place which determines how good a solution is (in terms of a GA: the *fitness* of a solution has to be evaluated). First, the amount of lights used would have to be calcu-

lated. This is done by summing up all non-zero components of a vector containing the individual light source intensities, i.e. $\vec{X_n} = [a_1, a_2, a_3, ..., a_n]$. This is done either by calculating the l_0 quasi-norm or l_1-norm of a vector:

$$\|X_n\|_0 = \sqrt[0]{\sum_{i=1}^{n} |a_i|^0}, \quad \forall a_i, a_i \in \mathbb{R}, \tag{1}$$

$$\|X_n\|_1 = \sum_{i=1}^{n} |a_i|. \tag{2}$$

The first case would be an integer optimization problem where lights only get turned off or on, the second metric adapts the lights intensities. However, using only the l_0- or l_1-norm of a vector to determine the fitness of a solution would be not sufficient, because it lacks the constraint that the output should be faithful to the input image.

A simple method to determine if a solution is good enough would be to define a threshold, which determines the maximum allowed difference between the initial image and a possible solution image. Calculating the difference is done by subtracting the images pixel-wise from each other and summing up the squared absolute differences. The solution is only considered as valid if the difference lies below the threshold, otherwise it is discarded. For valid solutions, the fitness gets calculated as the weight of a vector.
This trivial formalization is, however clearly not feasible as it leaves two problems unresolved. The first problem is that when creating an invalid solution, the algorithm never gets any kind of feedback on how close it was to a valid one. Not having this knowledge of how "close" a solution is makes optimization no better than any exhaustive sampling technique. The algorithm needs more elaborate feedback in order to know if the optimization is heading in a good direction. The second problem can be depicted by the following scenario: two valid solutions are available, both of them use three lights out of many. The three lights used in the first solution are different from the ones in the second - which of the two solutions is the better one, when both of them have the same fitness?

To solve both problems at once, the difference between the initial image and a temporary solution image is added after calculating the norm - in this way, the faithfulness of the output image is also considered, and with this knowledge, the algorithm can generate better and better solutions in every iteration. The overall fitness $f()$ of a solution vector $\vec{X_n}$ gets calculated as described in equation 3. The first term is the calculation of the norm $\|X_n\|_s$, as explained above in equations 1 and 2. The second term calculates the distance between the two images. This is done by summing up the squared differences of the pixels, where T_{ij} stands for the pixels of "target image" and C_{ij} for the pixels of the "current solution image". There are also two parameters for weighting the lights (p_L) and the difference (p_D). They act as a kind of quality switch and allow more control on the optimization procedure. The contribution to the weight of the vector of each light is multiplied by the factor p_L, therefore better fitness values are achieved if the algorithm tries to turn off lights instead of finding a solution which has little difference to the initial image. If on the other hand, a result image that's very close to the original is desired, the weight for the difference would have to be set to an appropriate value.

$$f(\vec{X_n}) = p_L \cdot \|X_n\|_s + p_D \sum_{i,j} \|T_{ij} - C_{ij}\|^2, \tag{3}$$

where $s \in \{0, 1\}$. Adding the second term to the overall fitness can be done both in integer and in continuous optimization mode. A threshold-value as mentioned above can also be used to let the user control how close a solution must match the original image to be considered as valid.

Thus, an objective quality metric has been defined for determining how good or similar a solution is, and a smaller number encodes a better solution. After having defined the fitness function, the general procedure of the algorithm follows the schema of a typical genetic algorithm. In each generation, new solutions are processed and ranked according to their fitness values. The best solutions are kept by the principle of elitism, while the others are modified in order to gain better solutions from generation to generation. The algorithm would have to find the settings which result in the smallest number representing the quality of a solution. Mathematically speaking, the whole problem can be seen as an optimization problem, where a global minimum has to be found among all possible solutions:

$$\min(f(\vec{X_n})), \quad \forall \vec{X_n} \in \mathbb{R}^n. \tag{4}$$

4 LuxRender

LuxRender is a physically based, state-of-the-art open-source software renderer based on Pharr's and Humphrey's physically based renderer, PBRT [10], which was developed for educational and academic use. *LuxRender* is a stand-alone renderer and not a modeling software. Thus the creation of scenes and models has to be done in other software, and then they have to be exported for rendering.

