DOI: 10.1111/cgf.12159
Eurographics Symposium on Rendering 2013
Nicolas Holzschuch and Szymon Rusinkiewicz
(Guest Editors)

Volume 32 (2013), Number 4

Interactive Lighting Design with Hierarchical Light Representation

Wen-Chieh Lin[1], Tsung-Shian Huang[1], Tan-Chi Ho[2], Yueh-Tse Chen[1], and Jung-Hong Chuang[1]

[1]Department of Computer Science, National Chiao Tung University, Taiwan
[2]Shing-Tung Yau Center, National Chiao Tung University, Taiwan

Abstract

Lighting design plays a crucial role in indoor lighting design, computer cinematograph and many other applications. Computer-assisted lighting design aims to find a lighting configuration that best approximates the illumination effect specified by designers. In this paper, we present an automatic approach for lighting design, in which discrete and continuous optimization of the lighting configuration, including the number, intensity, and position of lights, are achieved. Our lighting design algorithm consists of two major steps. The first step estimates an initial lighting configuration by light sampling and clustering. The initial light clusters are then recursively merged to form a light hierarchy. The second step optimizes the lighting configuration by alternatively selecting a light cut on the light hierarchy to determine the number of representative lights and optimizing the lighting parameters using the simplex method. To speed up the optimization computation, only illumination at scene vertices that are important to rendering are sampled and taken into account in the optimization. Using the proposed approach, we develop a lighting design system that can compute appropriate lighting configurations to generate the illumination effects iteratively painted and modified by a designer interactively.

Categories and Subject Descriptors (according to ACM CCS): I.3.4 [Computer Graphics]: Computer Graphics—Graphics Utilities

1. Introduction

Lighting design is a crucial task in computer animation production where it helps to establish the mood of a shot and enhance storytelling. It can be considered as an inverse problem of global illumination [JPP02]. From the theory of global illumination, it is apparent that the relationship between light parameters and the the resulting lighting effect is so complicated and counter-intuitive even for well-trained designers. To meet a lighting effect specified by an artist, a lot of lights are often needed and a lighting designer needs to carefully and repeatedly adjust the parameters of each light until the desired effect is achieved. Such a process is very labor-intensive, time consuming, and often counter-intuitive.

Lighting design based on user-painted illumination effects is thus a more favorable and intuitive way for lighting designers. This problem is conventionally formulated as an inverse lighting problem and solved through an optimization process, which is involved with discrete and continuous optimization since both the number and parameters of lights need to be optimized [PBMF07]. To reduce the difficulty of optimization, most of the existing approaches focus on continuous optimization over lighting parameters and do not deal with the discrete optimization problem of determining the number of lights [PBMF07, KP93, SDS*93, PRJ97].

In this paper, we proposed an approach to achieve discrete and continuous optimization of lighting configurations by integrating a hierarchical lighting representation into a continuous optimization process. Our approach is flexible as a designer can decide whether s/he wants to set the number of lights or not. The designer can either gradually add more illumination effects or sketch all illumination effects at a time. The proposed approach can automatically compute a complete lighting configuration that best matches the desired illumination effect. Furthermore, the illumination effects are painted in the 3D object space, rather than on a 2D image plane. Although specifying illumination effects on an 2D image is easier, painting desired illumination in 3D space makes the computed lighting configuration consistent under different views. This consistency cannot be guaranteed if the illumination effects are given separately on 2D image planes under different views. The consistency of lighting configura-

tion under different views is especially important for interior lighting design and theatrical lighting design applications. In these applications, it is actually more intuitive to specify illumination effects in 3D space.

Our approach consists of two major steps. First, an initial light configuration is estimated by iteratively spreading unit-intensity lights in the scene and then adjusting their intensities and removing the lights with low contribution. In this step, a coarse-to-fine strategy is used for spreading lights and a constrained linear least square solver is used to compute the light intensities. The computed initial lights are then grouped into clusters and from which a light hierarchy is built by repeatedly merging two light clusters that are suitable for combining. In the second step, we derive the optimal lighting configuration by alternatively finding a light cut on the light hierarchy, aiming to obtain as smaller number of lights as possible, and optimizing the positions and intensities of the lights associated with the light cut by using the simplex method.

The contributions of this paper are proposing:

- An automatic lighting design approach that achieves discrete and continuous optimization of lighting configuration given a scene description and desired illumination effect specified by the user.
- An efficient coarse-to-fine sampling method to obtain a good initial guess that makes an optimization process converge fast.
- A scheme based on a light hierarchy to obtain the smallest set of optimal lights by changing the combination of lights and optimizing their parameters using the simplex method.

2. Related Work

Lighting Design Using a Sketch System: Schoeneman et al. [SDS*93] first described a method where a user paints on a scene to be lit with global illumination, and their system solves for the intensities of a set of lights at known positions. As they assume the number and the position of lights are given, the intensities of the lights can be obtained by simply solving a constrained least square problem. Marks et al. [MAB*97] presented a design galleries system, which uses a user interface to assist a designer to explore a very large parameter space to achieve a desired lighting effect. Their system utilized computers as tools for estimating lighting configuration rather than solvers.

Poulin et al. [PRJ97] proposed a sketching interface in which users can specify the constraints of highlights and shadows, including penumbra and umbra, for ellipsoid objects. Their system can automatically compute the positions of point light sources and area light sources. Pellacini et al. [PTG02] developed an effective user interface that allows a user to drag, rotate and scale shadows on screen and their system can move lights or objects as required to generate corresponding change of shadows. Recently, Okabe et al. [OMSI07] presented an interactive illumination brush system that can handle environment light. To produce a realistic image, they represent all-frequency lighting using spherical radial basis function and apply precomputed radiance transfer to render the scene in interactive rate. Although their system performs well on computing environment light, it does not compute local lights, which are important in a lighting design system.

