國立交通大學

多媒體工程研究所

碩 士 論 文

光 照 筆 刷 繪 畫 方 式 之 燈 光 設 計

Lighting Design with Illumination Brush

研 究 生：陳岳澤

指導教授：莊榮宏 教授

林文杰 教授

中 華 民 國 九 十 七 年 八 月

光 照 筆 刷 繪 畫 方 式 之 燈 光 設 計
Lighting Design with Illumination Brush

研 究 生：陳岳澤　　Student：Yueh-Tse Chen

指導教授：莊榮宏　　Advisor：Jung-Hong Chuang

林文杰　　Wen-Chieh Lin

國 立 交 通 大 學
多 媒 體 工 程 研 究 所
碩 士 論 文

A Thesis

Submitted to Institute of Multimedia Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

August 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年八月

光照筆刷繪畫方式之燈光設計

研究生： 陳岳澤　　　　　　指導教授： 莊榮宏 博士

林文杰 博士

國立交通大學

資訊多媒體工程研究所

摘要

我們提出一個新的架構來處理燈光設計。使用者可以用光照筆刷直接在 3D 物件上著色，透過系統可以反向找出燈光的設定。這種燈光設計可以歸類於反向燈光問題，給定 3D 場景的描述以及使用者想看見的光照結果，找出光源的相關資訊，包括數量、位置、強度等。而我們的系統可以自動化找出這些光源訊息，協助使用者更方便的做燈光設計。

Lighting Design with Illumination Brush

Student: Yueh-Tse Chen

Advisor: Dr. Jung-Hong Chuang

Dr. Wen-Chieh Lin

Institute of Multimedia Engineering

College of Computer Science

National Chiao Tung University

ABSTRACT

We present a new approach to lighting design with illumination brush. User can paint some illumination on object surface directly and then find lighting configuration backwardly through our system. It is based on the inverse lighting problem of determining light settings for an environment from a description of the desired illumination. Our method is useful for determining lighting layout to achieve a desired effect and can be used in conjunction with any rendering algorithm. Given the description of a scene, including geometry and material, and a desired illumination, our system finds an appropriate lighting configuration automatically. Experiments show that our system can produce good quality lighting setups.

Acknowledgments

I would like to thank my advisors, Professor Jung-Hong Chuang, and Wen-Chieh Lin for their guidance, inspirations, and encouragement. I also want to thank another faculty, Sai-Keung Wong. He gives me many useful suggestions. I am grateful to Tan-Chi Ho for his comments and advices. Thanks to my colleagues in CGGM lab.: Chin-Hsian Chang, Kuang-Wei Fu, Hsin-Hsiao Lin, Ying-Tsung Li, Ta-En Chen, Hong-Xuan Ji, Cheng-Min Liu, Jau-An Yang, Tsung-Shian Huang, and Chien-Kai Lo for their assistances and discussions. It is pleasure to be with all of you in the pass two years. Lastly, I would like to thank my parents for their love, and support.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Lighting plays a crucial role in indoor lighting design, computer cinematograph and many other applications. To achieve high quality lighting, lighting designer often place tens or hundreds of carefully chosen lights, a very labor-intensive process, characterized by repeatedly adjustment of the parameters of each light in order to achieve the desired effect. The process is complicated by the fact that in typical illumination the relationship between the parameters of the lights and the resulting visual effects is often unintuitive even for trained designer. This challenge is exacerbated by the increasing complexity of illumination.

To address this issue, we propose a system wherein the designer can directly paint the desired effects of lights into the scene using an interface similar to that of conventional painting programs such as Adobe Photoshop, leaving the task of finding the best values for the parameters of the given lighting model to the computer. This process not only allows for faster setup of lighting configurations, but often helps designer, trained or not, to achieve higher quality results by letting them work in a more familiar and intuitive interface.

To support this user interaction, our system casts the lighting problem as a high-dimensional nonlinear optimization whose goal is to find the best settings for each light parameter to match the painted input, the desired illumination. The nonlinearity of the problem comes from the nature of lighting, for example, in the behavior of shadows or the responses of arbitrary surface shading.

As most nonlinear optimization problem, our problem requires a initial guess. To address this need, we propose an efficient algorithm to provide an effective initial guess and then do optimization.

Conceptually, our system is similar to three previous system, ”painting with light” [22], ”lighting with paint” [19], and ”lighting by guides” [12]. We solve related problems but each with different assumptions and restrictions. The painting with light system assumes that the number of lights and the positions of lights are known. But, our system does not have such a restriction. The lighting with paint system is an interactive lighting system in which designer paint contribution from a light at a time. It also allows more general lights. However, our system only supports point lights. The lighting by guides system is similar to our system, but it often find much more lights than the number of real lights. Our system find the smallest number of lights as possible.

Conclude those feature described above, our system alleviate many restriction. The system automatically finds parameters for the corresponding light, including the number of lights, positions and intensities for each light. Our need is only a desired illumination painted by designer and designer can achieve what they want very easily.

1.1 Contribution

The contributions of this thesis can be summarized as:

- When painting the desired illumination into the scene, designer often consider the overall atmosphere but not thinking about one light at a time.
- Domain of optimization is on 3D space, not 2D image plane, so designer can paint desired illumination on object surface directly on any viewing.

- Our method is useful for determining lighting layout automatically to achieve a desired effect and can be used in conjunction with any rendering algorithm.
- As other nonlinear optimization problems, the quality of our solutions heavily rely on good initial guesses and we proposes an efficient and effective algorithm to solve inverse lighting problem.
- Our proposed algorithm takes only several seconds on optimization. This advantage can avoid user waiting for long time and reach a interactive system.

1.2 Outline

The rest of the thesis is organized as follows: Chapter 2 gives the literature review, the background of inverse lighting problem and related system for assisting user to determine lighting parameters. Chapter 3 illustrate our proposed algorithm to solve inverse lighting problem. Chapter 4 shows our experimental result from our approach. In the end, conclusions and future work are discussed in Chapter 5.

CHAPTER 2

Background

The system we present shares common goals with a number of previously papers. Therefor, we introduce some related research.

2.1 Interactive Lighting Design

Schoeneman et al.[22] first described a method where the user paints on a scene to be lit with global illumination, and the system solves for the intensities of a set of lights with known positions. They assume the number of lights and positions of lights are given, so the problem can be converted into a constrained least square problem.

Because these lighting parameter mappings are often high-dimensional discontinuous functions, the inference of mapping parameters from their outputs is a difficult problem. Therefor, Marks et al.[13] presented design galleries system, which use a kind of user interface that is very different from the one we propose, in order to address a similar goal, namely, assisting a designer in the exploration of a very large parameter space to achieve a desired effect.

Shadows play important roles in lighting design. Adjusting shadows by adjusting light positions in previous work is not intuitive for users. Poulin et al.[20] proposed a sketching system that users can set constraints of highlights and shadows including of umbra and penumbra, and the system can automatically solve positions of point light sources and area light sources. However, the method is only suitable for ellipsoids. Pellacini et al.[18] proposed a user interface in which users can edit shadows by operations such as dragging, rotating, and scaling shadows on screen to change light sources and object positions. However, the method can only handle simple light sources and scenes. While these methods provide a remarkably intuitive interface for controlling the lighting parameters, they do not address the many different DOFs that would like to control and can be confusing when adjusting the parameters of many lights.

Previous described research only support local light. Okabe et al.[16] present an illumination brush interactive system which can deal with environment light. The system automatically creates the lighting environment by solving the inverse lighting problem. To obtain a realistic image, all-frequency lighting is used with a spherical radial basis function (SRBF) representation. Rendering is performed using precomputed radiance transfer (PRT)to achieve a responsive speed. However, it only support environment light, and local lights is more important in lighting design system.

Pellacini et al.[19] present a lighting with paint system wherein a designer paints desired lighting effect directly into the scene, and the computer solves for parameters that achieve the desired look. The designer can paint color, light shape, shadows, highlights and reflections using a suite of tools designed for painting light. However, only contribution of one light can be added or delete at a time. This might not be intuitive to the designer since they often paint whole illumination at the same time. The designer may know how to set up the feel as a whole but they are not necessarily aware of contribution of each light.

Kuo et al.'s[12] lighting-by-guides system has a goal very similar to ours. Given the description of s scene and an image as the lighting guide, automatically find a lighting configuration so that the resulted rendering best matches the lighting guide. They propose an efficient and effective algorithm for guessing initial lighting parameters, and then adjustment of lighting configuration through optimization method. However, it spends too much time on guess initial lighting parameters and it always estimate more lights than the number of real artificial lights.

2.2 Goal Based Lighting Design

Except for painting desired effect of light into the scene, some research use goal based approach to solve inverse lighting problem. Kawai et al.[9] proposed the Radioptimization system which allows users to describe high-level constraints and their objectives, and then automatically solves light intensities and scene reflectance. However, the system is designed only for radiosity so that it can only be used in diffuse environments. In addition, in order to reduce the dimensionality of parameters and computation, as Schoeneman et al., they assume that positions of light sources are given, and only estimate light source intensities.