In 2007, the creators adapted the original source code to make it suitable for artistic use [7]. *LuxRender* implements different state-of-the-art rendering algorithms and provides features such as different material types for objects, post-processing effects, HDR rendering, film response among many others. The implementation presented in this paper works on level of the "Film" stage depicted in figure 1 and makes use of a feature called *Light Groups*. When modeling a scene with several lights, those lights can be associated with a light group. An arbitrary

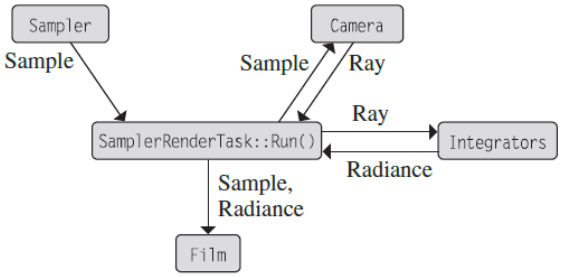


Figure 1: *Basic architecture of PBRT. The Sampler provides the SamplerRendererTask with random samples for BRDF sampling. With that sample, the camera then constructs a ray towards the image plane for the next pixel position and passes it to the Integrator. The integrator calculates the radiance carried along that ray. The collected radiance then gets saved on the film.*
[10]

number of lights - also only a single one - can belong to such a light group. During the rendering process, the light contribution of each group is saved in a separate buffer. Every group also has a intensity and a color temperature, which can be controlled by two parameters. This enables to change the initial light settings in a scene while the rendering is still in process or after it is finished. The user can modify those parameters in the GUI.

5 Results

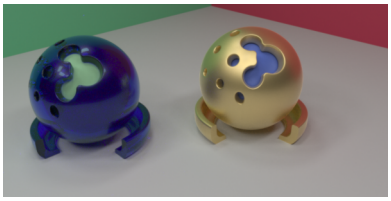
The following section consists of two parts, the first show our results of the empirical test study to verify the assumption that rendering with less light sources increases overall rendering speed. The second part presents the results achieved when using the light source cleaner (further referred to as LSC) on three specially designed test scenes. The scenes were all rendered on a computer with an Intel Core i7-2600K CPU @ 3.40GHz (8 (logical) CPUs), 16 GB DDR3 RAM, and an NVIDIA GeForce GTX 560 Ti with 1 GB of memory.

(a) *Cherry scene* (b) *Watch scene*

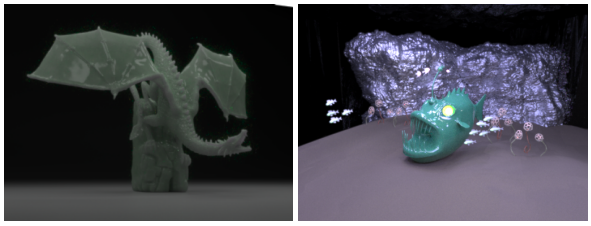
Figure 2: *Two of the scenes used for comparison of render times depending on amounts of lights used. The results are presented in Table 1.*

5.1 Test scenes with many/less lights

To verify the assumption that render time can be saved when using less lights in a scene, test renderings of five scenes of varying complexity were done. The test scenes where designed to make use of up to 47 light sources. Each scene was rendered twice with different light source setup: once only with a set of light sources that have a contribution to the lighting of the scene, and once with 17 additional light sources that have almost no contribution to the scene. With the exception of the School Corridor scene, which was rendered for half an hour due to its complexity, the test scenes were rendered for 10 minutes each. A ground truth image of each scene was also rendered for one hour.
The resulting images were then compared to the ground truth using the root mean square error metric (Table 1). The scenes where less lights are used always have a smaller difference to the ground truth image than the ones with many lights. When rendering the test scenes longer, the difference between using less or many lights gets clearly visible for the human observer, without using a comparison software.