Pellacini et al. [PBMF07] presented a great lighting design system in which a lighting designer paints desired illumination effects on a 2D image plane and their system solves for lighting parameters to achieve the desired illumination. A lighting designer can paint color, light shape, shadows, highlights and reflections using a suite of tools designed for painting light; however, their system only allows a designer to add or delete one light at a time. This feature may not be intuitive to designers as they may paint the illumination effect of multiple lights simultaneously. In some situations, lighting designers may just know how to set up the atmosphere as a whole but are not certainly aware of contribution of individual lights.

Kuo et al. [Kuo08] proposed a lighting-by-guides system whose goal is very similar to ours. Given a scene description and an image as the lighting guide, their system automatically finds a lighting configuration so that the resulting illumination best matches the lighting guide. They proposed an efficient and effective algorithm for guessing initial lighting parameters, and then adjust the lighting configuration using an optimization method. There are two major differences between our and their approach. First, we achieve discrete and continuous optimization by alternatively choosing representative lights and optimizing the parameters of all lights while they do not change the number of lights once the optimization starts, i.e., the discrete optimization is not executed in their system. Second, our approach computes a lighting configuration that is consistent under different views while theirs computes a lighting configuration under a specific view.

Lighting Design with High-level Description: In addition to painting desired lighting effects into the scene, some researchers used the high-level description to guide the inverse lighting problem. Kawai et al. [KP93] proposed the Radioptimization system that allows a user to describe high-level constraints, and then automatically computes lighting intensities and scene reflectance. However, it is not very intuitive for a user to converting high-level descriptions to an objective function. Therefore, it is hard to control the resulting lighting effects in Radioptimization. In addition, to reduce the dimensionality of lighting parameters, Kawai et al. assume that the positions of light sources are given, and only light source intensities need to be estimated.

Costa et al. [CSF99] let users to describe their design goals, which may include different types of constraints or

objectives. The lighting parameters are estimated through optimization by representing users' goals as an objective function. Jolivet et al. [JPP02] use a simple Monte-Carlo method to find the positions of the lights in the direct lighting situation. Their system allows users to specify their desired lighting effects using language description. In general, lighting design using high-level description is not accurate and cannot be controlled by a user easily.

3. Framework

Our approach for solving the lighting design problem is based on the formulation of the inverse lighting problem (section 3.1). Figure 1 shows an overview of our approach. First, we build light cache in preprocessing. After the user inputs the desired illumination using the painting system (section 4), we choose important surface samples from the scene to reduce the computation of our approach. We compute and minimize the illumination difference at these important surface samples since they have greater influence to the rendering results.

We estimate the initial light configuration by iteratively spreading unit-intensity lights in the scene, then adjusting their intensities and deleting lights with low contribution. A coarse-to-fine strategy is used to spread lights and a constrained linear least square solver is used to compute the light intensities. Based on the computed positions and intensities of lights, we further build a light hierarchy by repeatedly merging two light clusters with the lowest illumination difference to the desired illumination.

Finally, we optimize the lighting configuration by repeatedly finding a light cut in the light hierarchy and using the simplex method to compute the optimal lighting positions and intensities in the lighting configuration retrieved from the light cut. The optimization converges quickly as the initial configuration given by the light cut is usually close to the optimal lighting configuration. The details of our approach are described in the following sections.

3.1. Inverse Lighting Problem

We first define the notations used in this paper. Let S be a scene description including material properties and 3D geometry and p be a lighting configuration specifying the positions X and intensities L of all lights in the scene, where $X = (x_1, x_2, ..., x_n)$ and $L = (l_1, l_2, ..., l_n)$ for a lighting configuration contains totally n lights. We then define $I(p,S)$ as the illumination of the scene S produced by a rendering system R using the lighting configuration p. In most situations when the scene is fixed, we will skip S and use $I(p)$ to denote the illumination of the scene. We will also use $I(p,s_i)$ to denote the illumination value at a specific surface sample s_i.

With the above notations, an inverse lighting problem can be defined as follows: given a scene S and the desired illumination Ψ (usually specified by a user), find the optimal

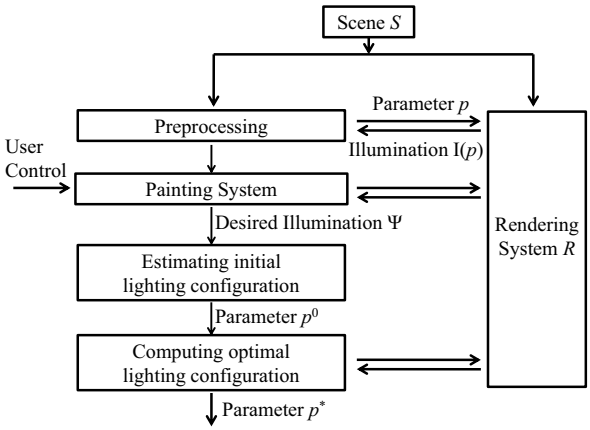


Figure 1: Our approach consists of four stages, preprocessing, painting, estimating initial lighting configuration, computing optimal lighting configuration. The inputs are the description of a scene S, including geometry and material properties, and desired illumination Ψ given by a user. Our system finds an optimal lighting configuration p^* to generate the desired illumination.

lighting configuration p^* so that the differences between the resulting illumination $I(p^*)$ and the desired illumination Ψ is minimized,

$$p^* = \arg\min_p D(I(p,S),\Psi), \tag{1}$$

where $D(I(p,S),\Psi)$ measures the illumination difference between Ψ and $I(p,S)$.

