Graz University of Technology
Senat

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am …………………………… ………………………………………………..
(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

…………………………… ………………………………………………..
date (signature)

Master's Thesis

Inverted Radiosity

Zmugg René

Institute of ComputerGraphics and KnowledgeVisualisation, TU Graz

www.cgv.tugraz.at

Abstract

Lighting design is nowadays more difficult than ever. The lighting conditions can make the difference between success or failure of designed spaces. Optimal lighting conditions can only be achieved when space as well as the lighting are designed together. Nevertheless this co-design is a very difficult and error-prone process. Sometimes this co-design is not even possible because the space was designed beforehand. Therefore there is a need for a tool that supports the lighting designer in his task. Illuminating 3D scenes is about placing light sources, setting their parameters, rendering the scene, and a lot of fine tuning for acquiring the optimal parameters for those light sources. We propose a new method that calculates potential light source positions with a given user defined target illumination as input.

The proposed method is independent of the scene's tessellation and assumes a geometry consistent of lambertian patches. Our method works with the predefined illumination to formulate a voting system that determines potential light source positions for a chosen light source size and brightness. Illuminated points cast votes for a set of positions where a light source with the chosen parameters would yield the observed illumination. Processing votes for a well defined set of points throughout the scene leads to a vote distribution where accumulation points of votes indicate a possible light source position. The proposed method is further capable of identifying midair light source positions. The resulting light sources produce a lighting as close as possible to the desired illumination. The deviation between desired and calculated illumination is based on the chosen light source parameters.

Keywords: inverse lighting, radiosity, global illumination, voting system

Kurzfassung

Licht-Gestaltung ist heutzutage schwieriger wie noch nie zuvor. Die Beleuchtung kann ausschlaggebend sein ob ein Raum akzeptiert wird oder nicht. Eine optimale Beleuchtung kann nur erreicht werden wenn der Raum zusammen mit der Beleuchtungskonfiguration entworfen wird. Nichtsdestotrotz ist dieses Co-Design ein sehr schwieriger und fehlerbehafteter Prozess. Manchmal ist es gar nicht mehr möglich die Beleuchtung zusammen mit den Räumlichkeiten zu entwerfen, da der Raum bereits gestaltet wurde. Folglich gibt es Bedarf für ein Programm, das dem Architekten dabei hilft nachträglich optimale Lichtbedingungen zu sorgen. Das Beleuchten von drei-dimensionalen Szenen handelt von dem Positionieren von Lichtquellen, Setzen derer Parameter, Rendern der Szene und der Feinabstimmung, die notwendig ist um die passenden Einstellungen für die Lichtquellen zu erhalten. Wir stellen eine Methode vor die potentielle Lichtquellen-Positionen, die basierend auf einer vom Benutzer definierten Zielbeleuchtung sind, berechnet.

Die vorgestellte Methode ist unabhängig von der Unterteilung der Szene und nimmt an, dass die Szene nur aus lambertschen Flächen besteht. Unsere Methode benutzt die vordefinierte Beleuchtung um eine Wahl für eine Lichtquelle mit ausgewählter Größe und Helligkeit zu formulieren. Beleuchtete Punkte stimmen für eine Menge von Positionen wo eine Lichtquelle mit den gewählten Parametern genau die beobachtete Beleuchtung erzielen würde. Die Stimmen werden von einer gut definierten Menge von Punkten in der Szene abgegeben. Diese Stimmen erzeugen eine Stimm-Verteilung in der Häufungspunkte der Stimmen Lichtquellen-Positionen andeuten. Die vorgestellte Methode ist weiters dazu in der Lage Lichtquellen, die in der Luft positioniert sind, zu berechnen. Die resultierenden Lichtquellen erzeugen eine Beleuchtung, die so nah wie möglich an der vordefinierten Beleuchtung ist. Der Fehler zwischen der resultierenden und der vordefinierten Beleuchtung ist abhängig von den gewählten Lichtquellen-Parametern.