(a) *LuxBalls Scene*



(b) *Dragon Scene* (c) *Fish Scene*

Figure 3: *Scenes used for testing the algorithm.*

5.2 Results of the LSC

We have used several scenes to test our method. In every scene, each light used was assigned to a separate light group - otherwise the light sources can not be manipulated individually.

First, the integer optimization mode was tested with the scenes that were already used in the empirical study. In those scenes, there were "fake" light sources which had no contribution to the lighting of the particular scene. The assumption was that the "unnecessary" lights should be easy to determine, so when running for enough generations, the algorithm should be able to detect all of them and turn them off. This worked fine for the tested scenes and

Scene	Balls		Dragon		Corridor		Cherry		Watch	
Lights	30	30+17UL	20	20 + 17UL	10	10 + 17UL	31	31 + 17UL	11	11+17UL
Path tracing	1040.44	1148.54	224.41	250.42	4918.52	5250.92	2012.82	2238.13	1246.18	1489.03
Bidirectional	1218.10	1333.16	257.46	275.04	2957.23	3827.03	2567.93	3115.10	1220.64	1887.13
Metropolis	1209.58	1347.72	283.97	304.10	3103.88	4082.38	2541.94	3031.82	1408.88	2342.89

Table 1: *RMSE of the test scenes compared to ground truth image. Each scene was rendered two times, once with a certain amount of light sources and a second time with 17 additional "unnecessary lights" ("UL"), which had no visible contribution to the scene. Rendering time was 10 minutes for each scene except for the Corridor scene, which was rendered for half an hour due to its complexity. The ground truth images were rendered for one hour. The table shows that the scenes with less light sources have a smaller difference to the ground truth image than the scenes with many lights. This verifies the assumption that rendering is more efficient when using less lights in a scene by canceling out lights with almost no contribution.*

should also work for any arbitrary scenes where lights are used which have (almost) no contribution. However, we show that our technique is capable of solving more complex scenarios beyond these trivial cases.

For the testing the continuous optimization mode, three additional scenes were modeled and tested (see Figure 3). Two rather simple cases were constructed to show that the algorithm generally works. The first one consists of two beveled spheres on a pedestal standing in front of a wall. Three area lights - two small lights and one big light - were then arranged in the following way: each one of the small lights is half the size and half the power of the big light, and the lights were positioned so that the two small lights together are covering the big light exactly. Also, the lights are placed exactly at the same height. Figure 4 shows a screenshot of the 3D-view of the scene for better understanding. The assumption was that if the algorithm worked correctly, it should be possible to achieve the same lighting for a scene when turning off the small light sources completely and increasing the intensity of the bigger area lights. This makes a simple test case which translates to a high-dimensional optimization problem, for which we exactly know the analytic solution.
To make the whole scenario more challenging, this basic setup of three lights was copied and pasted into the scene several times, so that 12 of those arrangements (this makes 33 lights in total) are present in the scene.

The second scene features a dragon model illuminated by 50 area lights, positioned pairwise on the same position and height with the same light intensity. So in this scene, it should be possible to turn off at least half of the lights when increasing the intensity of the other half. A 6x3 array of those lights illuminates the dragon from top, and 7 lights from the front. The light setup is also rather simple here, but the amount of lights is already high. Figure 5 depicts a screenshot again for better understanding.

The third scene which shows an angler fish is the most complex setup featuring 100 light sources. The majority are blue area lights, and there are also point lights of no contribution hidden behind the big stone wall. The algorithm should be able to both turn off many of the point lights and reduce the amount of area lights also significantly. Contrary to the previous scenes, the area lights are positioned arbitrarily instead of being arranged in a special way. This was done to simulate a scene of practical interest as it occurs in film production.