Costa et al.[2] allows users to describe their design goals, which may include different types of constraints or objectives, and then by taking goals into cost function, light source parameters can be estimated by optimization. Jolivet et al. [7] use a simple Monte-Carlo method to find the positions of the lights in a direct lighting, Declarative modeling will also be used to allow the designer to describe in a more intuitive way his lighting wishes. However, design goal of these method are often not intuitive to users.

In the field of visualization, Perception-based lighting design has included implicit approaches that aim to maximize illumination entropy for a fixed viewpoint. Gumhold[4] describes a perceptual illumination entropy approach in which he uses limited user studies to model user preferences in relation to brightness and curvature. Shacked et al.[23] proposed a perceptual quality metric combining features such as intensity, contrast, edge ,area intensity, and height of a light source to estimate quality of a rendered image.

2.3 Rendering

There is a vast body of work on computing illumination and shadows from light sources. Most techniques accelerate the processing of individual lights but scale linearly with the number of lights.

Several techniques have dealt explicitly with the many lights problem. Ward[30] sorts the lights by maximum contribution and then evaluates their visibility in decreasing order until an error bound is met. Paquette et al.[17] present a hierarchical approach, which provide guaranteed error bounds and good scalability, but cannot handle shadowing which limits the applicability. Fernandez et al.[3] accelerate many lights by caching per light visibility and blocker information within the scene, but this leads to excessive memory requirements if the number of lights is very large. Wald et al.[28] can efficiently handle many lights under the assumption that the scene is highly occluded and only a small subset contribute to each image. This subset is determined using a particle tracing preprocess.

Instant radiosity[10] is one of many global illumination algorithms based on stochastic particle tracing from the lights. It approximates the indirect illumination using many virtual point lights. Photon mapping[6] is another popular, particle-based solution for indirect illumination. It requires hemispherical final gathering for good results, typically with 200 to 5000 rays per gather.

Walter et al.[29] propose a lightcut system. It renders complex illumination by renders complex illumination by converting the problem into many point lights and using a hierarchical clustering of lights. Our approach also gathers light from point samples and uses a hierarchical clustering.

CHAPTER 3

Algorithm

This chapter describes our algorithm for lighting design using illumination brush. We first give an overview of our approach (Subsection 3.1). Next, we describe the detail of each step of the algorithm, including choose important vertices (Subsection 3.2), estimate initial lighting configuration (Subsection 3.3), compute optimal lighting configuration (Subsection 3.4). Finally, we illustrate our rendering system (Subsection 3.5).

3.1 Overview

We describe the inverse lighting problem formally. Inputs to our system consist of a scene S (including material, geometry) and a desired illumination Ψ painted by user. From these inputs, our system estimates a lighting configuration p so that the resulting illumination $I(p) = R(p, S)$ is close to the given desired illumination Ψ, where R is a rendering system returning the illumination of the scene S under the lighting configuration p, including the number of lights, the positions x_i and intensities l_i of each light. Thus, inverse lighting problem attempts to find the optimal lighting configuration p^* so that

$$p^* = arg\ min\ D(I(p), \Psi) \tag{3.1}$$

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

Figure 3.1 illustrates the workflow of our system, which consists of four components:

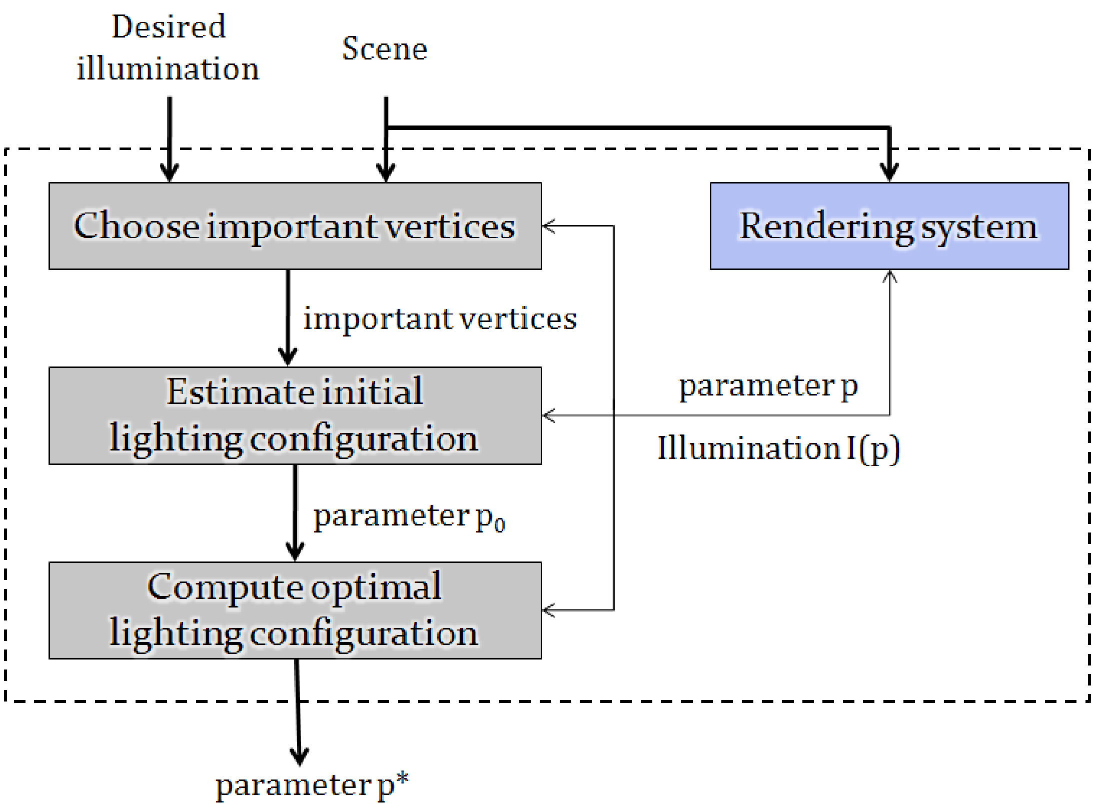


Figure 3.1: System overview. Our system consists of four components, choosing important vertices, estimating initial lighting configuration, computing optimal lighting configuration and rendering system. The inputs are the description of a scene S, including geometry and material, and desired illumination Ψ given by a user. Lighting design system generates an optimal lighting configuration p^* to approximate the desired illumination.

1. Choose important vertices

If we use information of all vertices to do optimization, it spends too much time and disobeys our goal: a interactive system. Therefor, we only choose important vertices from the scene to reduce the computation time in the optimization process.

2. Estimate initial lighting configuration

The desired illumination can be approximated by a linear combination of multiple fixed lights, whose intensities are solved by least-square. The initial guess component estimates light's distribution using this method and generates initial lighting parameters p_0. Usually, a good initial guess leads to a better optimization result.

3. Compute optimal lighting configuration

When initial lighting parameters p_0 are given, we can find a optimal lighting configuration p^* through optimization phase. This resulted illumination is close to the desired illumination.

4. Rendering system

Rendering system needs to be invoked many times for computing contribution of each light, a combination of lights, and the resulting illumination.

We describe an overview of our method to solve the inverse lighting problem in the following. First, we choose some important vertices from the scene to reduce computation time of optimization. Importance can be estimated from geometry and illumination properties. Geometry property is unchanged since the vertex-by-vertex visibility relationship is fixed. As for illumination property, it is changed everytime a user paints illumination.

We estimate light's distribution by spreading unity lights coarsely in the scene specified by a user or our system. Because the positions of lights are known, then the intensities of lights can be uniquely determined by solving a linear optimization problem through a least-square solver. We delete some weak lights which have low contribution to the desired illumination. We then spread new unity lights finely near remaining lights and repeat the same process, least-square solver and deleting weak light. As light's distribution is a set of groups, we can classify lights to several clusters using the segmentation algorithm.

We build a light tree by merging lights with similar contribution. We choose a cut in the light tree to determine the number of representative lights and retrieve lighting configuration p from the tree. We adjust lighting parameters using simplex method. Its output is the optimal lighting configuration p^*

Algorithm 3.1 summarizes our method.

Algorithm 3.1: The algorithm of our system

```
//step 1: Choose important vertices
for each vertex {
          compute the importance
}
Choose vertices based on the importance for each vertex

//step 2: Estimate initial lighting configuration
for( i = 0 ; $i < n$ ; i++) {
          spread unity lights
          p = NNLS(x)                 //non-negative least square
          delete weak lights
}
Classify lights to each group using segmentation method

//step 3: Compute optimal lighting configuration
Build a light tree
Choose a lightcut
$p^*$=Simplex(p)
```

3.2 Choose Important Vertices

There are many vertices in a scene and also too much information for optimization process. Using all information to do optimization is good idea, but it spends too much time and it disobeys our goals: a interactive system. Because of this reason, we must trade off between quality and speed.