The definition of $D(I(p,S),\Psi)$ depends on the domain where the desired illumination is defined. For instance, Schoeneman et al. [SDS*93] use L_2 norm and Pellacini et al. [PBMF07] use importance-weighted L_2 norm to measure the illumination difference on a 2D image plane since their system attempted to match illumination from a specific viewpoint. In our case, we need to define this error metric in 3D space since the goal of our system is matching the illumination in a 3D scene. Therefore, the illumination difference is defined as

$$D(I(p,S),\Psi) = \|(I(p,S)-\Psi)\|_2$$
$$= \sqrt{\textstyle\sum_{i=1}^{m}(I(p,s_i)-\Psi(s_i))^T(I(p,s_i)-\Psi(s_i))}, \tag{2}$$

where $\|\cdot\|_2$ is the L_2 norm and $\Psi(s_i)$ is the illumination value at the surface sample s_i. m is the number of surface samples at which the illumination differences are measured. Note that the dimensionality of these variables are $I(p,S) \in \Re^{3m\times 1}$, $\Psi \in \Re^{3m\times 1}$, $I(p,s_i) \in \Re^{3\times 1}$, and $\Psi(s_i) \in \Re^{3\times 1}$ as illumination is represented by RGB values. Similarly, the illumination difference of a scene under two lighting configurations p_1 and p_2 is defined as

$$D(I(p_1),I(p_2)) = \|I(p_1)-I(p_2)\|_2. \tag{3}$$

3.2. Estimating Initial Lighting Configuration

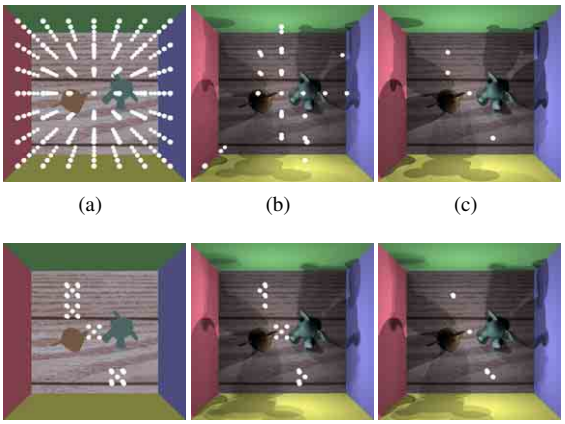


Figure 2: Estimating initial lighting configuration: (a) Spread lights coarsely; (b) Solve a least square problem to determine light intensities; (c) Delete weak lights; (d) Spread lights finely and uniformly around remaining lights; (e) Solve the linear least square problem again; (f) Delete weak lights; The contribution threshold is 5% in (c) and (f).

There are usually tens of thousands of vertices in a scene, in which each vertex's influence to the rendering result is different. Hence, it is time consuming and unnecessary to minimize the illumination difference over all vertices. This computational issue becomes more critical in the optimization stage as we need to render a scene every time for evaluating the objective function. Therefore, we choose important surface samples from a scene and focus on minimizing the illumination differences at these important surface samples. Our system uniformly samples the surface in a scene in preprocessing. Those samples that are in shadows or colored by painting system would get higher weighting. Our system then performs the importance sampling to represent the illumination effect of the scene.

To estimate the initial lighting positions for later optimization process, we develop an approach similar to Kuo et al.'s [Kuo08], which is an iterative framework to find an initial lighting configuration. They spread unit-intensity lights in the space and find the contribution from each light by solving a least square problem that minimizes the difference between the desired illumination and resulting illumination on the image plane. Lights with similar contribution are merged while lights with low intensity are removed. This framework is computationally expensive as it usually needs to repeat the above processes for many iterations.

To reduce the computational time, we apply a coarse to fine sampling scheme to spread lights and only run few iterations of Kuo's framework. At the first iteration, we subdivide the bounding volume of the scene into uniform cells coarsely. We then place a unit-intensity light at the center of each cell as shown in Figure 2(a). The intensities of lights are computed using a least-square solver [LH74] and lights of low contribution are deleted as shown in Figure 2(b)(c). At the second iteration, any cell containing a remaining light is subdivided into 8 subcells uniformly and a light is placed at the center of each sub-cell (Figure 2(d)). The spreading-solving-deleting process is repeated for few times until an initial lighting configuration is obtained. Note that we only use this process to obtain an initial lighting configuration, so few iterations are enough for our purpose. In our experiments, we found that two iterations are usually enough to get a reasonable initial guess. In addition, our method does not perform the merging in each iteration. We extract the merging from the initial guess for later usage (subsection 3.3), which provides the better control over the number of lights.

Once the position of lights are determined, the intensities of lights can be computed by solving a constrained linear least square problem [SDS*93]. Using the same notations in section 3.1, we can express the illumination of the scene rendered with n lights at positions X as a linear combination of n column vectors,

$$I(p) = I(X,L) = \sum_{j=1}^{n} \phi_j l_j, \tag{4}$$

where light intensity $l_j \in \Re^{3\times 1}$, represents a color light. $\phi_j \in \Re^{3m\times 3}$, which collects the RGB values at m important surface samples, is called the contribution matrix of the jth light. $\phi_j = \left[\, I(x_j,(1,0,0),S) \;\middle|\; I(x_j,(0,1,0),S) \;\middle|\; I(x_j,(0,0,1),S) \,\right]$, in which each column is the illumination of scene rendered with only a red, green or blue unit-intensity light placed at x_j, respectively. Thus, the lighting intensities $L^* = (l_1^*, l_2^*, ..., l_n^*)$ for rendering a scene that best matches the desired illumination Ψ in the lest-square sense can be found by solving the constrained least squares problem:

$$\min_L \|(I(X,L) - \Psi)\|_2^2,\ L > 0. \tag{5}$$

The nonnegative constraint is enforced as the intensities of physical lights should never be negative. We use a constrained least square solver *lsqnonneg* provided by Matlab, which adopts the algorithm described in [LH74], to solve Equation (5). Figure 2(b)(e) show the light configuration obtained from least-square solutions.