Preface

I would like to thank all of my reviewers for their comments and help in finishing this thesis. Thanks to Sven Havemann for supervising and guiding me through writing this thesis. I am grateful to Volker Settgast for providing me with all needed information regarding modeling and illuminating in Maya. Furthermore thanks to all of my friends for tolerating my absence from our planned events. Last but not least thanks to my family for all their support.

Contents

Chapter 1

Introduction

Architecture is about creating space for human beings, and human beings require light. The lighting conditions can make the difference between a space that will be accepted, and a space that will not be accepted. Optimal lighting conditions can be achieved when architecture and illumination are designed together. However, this co-design of space and illumination is an extremely difficult, tedious, and error-prone process. To overcome these problems this project will take the reverse direction of the lighting process.

This thesis proposes a novel method for lighting design. Instead of choosing a lighting setup, the user specifies the desired outcome by painting the desired illumination on parts of the scene. Once the target illumination is specified, the goal is to reverse engineer potential positions of light sources that illuminate the scene as close as possible to the chosen illumination. Consequently the inverse algorithm takes a partially illuminated scene as input and delivers several possible light source positions as output. This class of problems is called *inverse lighting problems*.

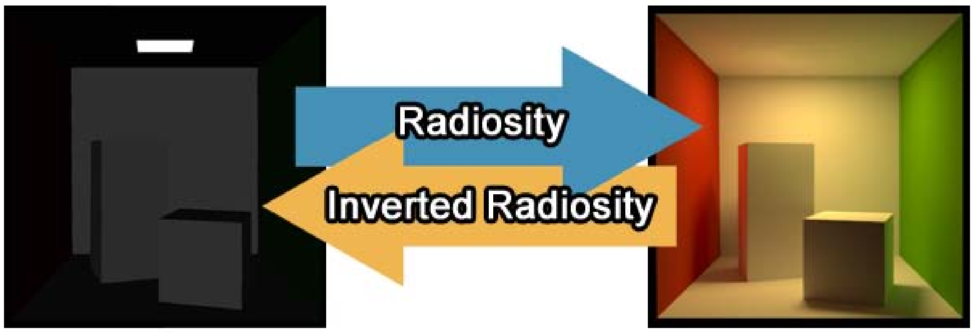


Figure 1.1: Illustration of the goal of Inverted Radiosity. On the right hand side a globally illuminated Cornell Box is shown as a result of a radiosity process. The left hand side shows the light source configuration producing the right result. Goal of Inverted Radiosity is the calculation of the light source position of the right picture with the information of the left picture as input.

Solving this kind of problem, however, is a difficult task. The nature of this problem is that it is very ambiguous. Just imagine an illuminated spot at the ground of a cubic room. A light could be positioned on the ceiling, on one of the four walls or maybe even in midair as, for example, a lamp hanging from the ceiling or a floor lamp. There is even the possibility of a set of intervening light sources which are establishing the desired illumination. Nevertheless, just the possibility of the light source being placed in midair creates countless configurations of position, size, and light strength of the to be found light sources.

Although infinite possibilities of placing light sources do exist, the probability is very low that one of those will ever reach the predefined illumination specified by the user. In virtually every case there will be a deviation between the painted illumination and the calculated one. Of course this is due to the fact that the probability of an user painting an illumination created by a light source is very low. However, the challenge is to decrease the space of potential light sources to a minimum and attain the configuration producing the lowest error. The geometry of the scene as well as the user-defined illumination can aid us in this effort.

For a human observer it is obvious which regions of the scene cannot contain a light source that will illuminate the desired spot. This is mostly due to the scene geometry. Enclosed as well as occluded environments can hardly contain a light source if the illuminated spots are not visible to them. Indirect illumination should not be forgotten here. However, the possibility of reaching the desired illumination only through indirect illumination is rather low for commonly used materials because other parts of the geometry will be illuminated directly.

Another way to decrease the amount of possible light source positions is the usage of not illuminated places. Places covered in darkness are not illuminated and are therefore not visible to the light source. However, it is still possible that the light source is far away and almost no light reaches that spot. If we assume that such constellations are not present then the amount of possible light source positions can be reduced by not considering places visible to not illuminated spots. In an incomplete user-defined illumination setting many places will not be illuminated, therefore it is necessary for the user to be able to specify regions that should be in shade.