Since each vertex has different importance, we can choose some important vertices from the scene to reduce optimization time. Importance of each vertex can be considered as how wide an area can be represented by this vertex. In other words, importance measure influence of lighting respect to nearby region.

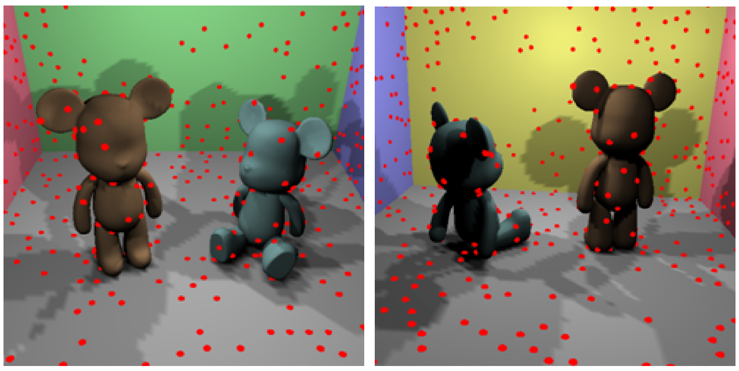
Importance is the weighting sum of geometry and illumination terms

$$Importance(v_i) = w_1 Geo(v_i) + w_2 Illumi(v_i) \tag{3.2}$$

where $Importance$ is a measurement of importance, Geo is a measurement of geometry property, and $Illumi$ is a measurement of illumination property. Geometry property is estimated by visibility of each vertex. Illumination property is estimated by illumination gradient. These two terms will be explained later.

Importance also can be considered as a probability density function. We can use the sampling technique to choose a fixed number of vertices in the scene. In later steps, optimization only operate on these vertices. The number of important vertices will affect quality of the resulting illumination. In general, more vertices often get better results, but take more time on computation. We discuss this issue in the next chapter.

Figure 3.2 is a example of twotoy scene. We choose important vertices (red balls) based on importance for each vertex. We found importance measured by geometry property and illumination property is a good way to estimate importance for each vertex.



(a) Front view of two toy (b) Back view of two toy

Figure 3.2: Importance is weighting sum of geometry and illumination terms . Red balls are selected vertices. (number of selected vertices is 600)

3.2.1 Geometry Property

Having taken a set of high geometry feature vertices, we can estimate over how large the area is viewing from each vertex. For example, illumination at a vertex in the middle of the ceiling of a room is likely to be vary more slowly than it is at the edge of the ceiling where the wall meets. In general, the more objects that are close to the vertex, the greater potential there is for rapidly changing illumination. Therefor, we compute geometry term using measurement of the harmonics mean of the distance, each of the sample rays traveled before intersecting an object (Figure 3.3),

$$Geo(v_i) = D_{max} - \frac{N}{\sum_i^N 1/d_i} \tag{3.3}$$

where D_{max} is diagonal length of bounding box of the scene, $\frac{N}{\sum_i^N 1/d_i}$ is harmonics mean of the distance, N is the number of sample rays and d_i is the distance that the i-th ray traveled. Harmonics mean of the distance serves as an upper bound viewed from each vertex for the estimate. The harmonics mean is a good metric, as a few rays that travel a long distance (or don't intersect anything and have infinity distance, for that matter) can't overwhelm the contributions of close-by distances, which are the most important ones to detect when the illumination value may be changing rapidly. A vertex with high geometry value means there are more objects around it and vice versa.

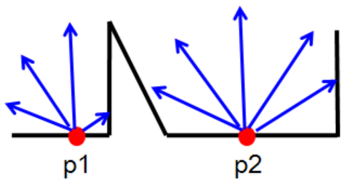


Figure 3.3: Shooting sample rays from a vertex

Geometry term $Geo(v_i)$ is something like visibility ratio of each vertex in ambient occlusion application. But visibility ratio is not good metric to estimate if more objects are around a vertex. It only knows occluding ratio in the viewing from each vertex, so we use harmonics mean of the distance to measure how long distance objects are around it. Figure 3.4 is twotoy scene rendered using geometry value. The edge of the ceiling and feet part of toys are high illumination gradient regions since more objects are around them.

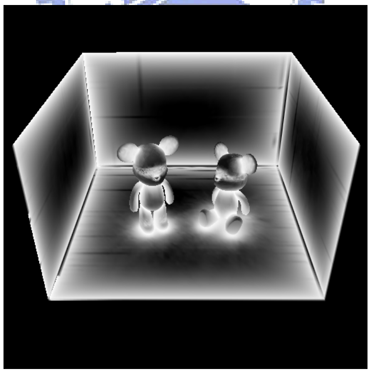


Figure 3.4: Twotoy scene is rendered using geometry value. White regions mean more objects are around them; black regions mean few objects are around them.

3.2.2 Illumination Property

The illumination painted by a user provides hints for rendering effect, such as shadow boundaries and high illumination gradient region. In general, human eyes are sensitive to high illumination gradient region so that if we could catch these regions, it would increase the results generated by our system. Therefore, we should choose vertices in these regions as many as possible.

Our system can automatically detect those regions with higher illumination gradient. Convert vertex color information from the RGB color space to the HSV color space. HSV color space corresponds closely to the human perception of color and it has revealed more accuracy to distinguish shadows or shadings. Figure 3.5 (a)(b) is a example that convert desired illumination Ψ from RGB to HSV. Figure 3.5(b) is rendered using only V channel.

After converting to HSV color space, we need to calculate illumination gradient on the discrete mesh. Xu et al. [31] proposed a framework to solve several surface modeling problems, including mean curvature flow, averaged mean curvature flow, surface diffusion flow, and even higher order flows. Let f be a illumination V function on a surface, then Δf is approximated over a triangle mesh by

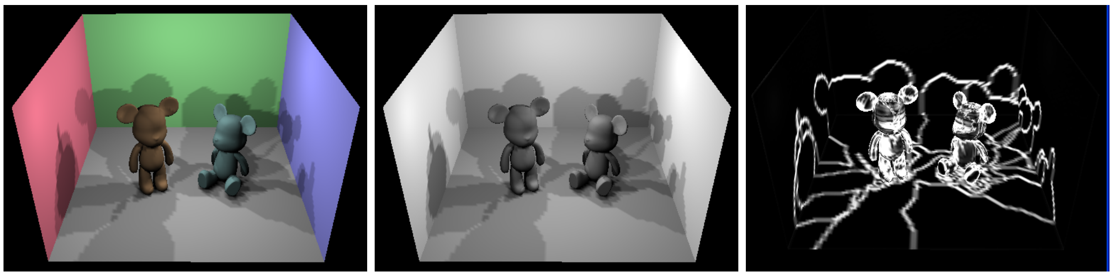
$$\Delta f(p_i) = \frac{1}{A(p_i)} \sum_{j \in N_1(i)} \frac{cot\alpha_{ij} + cot\beta_{ij}}{2} * [f(p_j) - f(p_i))] \tag{3.4}$$

where $N_1(i)$ is the index set of 1-ring of neightbor vertices of vertex p_i, α_{ij} and β_{ij} are the triangle angles shown in Figure 3.6 (Left). $A(p_i)$ is the area for vertex p_i as shown in Figure 3.6 (Right), where q_j is the circumcenter point for the triangle $[p_{j-1}p_jp_i]$ if the triangle is non-obtuse. If the triangle is obtuse, q_i is chosen to be the midpoint of the edge opposite to the obtuse angle.

Our system can detect high illumination gradient regions using this method shown in Figure 3.5 (c). White regions have high illumination gradient.