The initial images were rendered with bidirectional path tracing, and the algorithm was run on each of them several times for 100, 500 and 1.000 generations with 15 chromosomes each and different weighting parameters. At the beginning, some runs which last only 100 generations were made several times to ensure that the algorithm works "right" and gives similar results on each run. Figures 6, 7 and 8 show the result images on the $l1$-norm of the light vector at several stages of the optimization process.

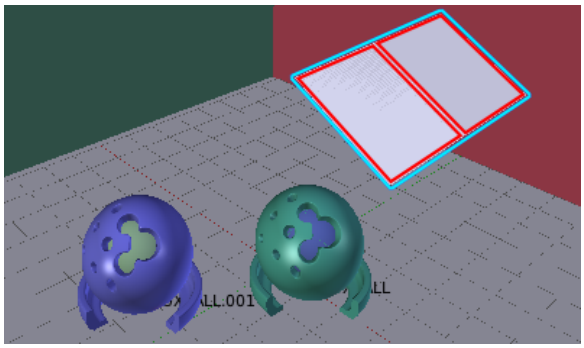


Figure 4: *Example for the light setup in the Luxballs scene. The cyan colored rectangle marks the big area light, while the two red ones mark where the two small area lights are located. The small lights together cover the same area as the big one and are placed exactly at the same height. 12 of those arrangements are present in the scene.*

6 Conclusion and discussion

We presented a light source minimization technique to provide a solution for reducing the overall amount of light sources used in a scene by applying a genetic algorithm to a multivariate optimization problem. A definitive strength is the simplicity of the concept and its general applicability. Although for this paper the implementation was done

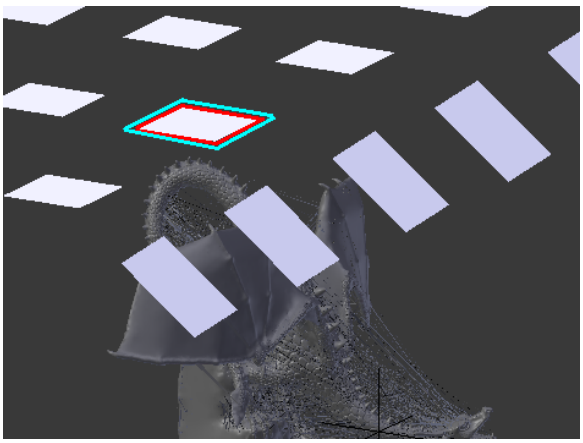


Figure 5: *Example for the light setup in the Dragon scene. The cyan and red rectangles exemplary mark two area lights, which are placed exactly at the same position and have the same intensity.*

in *LuxRender*, this technique can be implemented in any photorealistic rendering engine as long as there is a mechanism that stores the contributions of light sources at different locations. We demonstrated that our technique both passes on scenes with known analytic solutions and also works well on scenes of practical interest. We note that as we are using unbiased and consistent rendering algorithms, it is possible to reduce the number of light sources while the rendering is still in progress.

Another issue is that a global metric is used to measure similarity between two images. As the algorithm basically does a pixel-wise comparison, images with local extrema may impose problems due to the omittance of small local features. Using mean squared error metric instead of the simple difference between the pictures indeed helps, but still there is no means to explicitly consider "important" pixels like highlights or shadows. One possible improvement would be to let the user interactively pre-define which local effects are important and should be kept after optimization.

Our proposed technique is simple to implement, and offers a significant speedup in the execution time of the rendering step in difficult lighting scenarios with a vast amount of light sources.

7 Acknowledgements

We would like to thank Kai Schwebke for providing the LuxTime, Andreas Burmberger for the Cherry Splash, Simon Wendsche for the School Corridor and Peter Sandbacka for the Jade Dragon scene and the LuxRender community for the LuxBall model. The fish scene was created by using various models, we like to thank Paul aka. MajorNightmare for the angler fish, Peter Sandbacka for the sea weed and Chris Monson for the seabed.