Another problem arising with the desire to simulate lighting behavior is the computational complexity. Global illumination methods like *ray tracing* or *radiosity* require a large amount of computational effort to gain appreciable results. The situation for the reversal of such global illumination processes is no different. Despite the fast-paced development of modern CPUs computing a global illumination on a CPU in real time is still a difficult task. The processing power of modern graphics cards, however, exceeds the processing power of CPUs by far. This power was traditionally only available for the fixed function pipeline in use of pixel and vertex processing. With the invention of programmable shaders this power was partly granted for the programmer to use. Real time global illumination was coming closer, but adapting the algorithms to the limitations of shaders was a complex venture. Since the development of NVIDIA CUDA [CUD] the processing power of the graphics card became fully accessible to the programmer. With this invention real time global illumination is almost in reach. Even for the reverse approach the GPU is an inevitable way for programs with real time user interaction.

The goal of the Inverted Radiosity project is to acquire plausible configurations of light sources generating the target effect. This is done with the help of a geometrical method using local illumination features to define a set of possible light source positions. Generally speaking these local features are used to cast votes for the position of the light sources. The resulting vote distribution indicates where and how light sources need to be arranged for producing an approximation of the predefined illumination. To guarantee a low response time for the user NVIDA CUDA is used to reduce the computational time to a minimum. With the aid of an implementation on the graphics card, a change in settings that require a re-computation will not cause the user to wait several minutes or hours for the new result. The execution times are almost considered to be real time.

1.1 Structure of this Document

The following chapters of this thesis are organized as follows: The intention of chapter 2 is to review previous work already done in the field of inverse lighting problems. In chapter 3 the theoretical background of radiosity, optimizing strategies, and the concept of barycentric coordinates are reviewed. Chapter 3 also includes the concept of the designed Inverted Radiosity algorithm. First the background is reviewed. Afterwards the basic idea of the proposed algorithm is presented followed by a detailed explanation of the refined algorithm computing potential light source positions out of preilluminated scenes. The purpose of chapter 4 is to discuss the implementation and performance issues. The fourth chapter is also containing a tutorial on how to use the developed program. The fifth chapter contains a discussion about the algorithm presented in this thesis. Furthermore this chapter is used to present results. The conclusion as well as an outlook for future work is also presented in the last chapter.

Chapter 2

Related Work

In general, inverse problems are more difficult than forward problems. Forward problems calculate the output for a given input. Inverse problems, however, need to find a set of possible inputs, which produce the given output. In the field of inverse lighting problems the user defines a set of illumination effects and the task of the computer is to compute the causes, which produce the specified effects.

The solution to the forward problem is well covered in literature. However, in comparison to the forward problem, there is not that much literature available for the inverse problem. In this section we want to review the most important publications in the field of inverse lighting problems.

There are several strategies of solving the inverse problem. One approach is to take the position of the light sources as input and compute the parameters of those light sources. In 1993 Kawai et al. [KPC93] proposed a method, which takes the number and position of all light sources as input for determining the self exitance of these light sources, direction and distribution of spot lights and the reflectivity of all elements in the scene. Furthermore the user is able to specify targets which are based on human perception. With such targets an atmosphere of privacy or comfort can be established within the illuminated room. In the same year Schoeneman et al. [SDS*93] introduced an interface enabling the user to paint a desired illumination on all lambertian surfaces in the scene. Given the desired illumination, their method then computed the color as well as the intensities of a fixed number of light sources with given position. The drawback of all these methods is that the light sources positions need to be given. In contrary, our method computes the positions, but takes the light source parameters as input.

These proposals are the first attempts in solving this inverse problem, but are very restrictive. The following proposals are not taking the light source positions as input.These proposals show that it is no prerequisite that the positions of the light source need to be given. Two different basic approaches are used in these methods. On the one hand features of the geometry as well as features of the given illumination are used to solve the inverse problem. On the other hand the user-defined illumination is used to formulate a optimization problem based on different lighting equations. This review first focuses on the geometry based approaches.