3.3 Estimating Initial Lighting Configuration

This section describes how to estimate light's distribution and a initial guess for lighting configuration. we spread unity lights coarsely in the bounding volume of the scene. Because the positions of lights are known, then the intensities of lights can be uniquely determined by solv-



(a) Desired illumination (b) Convert RGB to HSV. Show V channel in the picture. (c) High gradient regions

Figure 3.5: Detection of high gradient regions

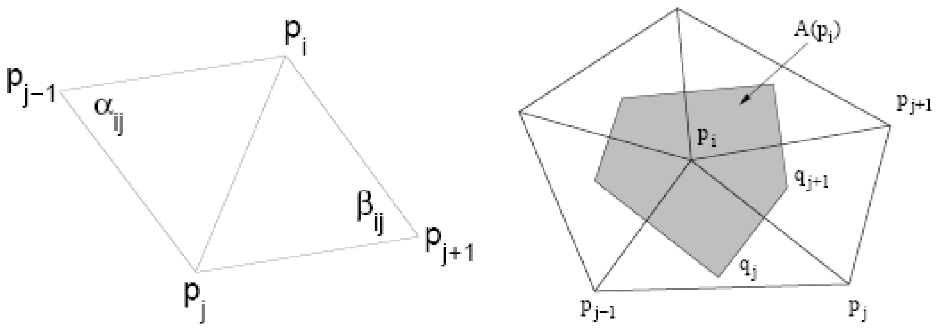


Figure 3.6: Left: The definition of the angle α_{ij} and β_{ij}. Right: The definition of the area $A(p_i)$)

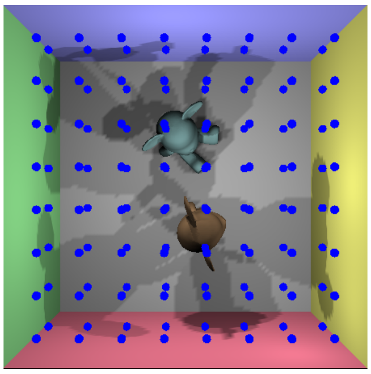
ing a linear optimization problem through least-square solver [22]. We delete some weak lights of low contribution with respect to the desired illumination. Again, we spread new unity lights finely near remaining lights and do the same process: least-square solver and deleting weak lights. We found that lights are distributed over the space as groups so we can classify lights to several clusters through segmentation method. Details are described in the following.

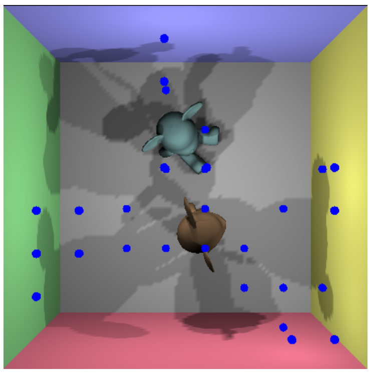
3.3.1 Estimate light's distribution

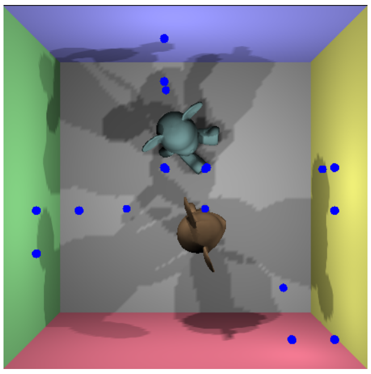
Kuo et al.[12] propose a iterative framework to find a initial lighting configuration. It spreads unity lights in the space, solving a linear optimization by least square, merging lights with similar contribution, filtering out some weak lights. The initial lighting parameters are solved by these four steps iteratively and unity lights will move to regions of high lighting. In general, it needs many times of iterations since it determines the number of lights based on this strategy. Therefor, we found that this approach spends too much time on finding light's distribution and classifying lights to each group automatically.

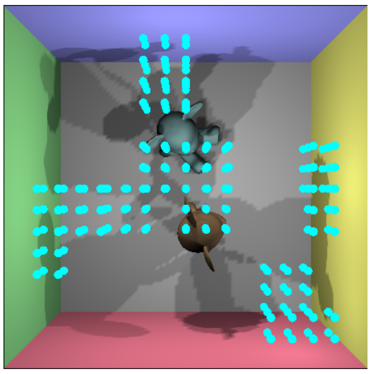
We propose a new framework to improve these drawbacks. If we uniformly spread a lot of unity lights in the bounding volume of the scene, then in principle we can solve the huge linear system to obtain the initial lighting configuration. However, there is a problems with this approach, making it unpractical. The resulting system is too big to be solved efficiently even if it can be solved.

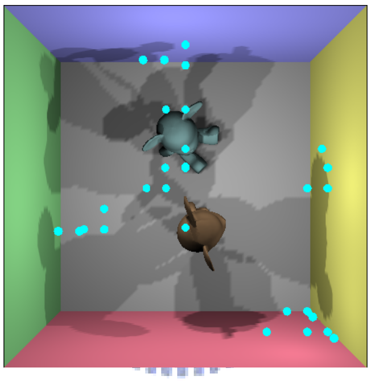
Instead, we spread lights few times. First iteration is spreading unity lights coarsely. The intensities of lights can be uniquely determined by least-square solver and we delete some lights of low contribution. Other iterations are spreading new unity lights finely near remaining lights of previous iteration. We do the same process: solving and deleting weak lights. This approach balances between computation time and the accurate solution, and corresponds to our goal: a interactive system. We don't rely on spread-solve-delete process to help us to determine the number of lights. We only use this approach to reduce computation time of least-square and find light's distribution, so few iterations are enough for our purposes.

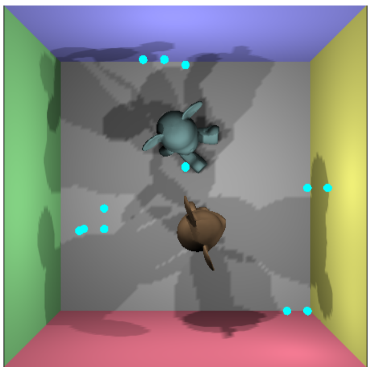

(a) Spread unity lights coarsely

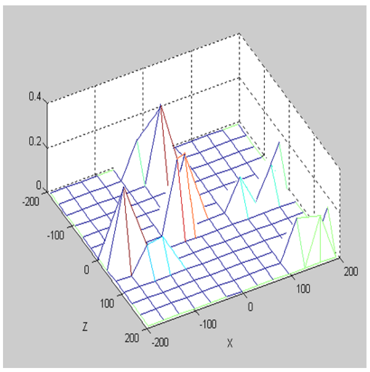

(b) Least-square solver


(c) Delete weak light


(d) Spread unity lights finely


(e) Least-square solver


(f) Delete weak light



(g) Light's distribution

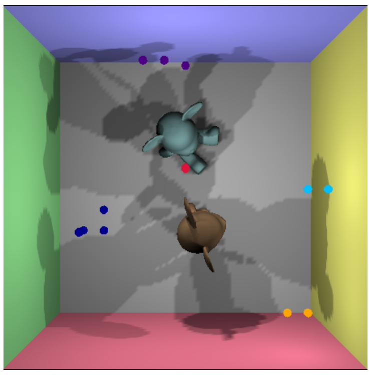

(h) Segmentation

Figure 3.7: Initial lighting configuration workflow

3.3.1.1 Spread unity lights

We uniformly spread unity lights coarsely ,whose intensities are unit, in the bounding volume of the scene shown in figure 3.7(a). For explanation convenient, unity lights are only distributed in the ceiling. Except for the first iteration, we spread new unity lights finely near remaining lights of previous iteration. Figure 3.7(d) is the second iteration of spreading lights.

3.3.1.2 Least Square Solver

Because of the additive nature of light, if the positions of lights are known, then the intensities of lights can be uniquely determined by solving a linear optimization problem[22]

Assume that there are n lights and there are m vertices in the scene S. These lights consist of a lighting configuration p which includes number of lights n, and six parameters, three for position x_i, and three for intensities l_i, for each light. We can obtain illumination of each light through rendering system and arrange RGB of each vertex as $3m$ dimension vector ϕ_i, called contribution vector. The illumination of the scene using these n lights of known positions can then be written as a linear combination of these n column vectors, $\sum_i l_i\phi_i$, where l_i is the intensity of the i-th light. Thus, the optimal intensities $I = (l_1, l_2, ..., l_n)^T$ to match the desired illumination Ψ can be found the least square solution to $AI - \Psi$, where $A = (\phi_1, \phi_2, ..., \phi_n)$ is a matrix of size $3m \times n$.

Because the solution I of least square solver has negative intensities. In general, the intensities of lights are always positive number physically. When we get the optimal intensities I, we must add positive constrains to sure that the solution corresponds to physical meaning. For explanation convenience, $l = NNLS(x)$ notation is represented as solution of non-negative least square solver. Figure 3.7 (b)(e) are solutions solved by least-square.

3.3.1.3 Delete weak lights

We delete lights of low contribution since they are noisy lights produced by least-square. When the intensities of unity lights are solved by least-square, we already place lights in fixed positions and their positions are different from the positions of actual lights. Therefore, least-square generates some noise lights to diminish illumination difference. Unfortunately, these noise lights should not be exitance in the scene and have low contribution to the desired illumination. Thus, we delete weak lights only if it satisfies the following condition:

$$\frac{\| l_i \phi_i \|}{\| \Psi \|} < \epsilon_w \tag{3.5}$$

where the contribution under the ϵ_w is a weak light. Figure 3.7 (c)(f) are results after deleting weak lights.

3.3.2 Segmentation

We found that lights are distributed over the space as groups so we can classify lights to several clusters through segmentation methods.