The computed light configuration is usually not accurate and may contain many weak (or noisy) lights with low intensities. This is because the least-square solver generates some additional lights to diminish illumination differences causing by inaccurate fixed light positions during the solving process. We delete weak lights by removing lights with low contribution ratio,

$$\frac{\|\phi_j l_j\|_2}{\|\Psi\|_2} < \varepsilon_w, \tag{6}$$

where ε_w is a threshold. After deleting weak lights, the

brightness of the scene may slightly decrease when there are many weak lights. To prevent this effect, we run the least square with the remaining lights once more, so the number of lights is reduced while the brightness is maintained. Figure 2(c)(f) are results after deleting weak lights.

3.3. Computing Optimal Lighting Configuration

The initial estimating usually produces more lights than the user desires, although we already delete weak lights. Finding an optimal lighting configuration is indeed involved with discrete and continuous optimization since we need to know the number of lights as well as the position and intensity of each light in the optimization process [PBMF07]. Instead of formulating the problem as discrete optimization, we integrate a hierarchical representation of lighting configuration into a continuous optimization algorithm to compute the optimal lighting configuration. Therefore, the proposed optimization framework can achieve discrete and continuous optimization simultaneously.

Our optimizing framework contains three steps: building light hierarchy, computing optimal lighting on chosen cuts, and determining the number of lights. First, we build a light hierarchy by progressively combining pairs of light clusters, which are obtained from estimated initial light configuration. Then we choose some cuts, which represent light configurations, to attend the optimization. Finally, we determine the number of lights by measuring the improvement of adding lights.

Building light hierarchy: The light hierarchy is built by a bottom-up approach. To maximize the quality of the clusters that a light hierarchy creates, we merge two point light clusters that produce least illumination difference to desired illumination. When two light clusters C_a and C_b are merged, there are three possible positions of the new light cluster: the original position of C_a, the original position of C_b and the weighted average of the position of the two merged clusters:

$$x_{new} = \frac{x_a\|\phi_a l_a\|_2 + x_b\|\phi_b l_b\|_2}{\|\phi_a l_a\|_2 + \|\phi_b l_b\|_2}, \tag{7}$$

Our system always chooses the one with the minimal illumination difference to the desired illumination, and the intensity of the new cluster is $l_{new} = l_a + l_b$. Our system repeatedly merges two light clusters that are best for merging until there is only one cluster. Then the light hierarchy is built. Figure 3(a) shows light clusters of initial light configuration. Figure 3(b) illustrates the corresponding light hierarchy, where leaf nodes represent lights in the initial light configuration.

Once a light hierarchy is constructed, we can represent different lighting configurations as cuts on the light hierarchy. The green line in Figure 3(c) shows a light cut, which means there are three representative lights as shown in Figure 3(d). A light configuration thus can be retrieved accordingly. Our system computes the illumination difference between the desired illumination and the rendering result of a light configuration represented by each cut. The normalized illumination difference $\overline{D}$ is the illumination difference normalized by the L_2-norm of the desired illumination Ψ.

$$\overline{D}(I(cut_i)) = \frac{\|\Psi - I(cut_i)\|_2}{\|\Psi\|_2} \tag{8}$$

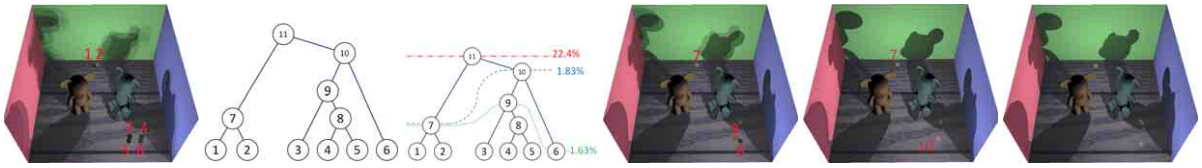
If the computational time is not an issue, we could run optimization for all light cuts; however, it is usually not needed to optimize for cuts with high $\overline{D}$ since these cuts can not achieve acceptable results in most cases. Therefore, we only choose the first N_{cut} cuts with illumination difference lower than a threshold ε_{opt} in the optimization process. In our experience, setting $N_{cut} = 4$ and $\varepsilon_{opt} = 15\%$ works well for most situations.

Computing optimal lighting: For each chosen cut we compute the optimal lighting parameters $p^* = (X^*, L^*)$ by minimizing $D(I(p), \Psi)$, where the initial guess $p^0 = (X^0, L^0)$ is given by the light cut. There are several factors that make our continuous optimization problem difficult to solve [PBMF07]. First, the objective function is nonlinear in the lighting parameters. Second, to evaluate the objective function $D(I(p), \Psi)$, we need to render the scene under p and compute the illumination difference between $I(p)$ and Ψ, which can be computationally expensive. Third, the search takes place in high dimension if we attempts to optimize many parameters or many lights. Fourth, there is no general strategy (other than sampling) to determine the local gradients of the objective function. Based on our experiments and the suggestion in [PBMF07], we found that the nonlinear simplex method [NM65, PaATV92] works well if the optimization starts from a good initial guess and the search space is well parameterized. Figure 3(e) shows the optimal lighting configuration generated by our system. The light positions in our result and the desired illuminated scene (Figure 3(f)) are very close, and the overall atmosphere of lighting effects is similar.