References

[1] Arthur Appel. Some techniques for shading machine renderings of solids. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS '68 (Spring), pages 37–45, New York, NY, USA, 1968. ACM.

[2] Per H. Christensen, David M. Laur, Julian Fong, Wayne L. Wooten, and Dana Batali. Ray differentials and multiresolution geometry caching for distribution ray tracing in complex scenes. In *In Eurographics 2003*, pages 543–552. Blackwell Publishers, 2003.

[3] António Cardoso Costa, António Augusto de Sousa, and Fernando Nunes Ferreira. Lighting design: A goal based approach using optimisation. In *Rendering Techniques*, pages 317–328, 1999.

[4] Christophe Hery and Ryusuke Villemin. Physically based lighting at Pixar, 2013. Accessed: 2013-12-04.

[5] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992.

[6] John K. Kawai, James S. Painter, and Michael F. Cohen. Radioptimization: Goal based rendering. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, pages 147–154, New York, NY, USA, 1993. ACM.

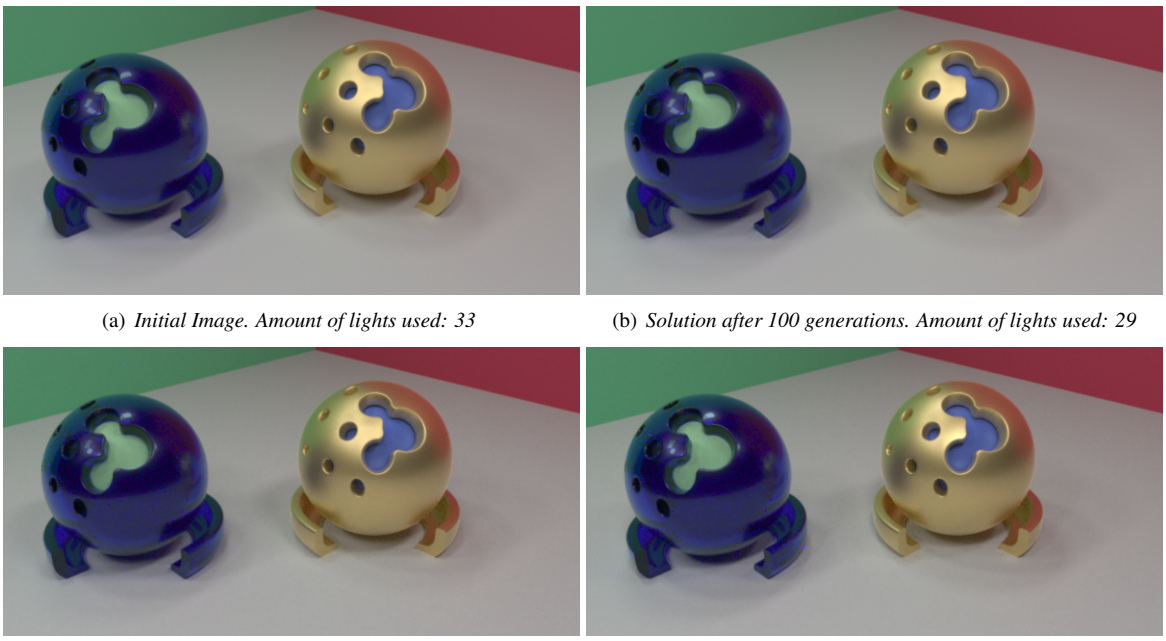
[7] LuxRenderProject. http://www.luxrender.net. Accessed: 2013-12-30.

[8] J. Marks, B. Andalman, P. A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall, J. Seims, and S. Shieber. Design galleries: A general approach to setting parameters for computer graphics and animation. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pages 389–400, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[9] Melanie Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA, 1998.