The positions and intensities for each light are known. We can formulate the relation between positions x and intensities l as a $l_i = f(x_i)$ equation, where f is mapping function between x and l. For explanation convenience, assume that all lights are placed on the same plane. Thinking about f in another way, f function can be figured as height map, where height means intensity shown in figure 3.7 (g).

Watershed segmentation [27] is a common segmentation algorithm. It's concept is very simple and easy to implement. First, we inverse the intensities for each light and build topographic surface based on relation between the positions and intensities. If we flood this surface from its minima, and if we prevent the merging of the waters coming from different sources, we partition the surface into two different sets: the catchment basins and the watershed lines. Each catchment basin region is represented as a light cluster.

We can extend lights on the same plane to 3D space. Watershed segmentation also can apply to 3D space. f function is a high dimensional height map function. Lights are dealt with the same process. Using segmentation technique is a good way to partition lights to clusters globally avoid merging locally. Figure 3.7 (h) is the result after watershed segmentation. Different colors mean different clusters. The resulting clusters are initial lighting configuration p_0.

3.4 Compute Optimal Lighting Configuration

When a initial lighting configuration p_0 is given, lights can be organized into a light tree for efficient partition finding. When building the light tree, similarity is a criterion for merging two clusters. Simultaneously, we also find a new light to replace the contribution of two children.

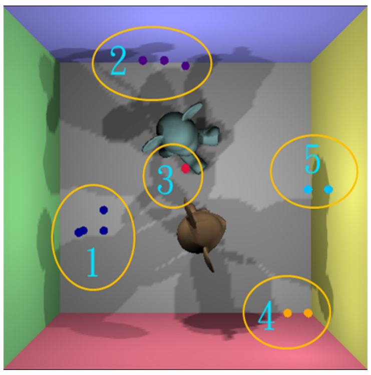
In order to choosing a cut from the light tree, we start with a very coarse cut (e.g., the root of the light tree) and then progressively refine it until our error threshold met. We call such a cut, a lightcut. When we decide a cut from the light tree, it means our system generates appropriate number of representative lights. Certainly, this lightcut can be adjusted by user if they want to add more lights or delete some lights.

The lightcut imply a lighting configuration p, which is not optimal solution. Our system tunes lighting parameters using simplex method and it generates optimal lighting configuration p^*.

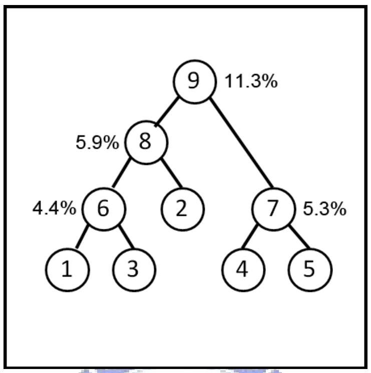
3.4.1 Build Light Tree

The light tree groups lights together into clusters. Ideally, we want to maximize the quality of the clusters it creates. Light tree is built using a greedy, bottom-up approach by progressively combining pairs of clusters.

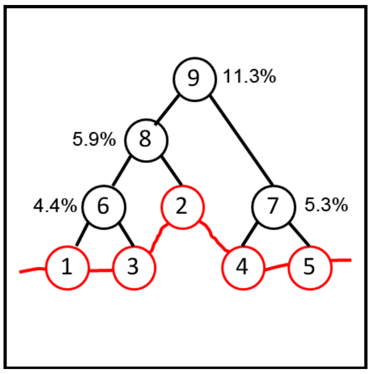
We choose the pair that will create the smallest cluster according to similarity metric. if one light's contribution to the rendering can be faithfully represented by another light, we say that those two lights have similar lighting behavior. Thus, we define our similarity metric as $D(I(p_1), I(p_2))$, where first cluster imply lighting configuration p_1 and second cluster imply another one. Be attention to this similarity metric, before measurement similarity, we must set the intensities of p_1 and p_2 as unity because we care about only their replacement, not their intensities. Figure 3.8 (a) show several clusters after watershed segmentation. Figure 3.8 (b) is a corresponding light tree, where leaf nodes are clusters described above.



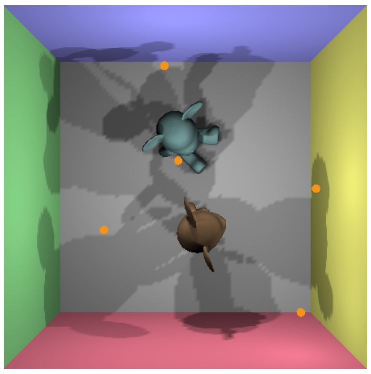
(a) Clusters after segmentation



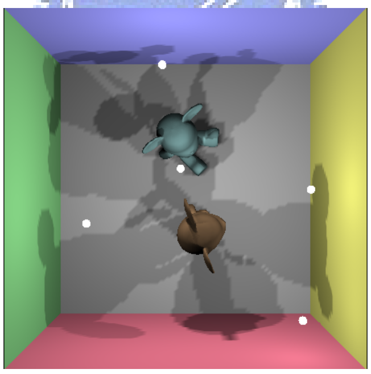
(b) Light tree



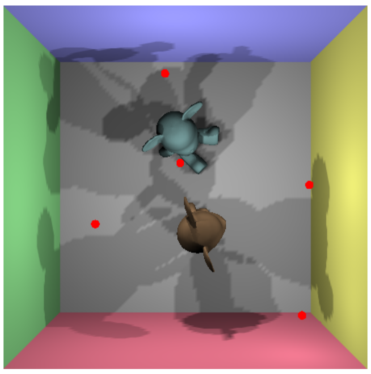
(c) a lightcut



(d) Lighting parameters determined by lightcut



(e) Simplex optimization



(f) Desired illumination

Figure 3.8: Optimal lighting configuration workfolk

Schoeneman et al. use L_2-norm[22] or Pellacini et al. use importance-weighted L_2-norm[19] to measure the difference between the desired illumination Ψ and the resulting illumination $I(p)$ on 2D image plane.

But, our system provide brushes to paint lighitng effect into the 3D scene directly so that our error metric is defined on 3D space. we implement L_2-norm of the difference between the desired illumination Ψ and the resulting illumination $I(p)$ on each vertex, normalized by L_2-norm of the desired illumination Ψ:

$$D(I(p), \Psi) = \frac{\parallel I(p) - \Psi \parallel}{\parallel \Psi \parallel} = \frac{\sqrt{\sum_{i=0}^{m}(I(p, v_i) - \Psi(v_i))^2}}{\sqrt{\sum_{i=0}^{m}\Psi(v_i)^2}} \tag{3.6}$$

where $\|.\|$ is L_2-norm notation, $I(p, v_i)$ is color of i-th vertex under lighting configuration p and $\Psi(v_i)$ is color of i-th vertex in the desired illumination Ψ. In addition to comparison with contribution of $I(p)$ and Ψ, difference measurement also apply for comparison with contribution of p_1 and p_2, where p_1 is a llighting configuration and p_2 is another one.

Beside merging two clusters, we also find a new cluster to replace contribution of two children. We apply weighted average for generating new position by this formula

$$x_{new} = \frac{x_i \parallel l_i \phi_i \parallel + x_j \parallel l_j \phi_j \parallel}{\parallel l_i \phi_i \parallel + \parallel l_j \phi_j \parallel} \tag{3.7}$$

and new intensity by this formular

$$l_{new} = l_i + l_j \tag{3.8}$$

3.4.2 Choose a Lightcut

When light tree is built using bottom-up approach, we want to decide a lightcut to determine the number of representative lights and acquire a lighting configuration p. Using a relative similarity metric assists us in deciding the cut. We start from the root of the tree and then progressively refine it until our error threshold met shown in figure 3.8 (c). Figure 3.8 (d) is a lighting parameters retrieved from light tree according to selected lightcut.

3.4.3 Optimization

At the heart of our system is an optimization algorithm whose purpose is to unburden the designer from having to optimize the light parameters manually. The goal function $D(I(p), \Psi)$ for the optimization is illumination difference between the desired illumination Ψ and the resulting illumination $I(p)$.

Several factors make this a difficult function to optimize analyzed by Pellacini07 et al [19]. First, it is nonlinear in the parameters of the lights. Second, to evaluate Goal function $D(I(p), \Psi)$, we render illumination of the scene under p and compare illumination difference between $I(p)$ and Ψ, which can be expensive in the inner loop of an optimization. Third, if we are optimizing many parameters or many lights, the search takes place in high-dimension. Fourth, there is no general strategy (other than sampling) to determine the local gradients of the goal function. Fifth, our search space has large plateaus in which the optimizer can get lost.