Determining the number of lights: To choose a suitable number of lights, we measure the illumination improvement between a light cut cut_i and its next cut cut_{i+1}.

$$Impro(i) = \frac{\overline{D}(I(cut_i))}{\overline{D}(I(cut_{i+1}))} \tag{9}$$

For example, if we consider the red cut in Figure 3(c) as cut_1, then the next cut under red cut is the blue cut, which is cut_2. The normalized illumination improvement between cut_1 and cut_2 is $22.4\%/1.83\% = 12.24$. Given an improvement threshold, we start from the root of the tree and then progressively refine the light cut until the improvement is lower than the threshold ε_{impro}. We choose 1.6 as the ε_{impro} for our system. For instance, in Figure 3(c), the system will choose the blue cut as our light configuration since its next improvement is $1.83\%/1.63\% = 1.12$ which is smaller than 1.6. The order of light cuts for optimization is chosen in a reversed order of light tree construction. When the light tree is being constructed, the system repeatedly merges two clusters



(a) Initial cofiguration (b) Light hierarchy (c) Cuts and $\overline{D}$ (d) Green cut in (c) (e) Result (f) Desired

Figure 3: Illustration of computing optimal lighting configuration. (a) Initial Lighting Configuration. (b) Light hierarchy built on (a). (c) Three light cuts and their error value after optimization. (d) The lighting configuration corresponds to the third cut (green) in (c). (e) Resulting illumination after simplex optimization. (f) Original input: the desired illumination.

that produce the smallest illumination difference to the desired illumination. Therefore, when determining the order of light cut, choosing the next light cut by breaking a light cluster formed later in the construction process often achieves maximal improvement. For example, in Figure3(c), the system breaks cluster 10 of blue cut into clusters 9 and 6 instead of breaking cluster 7 into clusters 1 and 2 because cluster 10 is formed later. This simple approach works well in our experiments. There are other ways to determine the order of cut, e.g., breaking and testing all possible light clusters to find the next light cut. This may produce better results, but would take much more computational time.

Please note that the *Impro* function is not always greater than one. For some cases, the merged cluster could be better than original two clusters. After the number of lights is determined, our system still keeps the hierarchy. If the number of lights does not satisfy the user's requirement, the user can manually change the number of lights by choosing the upper cut or the lower cut.

Our system supports not only point lights but also spotlights. In the initial estimating stage, we neither create a huge light cache nor spend a long time for precomputing spotlights due to their higher dimensional parameters. Parts of information, which include the position and direction, are acquired in painting stage (Section 4.1). Our system then creates additional contribution matrix for each spotlight. The intensities of point lights and spotlights are computed by the nonnegative least square solver at the same time. In the optimal stage, our system does not merge the spotlighs in the step of building light hierarchy. The spotlights only participate in the computing optimal lighting on chosen cuts. The parameters of spotlights such as position, direction, intensity, and cut of angle are optimized with the parameters of point lights. This ensures point light and spotlight are both considered.

3.4. Rendering System

Our rendering system generates the illumination of the scene S for a given lighting configuration p. It would be invoked hundreds of times during the preprocessing and the optimization process. Thus, it is essential to be able to evaluate the rendering function $R(p,S)$ quickly. To reduce the computational time, we adopt the imperfect shadow map [RGK*08] as our rendering system to render the scene with the first bounce of diffuse lighting since it can interactively compute global illumination in large and fully dynamic scenes based on approximate visibility queries. As the illumination of lights can be stacked, our system can be extended to render higher-order diffuse reflection; this would increase the rendering cost and slow down the interactive light design process.

We precompute and cache the illumination results in the preprocessing. We first uniformly place many unit-intensity lights in the bounding volume of the scene. We then evaluate the illumination of the scene for each unit-intensity light $I(x_j,(1,1,1),S)$ and store these illumination. Since we already have those light cache data, we can obtain the illumination of the scene for a light placed arbitrarily in the scene by interpolating nearby cached lights. The cache data is also used in the optimization. For the direct lighting, accessing the cache data may be slower than rendering the scene directly; however, for the indirect lighting, retrieving the cache data is often faster. Since the interpolation works well on indirect illumination but not well on direct illumination such as shadows or highlights due to rapid changes, we place a lot of lights finely in the space to alleviate the artifact on direct illumination. On the other hand, the illumination of spotlights must be computed during the optimization process, because light cache does not contain the information about spotlight.

4. Interactive Lighting Design System

4.1. Painting System

According to the user study in [KP09], it is harder for a novice user to draw a desired illumination effect that is physically correct. There are usually many conflicts between the user painting and the physically correct illumination. To overcome this issue, we provide a guiding system for the user to specify the desired illumination with ease. Our system helps user to create an inaccurate but feasible desired illumination as input.