Poulin et al. proposed two papers in this field. The first proposal [PF92] determined the position of extended light sources as well as the direction of parallel light sources by using highlight or shadow information of the scene. The main problem of this approach is that multiple reflections of light are ignored. Furthermore the results rely on view-dependent highlights. Their second paper [PRJ97] was published four years later. It is again geometrical approach for calculating the position of a point light source given either the sketch of a highlight or umbra. With the help of umbra and penumbra sketches even predefined polygonal light sources can be positioned. The above mentioned problems caused by view-dependent highlights and ignoring multiple reflections remain. Our method as well is a geometrical approach focusing on directly illuminated parts of the scene, but our approach relies not on view-dependent highlights.

The following approaches use an optimization problem to determine possible light source positions. These optimizations problems are either solved with stochastic methods or with the help of deterministic methods.

Tena et al. [TG97] picked up the idea of the previously mentioned Schoeneman et al. [SDS*93] paper. Based on Schoeneman's solution, Tena et al. implemented a new approach using genetic algorithms. The genetic algorithm is used to test several lighting configurations. For each configuration an error value to the desired lighting is computed. This error value is used to determine the next configuration. Their system automatically fixes the number, positions as well as the intensities of light sources. However, their system depends on the scene's tessellation and a lot of time is necessary for their genetic algorithm to find a configuration with a low error value.

In 1999 Costa et al. [ISC*99] published an approach capable of computing position, direction and angle of light sources. Their method is based on general reflectance functions and uses a stochastic method called simulated annealing for optimizing their parameters. Simulated annealing simulates heating and controlled cooling of a material such as steel and is a very common way for optimization. The execution time as well as convergence depend on the evolution of the temperature variable. Convergence to the global optimum is not assured as in all stochastic approaches. In worst cases there is no convergence or the execution time is very high. Their method takes the geometry, material properties and an illumination goal as input. Such a goal specifies the number and type of light sources and the desired illumination data. Relationships and constraints between parameters and the lighting data can also be included in the illumination goal. As mentioned before the major drawbacks of this method are the risk of non-convergence and the high computational time. The advantage of this method, as well as of our method, is that light sources placed in midair can be resolved.

In 2002 a paper solving the inverse problem in combination with radiosity was published by Contensin M. et al. [M.02]. Their method computes a lighting configuration that generates an illumination as near as possible to the illumination specified by the user. The user can define two types of constraints to an arbitrary subset of all patches in the scene: radiosities and forbiddings. Radiosities represent the illumination of a patch that the resulting lighting configuration has to generate. All patches in the scene are possible light sources except those provided with radiosities or those which are forbidden to receive any light. With the user-defined properties of the scene, the radiosity equation is reformed to a set of equations with a number of known parameters. The amount of known parameters is based on the amount of patches provided with constraints. These equations are solved by means of a pseudo-inverse resulting in a least squares solution. The resulting solution does satisfy all target illuminations, however, it does have too many light sources and even physically impossible patches that absorb light. These patches are represented by negative exitance values. This solution is used as a starting point of an iterative process which minimizes the number of light sources and eliminates all negative values such that all patches have positive self exitances. One major drawback of this method is that the solution depends on the tessellation of the scene as radiosity itself depends on the scene's tessellation. Furthermore light sources placed in midair cannot be achieved.

We propose a geometrical method for solving a given inverse lighting problem in which even light sources in midair can be resolved. In our method, as well as in the previously mentioned methods, the light sources do not need to be fixed. Moreover, our method is independent of the tessellation of the geometry. The resulting light sources are not represented as a patch in the given geometry. Or method returns a set of three-dimensional coordinates instead. These coordinates are the light source positions for light sources of the previously chosen size and brightness. The developed tool takes the intensities, which are stored in a texture, as illumination and is therefore providing the user with more freedom in generating own scenes with different illuminations.