Our system is designed to work with any optimization algorithm, but, as a practical matter, we found that the nonlinear simplex method of Nelder et al.[15] and Press et al.[21] works well provided that (1) the search space is parameterized well, and (2) it starts with a decent configuration. For a search in n dimensions, this method stores a set of n + 1 configurations representing a simplex in the search space. At every step, the optimizer considers a set of possible configuration changes for the worst vertex (the one with the worst value in the goal function), such as reflecting the vertex through the opposite face in the simplex. Each considered change requires one evaluation of the goal function. If no possible improvements are found, the simplex shrinks by uniform scaling toward the best vertex. When the simplex has collapsed to a very small size, it has found a local minimum.

This solution is typically good enough for tuning lighting parameters. Inputs of this component is a lighting configuration p, including n lights and six parameters, three for position x_i, and three for intensities l_i, for each light. Simplex optimizer $p^{'} = Simplex(p)$ adjusts lighting parameters, where $p^{'}$ is a tuned lighting configuration. Figure 3.8 (e) is the optimal lighting configuration generated by our system. we can compare the resulting illumination with desired illumination. Their positions are vert close, and overall atmosphere of lighting is consistent between desired and resulting illumination.

3.5 Rendering System

The rendering system is responsible to evaluate the illumination of the scene S for a given lighting configuration p. It will be invoked hundreds of times during the optimization process. Thus, it is essential to be able to evaluate the rendering function $R(p, S)$ quickly.

Since our system support painting on object surface directly from different viewing, if we compute lighting effect at run-time, our system is impossible to achieve interactive time. This disobeys our original motivations and goals. Therefor, we adopt a simple way to cache lighting information.

We uniformly place many unity lights in the bounding volume of the scene, where a unity light means its intensity is $(1, 1, 1)$ of RGB channel. We evaluate the illumination of the scene for each unity light $R(x_i, (1, 1, 1), S)$ and store these lighting information in the preprocess. Since we have these lighting cache data, we can evaluate illumination of the scene for a arbitrary light placed anywhere by interpolation of nearby cached lights.

This method apply well to indirect illumination and is not well for direct illumination such as shadows or highlights because direct illumination changed rapidly. Thus, we place a lot of lights in the space to alleviate these artifact. Actually, our system can be conjunction with any vertex lighting algorithm for better lighting effect, including those that account for interreflection and shadows such as local radiance transfer [11].

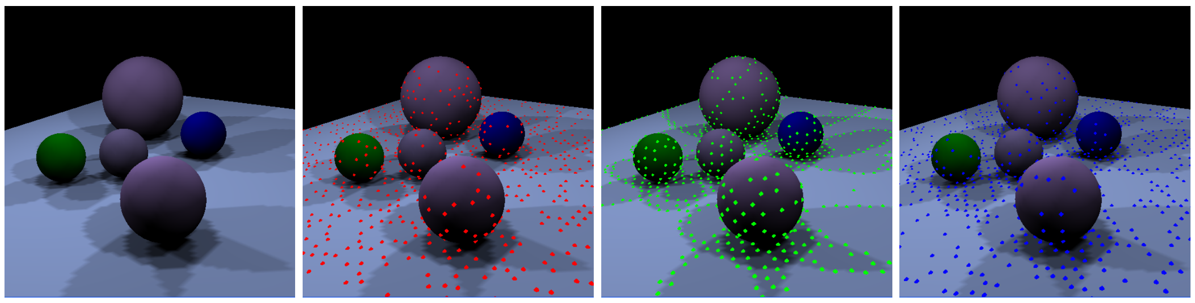
CHAPTER 4

Result

All experiments are performed on Intel core 2 duo CPU E6750 2.66GHz using NVIDIA Gefore 8600 GT graphics hardware. The program only use one thread without taking advantage of multi-core of the CPU.

Figure 4.1(a) shows fiveballs scene, whose illumination produced by five artificial lights specified by our. It can be considered as ground truth in order to comparison with our resulting illumination. This scene is not closed model and five diffuse balls in the scene, so it is convenient for testing, visualization, and evaluation error. We use ground truth as desired illumination and then generate the resulting illumination through our algorithm. Figure 4.1(b)(c)(d) show that different importance criterion will produce different sampling results. If we only consider geometry property for each vertex, more selected vertices are distributed in the region, where many occluders around them. If we only consider illumination property, we can observe more selected vertices near high illumination gradient region. Therefor, we combine advantages of both properties to do sampling.

Figure 4.2 shows the workflow of estimating initial lighting configuration and computing optimal lighting configuration. Figure 4.2(a)(b)(c) is first iteration, including spreading unity



(a) Ground truth (b) only geometry term (c) only illumination term (d) importance term

Figure 4.1: Illustration of choose important vertices. (a) is desired illumination simulated from 5 artificial lights. (b) considers only geometry term for important sampling. (c) considers only illumination term for important sampling. (d) is sampling result considered by both geometry term and illumination term

lights coarsely, solver, and deleting weak lights. Its goal is to approximate the locations and directions of lights. Next, we do second iteration just like as first iteration shown in figure 4.2(d)(e)(f). Only be attention to spreading unity lights finely around remaining lights in previous iteration. Its goal is to locate light's distribution, which help us determine lighting parameters. When light's distribution is known, we cluster lights to each group using segmentation methods shown in figure 4.2(g). Although segmentation find local maximal well, but it has a potential problem, over-segmentation ,so we sometimes observe that some lights should be clustered into the same group, but they are classified into different groups. In our application, over-segmentation is not a big problem because it can be solved by our light tree mechanisms.

After segmentation, we know light's distribution and each light belonging to which clusters. We use these information to build light tree. Figure 4.2(h) shows each leaf node in light tree. In this figure, there are 7 lights which are leaf nodes in light tree. According to error threshold, we choose a lightcut which also imply a lighting parameters shown in figure 4.2(i). The lighting parameters are adjusted globally using simplex method and white balls are final resulting positions of lights shown in figure 4.2(j). Figure 4.2(k) is ground truth produced by our artificial lights shown in red balls. Compare the positions of white balls with red balls. We hardly observe difference between them.

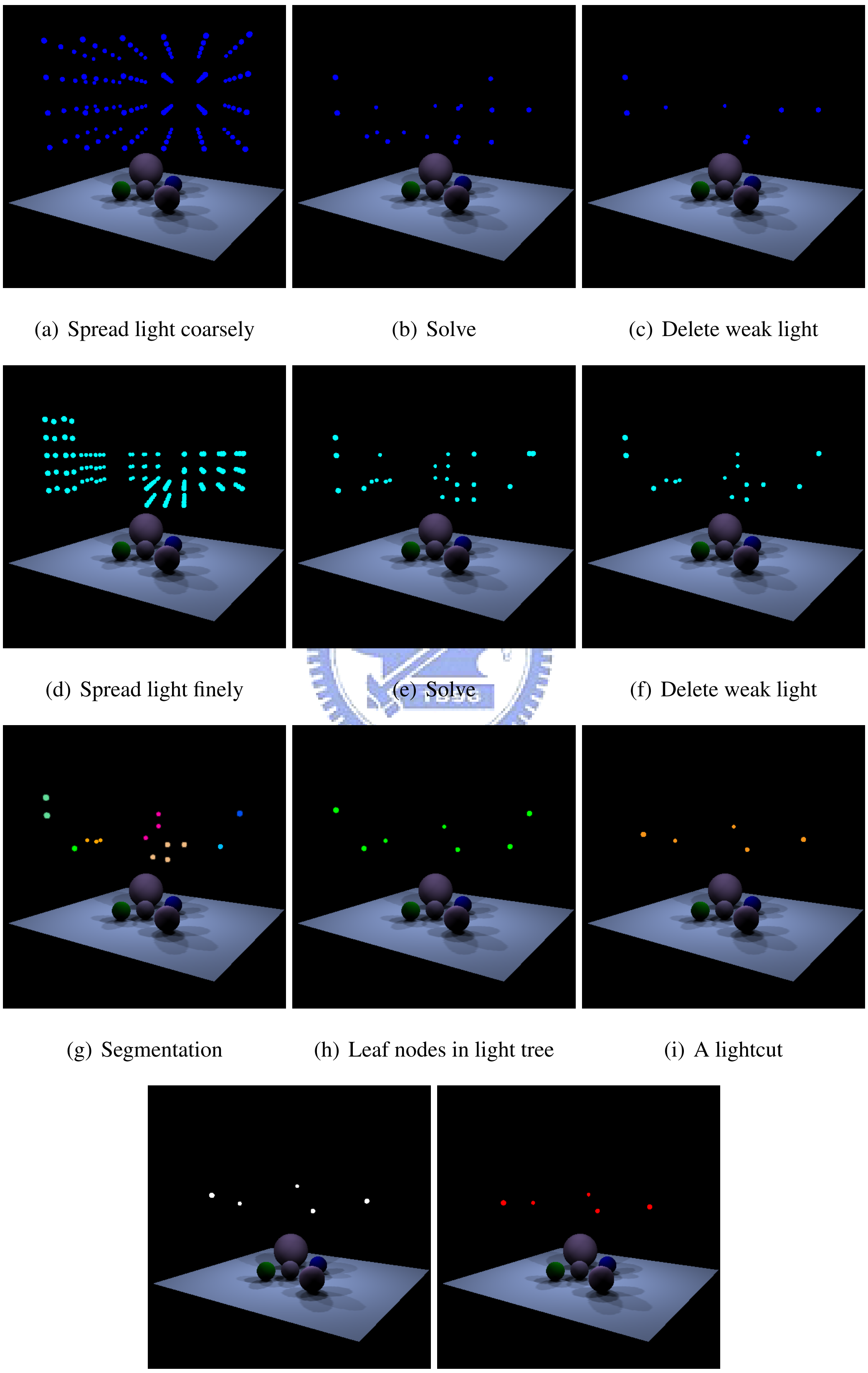
Figure 4.3(a) is ground truth in fiveballs. We directly use it as system input. Figure 4.3(b) shows the resulting illumination generated by our system. We compare ground truth with the resulting illumination shown in figure 4.3(c). We found that human eye hardly distinguishes low illumination difference. Figure 4.3(d) is 8 times of illumination difference. Figure 4.5 is another example with a more complex scene. Illumination difference is still low.