[10] Matt Pharr and Greg Humphreys. *Physically Based Rendering, Second Edition: From Theory To Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2010.

[11] Chris Schoeneman, Julie Dorsey, Brian Smits, James Arvo, and Donald Greenberg. Painting with light. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, pages 143–146, New York, NY, USA, 1993. ACM.



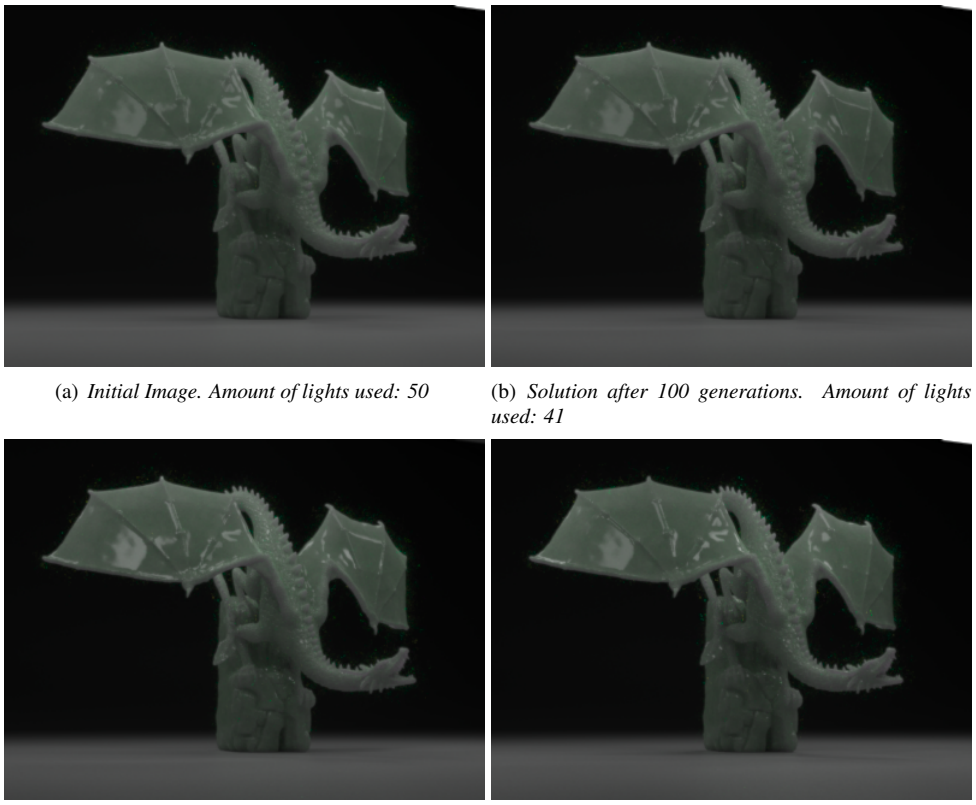
(a) *Initial Image. Amount of lights used: 33*