Another related research in the field of inverse lighting problems was done in 1999 in the paper *"Inverse global illumination: recovering reflectance models of real scenes from photographs"* [YDMH99]. Their method retrieves material properties out of photographs of a real scene. Combining their results with ours is a very interesting research topic and will be discussed in chapter 6.

Chapter 3

Theory

Contents

The purpose of the designed algorithm is to retrieve possible light source positions which match a user-defined illumination in a given scene as well as possible. As mentioned in the previous chapters the goal is to calculate causes producing the outcome defined by a user. Therefore an inverse lighting problem has to be dealt with.

This chapter provides all necessary background knowledge as well as the proposed method in theory. First the theoretical foundations of this thesis are reviewed in the following section. Afterwards the theory of the Inverted Radiosity algorithm is explained in detail.

3.1 Theoretical Foundations

This section provides all necessary information for understanding this thesis. In the first section the basic radiosity method is reviewed. Afterwards the basics of optimizing functions are coped with. This optimization based section deals with deterministic as well as non-deterministic methods. In the last section the concepts of barycentric coordinates are reviewed.

3.1.1 Radiosity

The method for retrieving potential light sources is based on the radiosity method. Radiosity is a global illumination method which considers physical laws in illuminating a scene consisting out of diffuse surfaces. In contrary to other global illumination methods illuminated parts of the scene also act as light emitters. Light is reflected diffusely at all surfaces. Consequently parts usually not visible to the light source can be illuminated as well. This is a short summary of theory, please refer to the book from Glassner [Gla94, volume II, chapter 18] for the details.

For calculating the radiosity of a scene the geometry is divided into patches. For the classical radiosiy method the patches of the scene are assumed to be small. The classical radiosity equation for a scene divided into n patches is formulated as follows:

$$B_i = E_i + \rho_i \cdot \sum_{k=1}^{n} B_k \cdot F_{i,k} \tag{3.1}$$

whereas B_i denotes the radiosity of a patch i, which consists of self exitance E_i plus energy received from all other visible patches in the scene. This received radiance can be expressed by a sum over all patches k. The *form factors* $F_{i,k}$ represent the fraction of the amount of energy, which is transmitted by a patch i and received by another patch k. Furthermore this sum is weighted by the reflectance factor ρ_i of patch i. Form factors are defined in dependence to the orientation of the patches to each other.

In general form factors are defined as a percental light transport between two patches with finite size. $F_{i,k}$ is therefore defined as follows:

$$F_{i,k} = \frac{1}{A_i} \int_{A_i} \int_{A_k} \frac{\cos(\Theta_i) \cdot \cos(\Theta_k)}{\pi \cdot r^2} \cdot V(i,k) \cdot dA_k \cdot dA_i \tag{3.2}$$

The angles Θ_i and Θ_k are defined as the angle between the normal of the respective patch and the connection of both patches. A_k as well as A_i represent the area of the respective patches and r represents the distance between the related patches. The inner integral of equation 3.2 integrates the form factor over the area of patch k with respect to the visibility $V(i,k)$. $V(i,k)$ is simply defined as 1 when the respective patches are visible to each other and 0 if they are not. The outer integral integrates over the area of the light emitting patch i. A constant radiance over the surface of each patch is assumed, therefore the integral is weighted by the factor $\frac{1}{A_i}$.

An important property of form factors is that all energy of a patch is distributed among all other patches, consequently following condition is fulfilled for a scene containing n patches:

$$\forall i : \sum_{k=1}^{n} F_{i,k} = 1 \tag{3.3}$$

In case of infinitely small patches the form factor formula can be put into a more comprehensible form:

$$F_{i,k} = \frac{\cos(\Theta_i) \cdot \cos(\Theta_k)}{\pi \cdot r^2} \cdot A_k \tag{3.4}$$