Speed bottleneck of our algorithm is computation time of least-square solver. Using fewer important vertices can reduce computation time so we choose a balance between number of vertices and quality of resulting illumination. Table 4.1 summarizes detailed statistics for fiveballs scene under 3 artificial lights with different number of important vertices. Table 4.2 shows statistics under the same environment except for 6 artificial lights. We observe that more artificial lights usually have slightly higher error because more artificial lights exit in the same space and they produce more complexity illumination. It's very hard to estimate accurate positions for each light, but we generate resulting illumination which is very close to ground truth. From observing tables, computation of least-square solver in estimating initial lighting configuration takes time much more than computation of simplex method in computing optimal lighting configuration. Therefor, choosing appropriate number of lights is very important to performance in our system. In general, the more complexity scene usually choose more vertices, which leads to better results.

important vertices	lights	guessed lights	error	total(s)	initial(s)	optimal(s)
100	3	3	8.34	1.016	1.001	0.015
500	3	3	2.62	1.547	1.515	0.032
1000	3	3	2.24	2.757	2.710	0.047
1500	3	3	2.03	3.228	3.171	0.057
2000	3	3	1.77	4.281	4.187	0.094

Table 4.1: fiveballs scene under 3 artificial lights with different important vertices

Table 4.3 shows the test case of fiveballs scene with different artificial lights. Fewer artificial



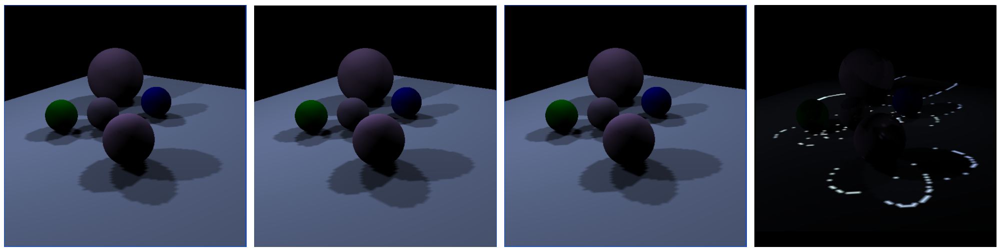
(a) Spread light coarsely (b) Solve (c) Delete weak light

(d) Spread light finely (e) Solve (f) Delete weak light

(g) Segmentation (h) Leaf nodes in light tree (i) A lightcut

(j) Simplex optimization (k) Ground truth

Figure 4.2: Illustration of initial lighting configuration and optimal lighting configuration



(a) Ground truth (b) Resulting illumination (c) Error (d) 8 x Error

Figure 4.3: Fiveballs scene with ground truth applied

important vertices	lights	guessed lights	error	total(s)	initial(s)	optimal(s)
100	6	7	8.34	3.297	3.234	0.063
500	6	7	4.49	5.578	5.141	0.437
1000	6	6	4.10	7.125	6.750	0.375
1500	6	6	4.01	10.757	10.203	0.552
2000	6	6	3.75	13.515	12.724	0.791

Table 4.2: fiveballs scene under 6 artificial lights with different important vertices

lights produce more simple illumination so we can estimate accurate number of lights. However, more artificial lights are distributed in the small space, it will produce very high complexity of illumination and lead to overestimate or underestimate number of real lights. Experiments show that overall atmosphere of lighting is consistent between ground truth and resulting illumination. Fugure 4.4 is an example in this situation.

Table 4.4 shows the test case of fiveballs scene with different lightcut. Each lightcut implies a lighting configuration. If we specify the number of guessed lights by manual setting, the results show the fact that overestimating number of lights won't increase error between ground truth and resulting illumination. Underestimating number of lights will increase error very quickly.

Figure 4.6 test our system on a real case. A user paints some shadow and shading near balls shown in figure 4.6(a). Then, this illumination can be considered as desired illumination

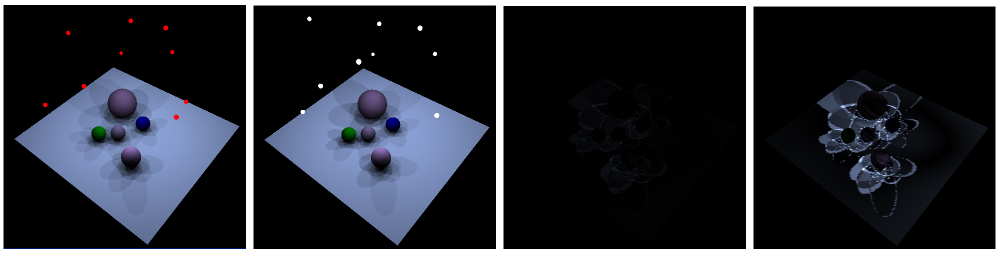
important vertices	lights	guessed lights	error	total(s)	initial(s)	optimal(s)
1000	1	1	2.01	0.594	0.578	0.016
1000	3	3	2.24	2.757	2.710	0.047
1000	5	5	3.82	6.203	6.016	0.187
1000	7	7-8	5.51	9.805	9.188	0.617
1000	9	8-11	6.87	14.093	12.812	1.281
1000	11	9-13	8.77	18.422	15.078	3.344

Table 4.3: compare the number of artificial lights with guessed lights in five balls scene

important vertices	lights	guessed lights	error
1000	3	1	13.18
1000	3	2	8.84
1000	3	3	2.24
1000	3	4	2.33
1000	3	5	2.42
1000	3	6	2.37

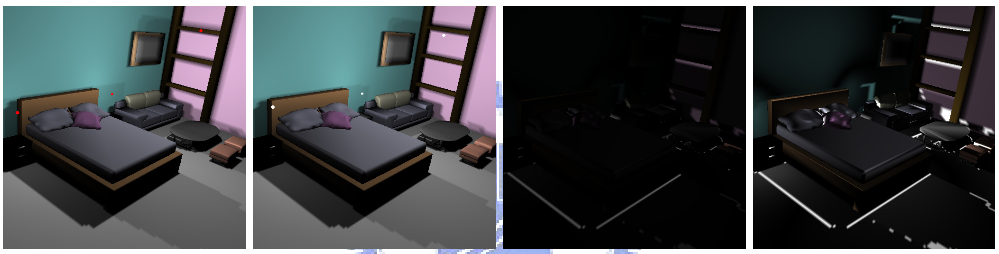
Table 4.4: fiveballs scene with different number of guessed lights

and it is solved by our system. Finally, our system generates resulting illumination shown in figure 4.6(b). Both illumination have similar overall atmosphere. Figure 4.7 is a more complex illumination scene which is painted by a user. Our system generates two lights in the scene successfully.