(b) *Solution after 100 generations. Amount of lights used: 29*

(c) *Solution after 500 generations. Amount of lights used: 11*

(d) *Solution after 1000 generations. Amount of lights used: 12*

Figure 6: *Luxballs Scene. 33 Lights in total.*



(a) *Initial Image. Amount of lights used: 50*

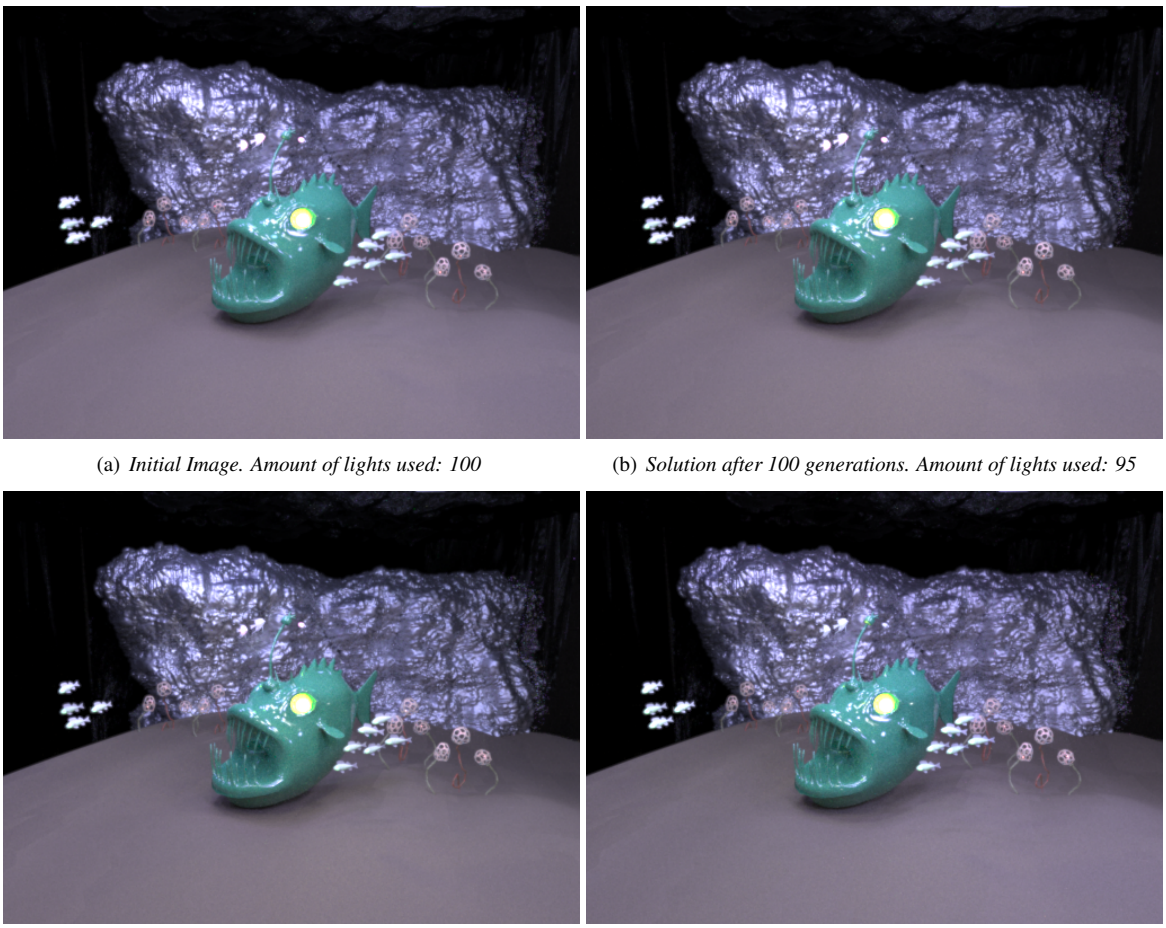
(b) *Solution after 100 generations. Amount of lights used: 41*

(c) *Solution after 500 generations. Amount of lights used:26*

(d) *Solution after 1000 generations. Amount of lights used: 24*

Figure 7: *Dragon Scene. 50 lights in total.*

[12] Karl Sims. Artificial evolution for computer graphics. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '91, pages 319–328, New York, NY,



(a) *Initial Image. Amount of lights used: 100*

(b) *Solution after 100 generations. Amount of lights used: 95*

(c) *Solution after 500 generations. Amount of lights used: 70*

(d) *Solution after 1000 generations. Amount of lights used: 58*

Figure 8: *Fish scene. 100 lights in total.*

USA, 1991. ACM.

[13] Bruce Walter, Sebastian Fernandez, Adam Arbree, Kavita Bala, Michael Donikian, and Donald P. Greenberg. Lightcuts: A scalable approach to illumination. *ACM Trans. Graph.*, 24(3):1098–1107, July 2005.