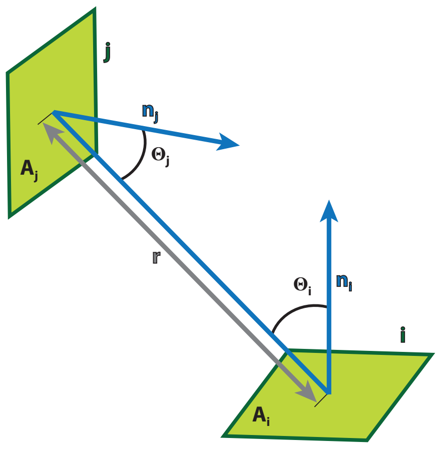


Figure 3.1: Illustration showing all terms necessary in calculation of form factors between two chosen patches i and j. A_i and A_j denote the areas of the respective patches; n_i and n_j represent the respective surface normal. The length of the connection between the centroids of both patches is labeled r and the angles Θ_i and Θ_k stand for the angles between this connection and the according normal vectors.

In addition to the formulas in 3.2 and 3.4 figure 3.1 illustrates all necessary terms used in the calculation of the form factors between the chosen patches i and j. For all further calculations we assume the use of the form factor equation in 3.4.

When the associated two patches of a form factor $F_{i,k}$ do not face to each other, meaning they are not visible to each other, the form factor is defined as 0. This is due to the fact that patches that do not face to each other or are occluded cannot illuminate each other. Such a setting leads to either $\cos(\Theta_i)$ or $\cos(\Theta_k)$ being less than 0. Because negative form factors cannot exist the associated form factor is set to 0. Likewise the form factor $F_{i,k}$ is defined as 0 for $i = k$ because all patches are assumed to be planar and therefore cannot illuminate themselves.

Both cosine terms can also be interpreted as an inner product of two vectors. Let $\mathbf{d}$ be the normalized direction from patch i to patch j and therefore $-\mathbf{d}$ the normalized direction from patch j to i. Furthermore let $\mathbf{n_i}$ and $\mathbf{n_j}$ be normalized. Then the form factor equation can be rewritten as:

$$F_{i,k} = \frac{\langle \mathbf{n_i}, \mathbf{d} \rangle \cdot \langle \mathbf{n_j}, -\mathbf{d} \rangle}{\pi \cdot r^2} \cdot A_k \tag{3.5}$$

As seen in the rewritten form factor equation for fixed size patches (see 3.5), the form factor reaches a maximum, when both patches exactly face each other (in this case $\mathbf{n_i} = \mathbf{d}$ and $\mathbf{n_j} = -\mathbf{d}$). Moreover if $\mathbf{n_i} = -\mathbf{n_j}$ then $\mathbf{n_i} = \mathbf{d}$ as well as $\mathbf{n_j} = -\mathbf{d}$. With these identities both cosine terms become 1 and therefore are maximized. Obviously this setting maximizes the numerator of the fraction in the form factor equation because both cosine terms become 1. For fixed size and orientation of the patches, the only remaining parameter is the distance r. The form factor increases when decreasing the distance r.

The form factor equation also leads to another relationship between two form factors:

$$F_{i,k}A_i = F_{k,i}A_k \tag{3.6}$$

After calculating all form factors of all pairs of patches the light transfer can be computed. There are many approaches for calculating the radiosity of all patches of a scene. The original approach initially sets $B_i = E_i$ for all patches in the scene. Then the patch with the highest radiosity value, which is at the current iteration step the brightest light emitter in the scene, is selected and all its energy is distributed in accordance to its form factors. This process is repeated so that patches are chosen depending on their current radiosity value, then all other patches are illuminated based on the amount received from the current brightest patch. The basic radiosity method approximates the actual light distribution in a scene after all patches are processed once. The smaller the patches the smaller the error to a real world lighting example. For more information on other radiosity methods see [Gla94, volume II, chapter 18].

The previous discussion of radiosity is sufficient for understanding the content of this thesis. The form factors as reviewed in this section are an essential part in the proposed method. Due to the representation of the fraction of energy leaving one path and arriving at another patch, it is possible to calculate the amount of energy that can be transmitted between two patches. The form factors described in 3.4 are sufficient for achieving the desired results because only point to point form factor calculations are made in the remains of this thesis.