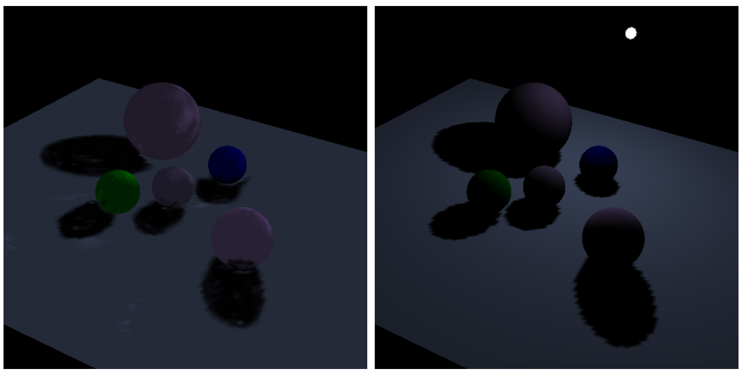
(a) Ground truth (b) Resulting illumination (c) Error (d) 8 x Error

Figure 4.4: Fiveballs scene with ground truth applied



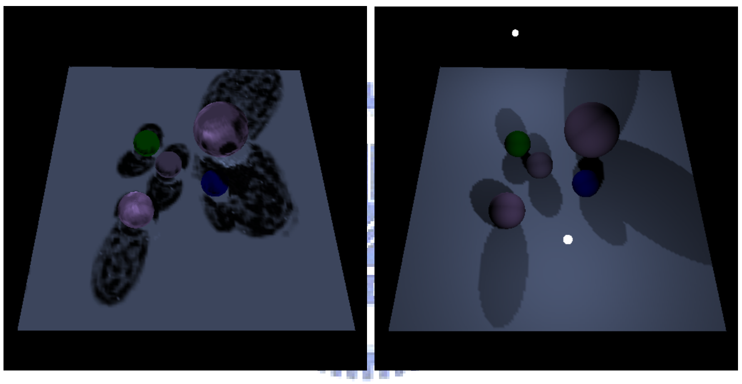
(a) Ground truth (b) Resulting illumination (c) Error (d) 8 x Error

Figure 4.5: Bedroom scene with ground truth applied



(a) painted by user (b) relighting by our system

Figure 4.6: Compare desired illumination painted by user with resulted illumination



(a) painted by user (b) relighting by our system

Figure 4.7: Compare desired illumination painted by user with resulted illumination

CHAPTER 5

Summary

In this chapter, we give brief conclusion of the thesis, and suggest some direction of the future work.

5.1 Conclusion

We propose a framework where a user can paint desired illumination on 3D object surface directly in any viewing, and the system automatically finds the best lighting parameters, including number of lights, the positions and intensities for each light, to achieve the desired lighting effect. This approach can be more intuitive for the user and easier to be used.

Our system achieves these goals by casting the problem as a high-dimensional nonlinear optimization. we propose an efficient algorithm to provide an effective initial estimation and then find optimal lighting parameters globally. Experiments show that our resulting illumination is very close to desired illumination. Our system focus on overall atmosphere but not thinking about one light at a time. Our system takes little time on optimization. This advantage avoids the user waiting for long time and reach a interactive system.

5.2 Future Work

Currently, our system only considers point light sources in diffuse environment. In order to making our system more powerful, it is necessary to support spot lights and area lights. But, these light sources will extent 4D problem to much higher dimensional problem. It will be difficult to handle them.

Because a user paints lighting effect on 3D object surface directly, it leads our rendering system only supports diffuse lighting. In the future, we can choose other rendering system, just something like as local radiance transfer [11], it will increase our quality and speed of relighting.

In our algorithm, least-square is a important role since it is the bottleneck of our consuming time. Currently, least-square is invoked from Matlab, which provides precision calculation but it's too slow for us. We don't need to high precision solution in least-square since it is just a estimating initial lighting configuration. If we adopt C-code based solver, it will reduce much more time.

In order to make the system usable for lighting designer, some way of mapping screen intensities to physical units in the system must be found. Currently, we use L_2 norm to measure difference between desired illumination and resulting illumination, but perception difference is more make sense for human eye.

Bibliography

[1] F. Anrys and P. Dutre. Image based lighting design. In *In The 4th IASTED International Conference on Visualization, Imaging, and Image Processing*, 2004.

[2] A. C. Costa, A. A. Sousa, and F. N. Ferreira. Lighting design: A goal based approach using optimization. In *In Rendering Techniques 1999*, pages 317–328, 1999.

[3] S. Fernandez, K. Bala, and D. Greenberg. Local illumination environments for direct lighting acceleration. In *Thirteenth Eurographics Workshop on Rendering 2002*, pages 7–14, 2002.

[4] S. Gumhold. Maximum entropy light source placement. In *In Proceedings of IEEE visualization 2002*, pages 275–282, 2002.

[5] M. Hašan, F. Pellacini, and K. Bala. Direct-to-indirect transfer for cinematic relighting. In *In Proceedings of ACM SIGGRAPH 2006*, pages 1089–1097, 2006.

[6] H. W. Jensen. *Realistic Image Synthesis Using Photon Mapping*. A. K. Peters, 2001.

[7] V. Jolivet, D. Plemenos, and P. Poulingeas. Inverse direct lighting with a monte carlo method and declarative modelling. In *Lecture Notes in Computer Science*, 2002.

[8] T. Jung, M. D. Gross, and E. Y.-L. Do. Light pen: Sketching light in 3d. In *Computer Aided Architectural Design Futures 2003*, 2003.

[9] J. K. Kawai and J. S. Painter. Radioptimization: goal based rendering. In *In Proceedings of ACM SIGGRAPH 1993*, pages 147–154, 1993.

[10] A. Keller. Instant radiosity. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 49–56, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co. ISBN 0-89791-896-7. doi: http://doi.acm.org/10.1145/258734.258769.

[11] A. W. Kristensen, T. Akenine-Moller, and H. W. Jensen. Precomputed local radiance transfer for real-time lighting design. *ACM Transactions on Graphics*, 24(3):1208–1215, 2005.

[12] H.-T. Kuo1, C.-Y. Hsieh, K.-Y. Lee, S.-H. Guan, and Y.-Y. Chuang. Lighting by guides. In *Computer Graphics Workshop 2008*, 2008.

[13] J. Marks, B. Andalman, P. Beardsley, W. Freeman, S. Gibson, J. Hodgins, and T. Kang. Design galleries: a general approach to setting parameters for computer graphics and animation. In *In Proceedings of ACM SIGGRAPH 1993*, pages 389–400, 1997.

[14] M. Meyer, M. Desbrun, P. Schröder, and A. H. Barr. Discrete differential-geometry operators for triangulated 2-manifolds. In H.-C. Hege and K. Polthier, editors, *Visualization and Mathematics III*, pages 35–57. Springer-Verlag, Heidelberg, 2003.

[15] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer-Journal*, 7:308–313.

[16] M. Okabe, Y. Matsushita, L. Shen, and T. Igarashi. Illumination brush: Interactive design of all-frequency lighting. In *In Proceedings of Pacific Graphics 2007*, pages 171–180, 2007.

[17] E. Paquette, P. Poulin, and G. Drettakis. A light hierarchy for fast rendering of scenes with many lights. In *Proceedings of EUROGRAPHICS98*, pages 63–74. Eurographics, 1998.

[18] F. Pellacini, P. Tole, and D. P. Greenberg. A user interface for interactive cinematic shadow design. In *In Proceedings of ACM SIGGRAPH 2002*, pages 563–566, 2002.

[19] F. Pellacini, F. Battaglia, R. K. Morley, and A. Finkelstein. Lighting with paint. *ACM Transactions on Graphics*, 26(2):9:1–9:13, Jun 2007.

[20] P. Poulin, K. Ratib, and M. Jacques. Sketching shadows and highlights to position lights. In *In Computer Graphics International 1997*, pages 56–63, 1997.

[21] W. H. Press, B. P. F. andSaul A. Teukolsky, and W. T. Vetterling. *Numerical Recipes*. Cambridge University Press, 3nd edition, 1992.

[22] C. Schoeneman, J. Dorsey, B. Smits, J. Arvo, and D. Greenberg. Painting with light. In *In Proceedings of ACM SIGGRAPH 1993*, pages 143–146, 1993.

[23] R. Shacked and D. Lischinski. Automatic lighting design using a perceptual quality metric. In *In Proceedings of Eurographics 2001*, pages 215–226, 2001.

[24] P.-P. Sloan, J. Kautz, and J. Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Transactions on Graphics*, 21(3): 527–536, Jul 2002.

[25] P.-P. Vazquez. Automatic light source placement for maximum visual information recovery. *Computer Graphics Forum*, 26(14):143–156, Jun 2007.

[26] P.-P. Vazquezz, M. Feixasz, M. Sbert, and W. Heidrich. Viewpoint selection using viewpoint entropy. In *In Proceedings of VMV 2001*, pages 273–280, 2001.

[27] L. Vincent and P. Soille. Watersheds in digital spaces: An efficient algorithm based on immersion simulations. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 13(6).

[28] I. Wald, C. Benthin, and P. Slusallek. Interactive global illumination in complex and highly occluded environments. In *Proceedings of Eurographics Symposium on Rendering*, pages 74–81, 2003.

[29] B. Walter, S. Fernandez, A. Arbree, K. Bala, M. Donikian, and D. P. Greenberg. Lightcuts: a scalable approach to illumination. In *In Proceedings of ACM SIGGRAPH 2005*, pages 1098–1107, 2005.

[30] G. Ward. Adaptive shadow testing for ray tracing. In *Eurographics Workshop on Rendering*, pages 11–20, May 1991.

[31] G. Xu, Q. Pan, and C. L. Bajaj. Discrete surface modeling using geometric flows. In *Tech. report, Dept. of Computer Sciences, University of Texas, Austin*, 2003.