The painting system contains three stages: light candidates estimation, brightness adjustment, and detail modification. Figure 4 demonstrates the process. For light candidate estimation, the user can first select an object in the scene as an occluder, and then specify its shadow as the position of

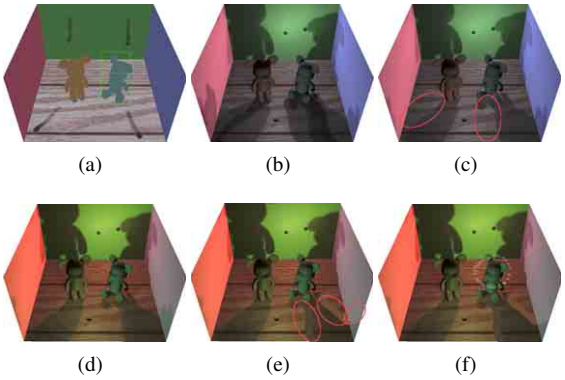


Figure 4: Painting system work flow: (a) Selecting occluder (green line) and assigning shadow (gray strokes); (b) Computing associated point lights (dots); (c) Erasing unwanted shadows (in red circle); (d) Adjusting the brightness and color; (e) Modifying details (such as shadow boundary inside red circle); (f) Assigning spotlight; Please also see the accompanying video.

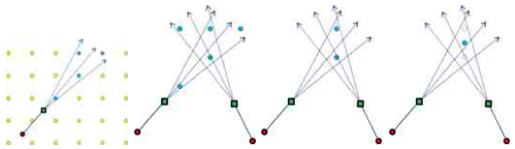


Figure 5: Point lights estimation in Painting System. Green: The center position of an occluder. Red: The position of receivers. Yellow: Rejected point lights. Blue: light candidates. (a) Estimating light candidates from an occluder-receiver pair; (b) Light candidates of two occluder-reciever pairs before reducing; (c) Determining the intersection set; (d) Average position of the set;

the receiver as shown in Figure 4(a). We treat each occluder-receiver pair as a shadow request information. Figure 5 illustrates our idea. Each request contains the position data of the center of the occluder and the center of shadow area. We use a cone to select the point light candidates for each request as shown in Figure 5(a). The apex of cone is the position of occluder center, and the direction of cone's axis is from the shadow center to the occluder center. The initial span angle of cone is 15 degrees. If the cone does not cover enough point light samples, the system will gradually increase the angle of the cone by 5 degrees until it covers enough light samples. In practice, we use 8 as the number of enough point light samples. To reduce the redundant point light candidates estimated from all occluder-receiver pairs, our painting system tries to find the intersection set among those candidate sets. If the intersection set is not empty, we use only the candidate lights in the intersection set as shown in Figure 5(c); otherwise, the system just keeps all candidate lights in each set. After that, each remaining set is reduced to a single light by taking the average of the position of lights as shown in Figure 5(d).

For brightness adjustment, our system can erase the shadow or change the color of lights. When user wants to erase a shadow, he/she can simply click the shadow area as shown in 4(c). Our system casts a ray from a clicked position to the location of each remained point light, then the ray-caster can detect the possible occluder and light source. If there is only one possible pair of occluder and light source casting the shadow on the clicked position, the system directly erases the shadow by setting the occluder to be invisible. If there are multiple possible pairs of occluder and light source, our system allows the user to pick the shadow to be erased. When user wants to change the color of lights, user has no need to paint the entire scene. User can use brush to modify the illumination of a small area, and our system changes the lights which are able to light the area as shown in 4(d). If the area is under shadows, the light sources which cast the shadow do not change. Thus user can modify only part of lights.

For detail modification, our system allows the user to edit the illumination by brush. The user can also select and paint a specified color on the scene to modify the desired illumination as shown in Figure 4(e). Moreover, the user can specify an area to be lit by a spotlight as shown in Figure 4(f). Our system sets the position and direction of the spotlight as those of the current camera, then it finds the minimal cut-off angle that is large enough to cover all the lit area requested by the user.

4.2. Iteratively Adjusting

The proposed approach can be effectively integrated into an interactive lighting design system. A user can create the desired illumination once and our system can automatically compute an optimal lighting configuration. Or the user can iteratively add or modify the desired illumination and our system can adjust the lighting configuration with respect to the latest modification. In this way, the user can better control the lighting design process. Figure 6 demonstrates an iterative editing process.

We develop two adjustment schemes in our system: fine-tune and adding-light. To illustrate these two schemes, we use Ψ_r and $I(p_r^*)$ to denote the desired illumination and the resulting optimal illumination at the current state, respectively. When a user modifies the current illumination $I(p_r^*)$ to a new desired illumination Ψ_{r+1}, our adjustment schemes need to find a new lighting configuration p_{r+1}^*, so that the Ψ_{r+1} is best satisfied.

In the fine-tune scheme, the user can use a brush to edit $I(p_r^*)$. Our system does not add any new light to the scene and only adjusts the intensity and position of the existing lights by re-running the importance sampling and the simplex optimization. The initial guess $p_{r+1}^0 = p_r^*$ and the de-

sired illumination Ψ_{r+1} modified from $I(p_r^*)$ are used in the optimization process.

In the adding-light scheme, our system runs the full process in Figure 1. We use $\Delta\Psi = \Psi_{r+1} - I(p_r^*)$ as the desired illumination to get the initial lighting configuration p_{new}^0. As the illumination should be non-negative, all the negative values of $\Delta\Psi$ are set to zero. Next, a brand-new light hierarchy is built for lights of p_{new}^0. We run the simplex optimization for each chosen cut with the desired illumination $\Delta\Psi$, and determine the number of lights as same as Section 3.3. After obtaining the p_{new}^*, our system uses the light configuration $p_{r+1} = \{p_r, p_{new}^*\}$ to render the scene. The lights of old configuration p_r do not change. This allows user to add lighting effects step by step.

Although our system can determine the number of lights automatically, the user can also opt to control the number of lights in the scene. This can be easily done by choosing a light cut on a light hierarchy.