3.1.2 Optimizing Strategies

Throughout the process of finding potential light source positions certain functions need to be minimized or maximized. A function $g(\mathbf{x})$ to be maximized can always be transformed into a function $h(\mathbf{x})$ to be minimized[1]. Therefore only minimization problems will be discussed in this section.

To achieve good approximations for a set of input variables that minimize a certain function, optimizing strategies need to be applied. In this thesis two different approaches are tested and used for calculating a minimizing set of variables. On the one hand the deterministic *Newton's method* is used, however, in certain cases another non-deterministic method called *(1+1) evolution strategy* yields better results. Both methods with their advantages and disadvantages are reviewed in the following sub-sections of this chapter.

3.1.2.1 Deterministic Methods

In this section the main focus lies on Newton's method [Fle87] [GMW81], which uses the second derivative of the function to be minimized. Newton's method is designed to yield faster convergence times as the step size of the gradient is adjusted based on an approximation of the function. Let $\mathbf{x}$ be an n-dimensional parameter vector and $f(\mathbf{x})$ be the objective function to be minimized. Newton's method is based on a quadratic model of the objective function. Due to this interpretation a quadratic model of $f(\mathbf{x})$ is created by using the second order Taylor expansion of $f(\mathbf{x})$:

$$f(\mathbf{x}+\mathbf{p}) \approx f(\mathbf{x}) + \langle g(\mathbf{x}), \mathbf{p} \rangle + \frac{1}{2} \langle \mathbf{p}, H(x)\mathbf{p} \rangle \tag{3.7}$$

whereas $g(\mathbf{x})$ symbolizes the gradient $\nabla f(\mathbf{x})$ and $H(\mathbf{x})$ symbolizes the Hessian $\nabla^2 f(\mathbf{x})$. The quadratic function 3.7 is formulated in terms of another n-dimensional vector $\mathbf{p}$. This vector $\mathbf{p}$ symbolizes the step required to reach the minimum of the quadratic function. It is known that the formula in 3.7 attains its minimum if $\mathbf{p}$ solves following linear equation:

$$H(\mathbf{x})\mathbf{p} = -g(\mathbf{x}) \tag{3.8}$$

The vector $\mathbf{p}$ (as the solution of 3.8) is called *Newton direction*.

If the formulation of the objective function in 3.7 is accurate[2] and the Hessian $H(\mathbf{x})$ is positive definite one iteration step is required to attain the minimum of $f(\mathbf{x})$. However, the basic Newton's method is not suitable for most real world

[1] $h(\mathbf{x}) = -g(\mathbf{x})$

[2] meaning that $f(\mathbf{x})$ is a quadratic function

examples due to the problem that $H(\mathbf{x})$ may not be positive definite when the starting point $\mathbf{x}$ is chosen badly. Even if $H(\mathbf{x})$ is positive definite the formulation in 3.7 may not be accurate enough to reach convergence in the minimization process.

The latter problem can be solved by using the Newton direction $\mathbf{p}$ as a search direction for a line search algorithm. Such a line search algorithm searches a minimum along the direction $\mathbf{p}$, meaning it minimizes $f(\mathbf{x}+\alpha\cdot\mathbf{p})$ in respect to α (assuming that $\mathbf{p}$ is a unit direction). Therefore a new iteration point $\mathbf{y}=\mathbf{x}+\alpha\cdot\mathbf{p}$ can be found for Newton's Method.

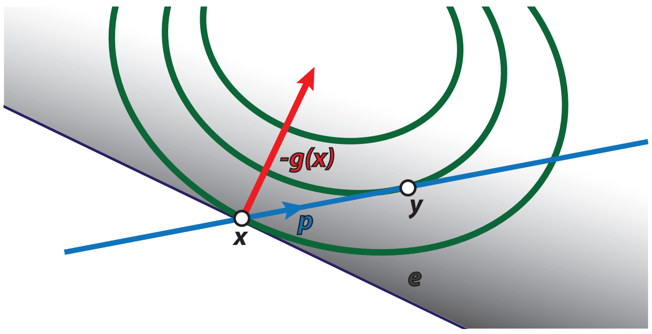


Figure 3.2: Illustration of one iteration of Newton's method with line search. The direction $\mathbf{p}$ is the Newton direction calculated in the basic Newton's method. The line search algorithm minimizes $f(\mathbf{x}+\alpha\cdot\mathbf{p})$ in respect to α. The point $\mathbf{y}$ for the next iteration of Newton's method is found by $\mathbf{x}+\alpha\cdot\mathbf{p}$. Furthermore all vectors leading from $\mathbf{x}$ into the half space e are directions fulfilling the descent property (see 3.9).