(a) Input to Pass 1 (b) Result of Pass 1 (c) Input to Pass 2

(d) Result of Pass 2 (e) Input to Pass 3 (f) Result of Pass 3

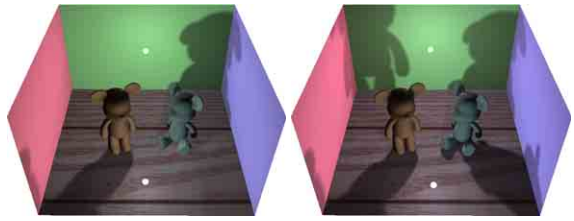
Figure 6: Lighting design in the two toys scene. User can adjust or add light source into the scene iteratively.

5. Experimental Results

We first validate the robustness of our system using physically infeasible illumination as input. We then evaluate the accuracy of our system by finding the optimal lighting configuration in 3D scenes rendered with known lighting configurations. We can quantitatively measure the error of the computed lighting configuration as the ground truth is known. We also test our system on generating desired illumination specified by user. Finally, we analyze the performance of our approach and discuss limitations of our approach.

Validation Experiments: We first use a physically infeasible illumination as input to test the robustness of our system. Figure 7(a) shows the two toy scene with two known lights. Two of the toys' shadows are intentionally removed, and the scene is treated as the desired illumination of our system. In this case, our system can still achieve a good result with two lights as shown in Figure 7(b). The position and intensity of lights are similar to those of the known lights.

Next, we compare our result with the desired illumination



(a) Physically infeasible input (b) Resulting illumination

Figure 7: (a) The physically infeasible input illumination, two of the shadows is removed. (b) The result of our system, the solution is still two lights and similar to the input.

(a) Ground truth (b) Resulting illumination

Figure 8: (a) Interior scene illuminated with 4 artificial lights. (b)The scene rendered using the light configuration computed by our approach.

rendered with a known lighting configuration. Figure 8(a) and (b) show the desired illumination and the computed illumination of an interior scene rendered with 4 artificial lights. The normalized illumination difference $\overline{D}$ is 1.49%, and the difference between two light configurations is shown in Table 1.

In a complex lighting environment, even though we can produce illumination that is very close to the desired illumination, it can be hard to estimate the light positions and the total number of lights accurately. For example, in Figure 8,

Table 1: Comparison of the positions and intensities of the artificial lights and the computed optimal lights in Figure 8. The mean position difference is 6.42, and the mean intensity difference is 0.011.

Artificial Light		
Light	position (x,y,z)	intensity (r,g,b)
1	(-21.95, 110.54, 132.27)	(0.225 0.225 0.225)
2	(-81.95, 100.54, -47.73)	(0.225 0.225 0.225)
3	(8.05, 70.54, -7.73)	(0.225 0.225 0.225)
4	(-111.95, 70.54, 132.27)	(0.225 0.225 0.225)
Optimal Light		
Light	position (x,y,z)	intensity (r, g, b)
1	(-21.95, 99.26, 133.09)	(0.28, 0.31, 0.30)
2	(-82.70, 101.73, -43.34)	(0.21, 0.17, 0.21)
3	(9.35, 74.57, -6.64)	(0.32, 0.33, 0.31)
4	(-113.48, 74.29, 128.71)	(0.20, 0.21, 0.20)

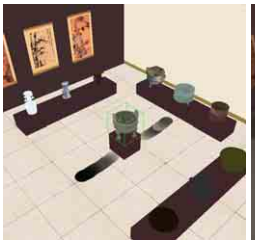
our system outputs correct number of lights, but the lighting positions are slightly different from those of the artificial lights (ground truth). Even so, the resulting illumination is very close to the desired illumination visually. Table 1 compares the position and intensities of the artificial lights and optimal lights. The average distance of position difference is 6.42 or 1.87% of the diagonal length of the scene (342.41) and the average intensity difference is 0.011.

(a) Assigning shadow

(b) Drawing highlighted regions

(c) Final Result

Figure 9: Lighting design in the interior scene. (a) User assigned shadow area. (b) Drawing two highlighted regions for spotlight. (c) Final result with indirect lighting.



(a) Assigning Shadows

(b) Assigning other Shadows

(c) Estimated Result

(d) Drawing highlighted region

(e) Final Result

Figure 10: Lighting design in the museum.

Lighting Design Using Painting System: We tested our system on 3D scenes in which the desired illumination is specified using our painting system. Figure 9(a) shows a user painting some shadows under the chair. Then the user draws two highlighted regions to the vase and the painting on the wall as shown in Figure 9(b). Finally, our system finds the optimal lighting configuration that minimizes the illumination difference to the desired illumination. Figure 9(c) shows the resulting illumination of our system. One can find out that our system produces shadows, point lights, and spot lights that closely match user's desired illumination effects.

Figure 10 shows another lighting design example. User selects the Mao-Gong Ding in the center as a shadow occluder and then specifies the positions of the shadows as shown in Figure 10(a)(b)(c). After that, the user draws a highlighted region on the Mao-Gong Ding. Figure 10(d) and Figure 10(e) shows the assigned painting and displays the final result, respectively.

It is worth to notice that the user does not need to paint the entire scene as our system can automatically guess the lights and create a desired illumination. This ability provides a more convenient way for inverse lighting design, thus it can help the user to reduce the time for editing.

Discussion: We analyzed the computational performance of our approach in all examples. All experiments are run on In-

tel Core I5 760 2.8GHz CPU with NVIDIA GeFore GTX460 graphics hardware. Our program only uses single thread without taking advantage of multi-core CPU. The computational time of our approach is about 10 to 40 seconds, and the normalized illumination difference $\overline{D}$ is lower than 1.5%. The process time mainly depends on the number of lights.

There are some parameters which affect the speed or quality of our system, such as ε_w, N_{cut}, ε_{opt}, and ε_{impro}. These parameters are manually set, and the same parameters are used for all experiments. ε_w is the threshold to delete weak lights for estimating the initial lightings. Because the number of estimated lights affects the computational time of building light hierarchy, it is necessary to set a large enough ε_w; The first N_{cut} cuts with illumination difference lower than a threshold ε_{opt} are chosen to enter the optimization. To prevent local minimum, choosing more cuts is better. However, the computational time of optimization is the bottleneck of our method. It is important to limit the chosen cuts. Setting values of N_{cut} and ε_{opt} is a trade-off between the accuracy and speed; ε_{impro} is the threshold for determining the number of lights. If ε_{impro} is large, the system tends to choose the cut with fewer lights. On the other hand, the system tends to choose the cut with more lights if ε_{impro} is small.

As our optimization process involves discrete and continuous optimization, it is not easy to prove the convergence of the optimization theoretically. In practice, our optimization algorithm converges in all of our experiments since the coarse-to-fine least square solver usually provides a good initial guess, which effectively prevents the simplex optimization from diverging.

6. Conclusion and Future Work

In this paper, we propose an efficient approach for 3D lighting design by solving an inverse lighting problem. We applied our approach to develop an interactive lighting design system where a user can paint desired illumination on a 3D object surface from any view, and our system automatically finds the best lighting configuration to achieve the desired lighting effect. The painted illumination effects can be iteratively modified by the user. In this way, the user has a better control to the lighting design process. The validation experiments show that our approach can correctly compute the lighting configuration for rendering a 3D scene to match the desired illumination. As our system takes only about half minute to process, it provides an efficient way for interactive lighting design.

Currently, our system only considers point light and spot light sources in diffuse environment. In order to make our system more powerful, it is necessary to support environmental light and area lights; however, these types of light sources will greatly increase the search difficulty as the dimensionality of our lighting optimization problem increases significantly. Also, we simply use L_2 norm to measure difference between desired illumination and resulting illumination currently. We would like to adopt perceptual difference as it is a more appropriate metric for human eye. Finally, we will make the user interface more intuitive and improve our system by conducting a user study in the future.

7. Acknowledgement

This work was supported in part by the UST-UCSD International Center of Excellence in Advanced Bioengineering sponsored by the Taiwan National Science Council I-RiCE Program under Grant Number: NSC-101-2911-I-009-101.We also thank Digimax, Inc. for providing test scenes.

References

[CSF99] COSTA A. C., SOUSA A. A., FERREIRA F. N.: *Lighting design: A goal based approach using optimization*. Springer Verlag, Englewood Cliffs, NJ., 1999. 2

[JPP02] JOLIVET V., PLEMENOS D., POULINGEAS P.: Inverse direct lighting with a monte carlo method and declarative modelling. In *Lecture Notes in Computer Science* (2002). 1, 3

[KP93] KAWAI J. K., PAINTER J. S.: Radioptimization: goal based rendering. In *ACM SIGGRAPH* (1993), pp. 147–154. 1, 2

[KP09] KERR W. B., PELLACINI F.: Toward evaluating lighting design interface paradigms for novice users. *ACM Transactions on Graphics* (2009). 6

[Kuo08] KUO H.-T.: *Lighting by Guides: Lighting Parameters Inference from Lighting Guides*. Master's thesis, Dept. of CSIE, National Taiwan University, 2008. 2, 4

[LH74] LAWSON C. L., HANSON R. J.: *Solving Least Squares Problems*. Prentice-Hall, Englewood Cliffs, NJ., 1974. 4

[MAB*97] MARKS J., ANDALMAN B., BEARDSLEY P., FREEMAN W., GIBSON S., HODGINS J., KANG T.: Design galleries: a general approach to setting parameters for computer graphics and animation. In *ACM SIGGRAPH* (1997), pp. 389–400. 2

[NM65] NELDER J. A., MEAD R.: A simplex method for function minimization. *The Computer Journal 7*, 4 (1965), 308–313. 5

[OMSI07] OKABE M., MATSUSHITA Y., SHEN L., IGARASHI T.: Illumination brush: Interactive design of all-frequency lighting. In *Pacific Graphics* (2007), pp. 171–180. 2

[PaATV92] PRESS W. H., ANDSAUL A. TEUKOLSKY B. P. F., VETTERLING W. T.: *Numerical Recipes*, 3nd ed. Cambridge University Press, 1992. 5

[PBMF07] PELLACINI F., BATTAGLIA F., MORLEY R. K., FINKELSTEIN A.: Lighting with paint. *ACM Transactions on Graphics 26*, 2 (Jun 2007), 9:1–9:13. 1, 2, 3, 5

[PRJ97] POULIN P., RATIB K., JACQUES M.: Sketching shadows and highlights to position lights. In *Computer Graphics International* (1997), pp. 56–63. 1, 2

[PTG02] PELLACINI F., TOLE P., GREENBERG D. P.: A user interface for interactive cinematic shadow design. In *ACM SIGGRAPH* (2002), pp. 563–566. 2

[RGK*08] RITSCHEL T., GROSCH T., KIM M. H., SEIDEL H.-P., DACHSBACHER C., KAUTZ J.: Imperfect shadow maps for efficient computation of indirect illumination. *ACM Trans. Graph. 27*, 5 (2008), 1–8. 6

[SDS*93] SCHOENEMAN C., DORSEY J., SMITS B., ARVO J., GREENBERG D.: Painting with light. In *ACM SIGGRAPH* (1993), pp. 143–146. 1, 2, 3, 4