Figure 3.2 illustrates one iteration of Newton's method with line search. As the gradient $g(\mathbf{x})$ is the outwards facing normal vector to the equipotential lines of $f(\mathbf{x})$, $-g(\mathbf{x})$ is always the vector towards the steepest descent. For any search direction $\mathbf{s}$ leading from $\mathbf{x}$ into the half space e (as defined in figure 3.2) $f(\mathbf{x}+\alpha\cdot\mathbf{s})$ decreases the objective function for a specific α. Therefore all vectors with an angle less than $\frac{\pi}{2}$ between itself and $-g(\mathbf{x})$ are directions which can decrease the objective function in a certain interval. Consequently a descent property can be attained by using $g(\mathbf{x})$:

$$\langle g(\mathbf{x}),\mathbf{p}\rangle < 0 \tag{3.9}$$

The descent property in 3.9 is fulfilled for the Newton direction $\mathbf{p}$ if and only if $H(\mathbf{x})$ is positive definite.

Still the Hessian $H(\mathbf{x})$ needs to be positive definite. When the problem arises that $H(\mathbf{x})$ is not positive definite one possible approach to overcome this problem is the *Levenberg-Marquardt method*. This method modifies the Hessian by giving it a bias towards $-g(\mathbf{x})$[3]. This is achieved by simply adding a multiply of the unit matrix I to the Hessian before solving the system in 3.8. The modified system looks like this:

$$[H(\mathbf{x})+\lambda I]\mathbf{p}=-g(\mathbf{x}) \tag{3.10}$$

The variable λ in 3.10 is chosen in a way that $[H(\mathbf{x})+\lambda I]$ becomes positive definite. For matrices $H(\mathbf{x})$ being close to being positive definite a small value of λ may be enough to give a good search direction $\mathbf{p}$. For more detailed information on the Levenberg-Marquardt method see [Lev44] and [Mar63].

3.1.2.2 Stochastic Methods

The main focus in this section lies on stochastic optimization techniques not requiring any derivates such as evolution strategies. Stochastic methods are often used when there is little knowledge about the objective function or the existence of local minima. These methods, however, can also be used for getting asymptotic solutions very quickly. Because of the non-deterministic behavior of stochastic methods it is not guaranteed that the exact parameters of the minimum are reached. For reaching the exact parameters of the minimum a high number of iterations is required, however, often asymptotic solutions are enough.

The goal of evolution strategies is to simulate the natural evolution process. The development of such methods was started by Rechenberg in 1973 [Rec73], [Rec94]. For such a simulation different generations of parameter configurations are necessary. Each of these configurations also carries a quality value. Let $\mathbf{x_i}$ be an n-dimensional parameter configuration and let $f(\mathbf{x})$ be the objective function to minimize. Then the quality value of $\mathbf{x_i}$ is computed by evaluating the objective function at $\mathbf{x_i}$. The purpose of this quality value is stating how good this configuration is. In case of a function to be minimized a low quality value indicates a better configuration then a bigger one.

[3]the vector of steepest descent

Those generations are based on a start configuration $\mathbf{x_0}$ and evolve over time. Therefore several natural processes are simulated. The most common simulated processes are: *mutation*, *selection*, *competition*, and *reproduction*. Mutation is realized in such a way that each newly derived configuration is mutated by adding a Gaussian distributed value vector having the same dimension as the parameter configuration. Selection, however, is accomplished by only choosing the best configurations of the last generation to be participating in production of the next generation. The process competition gives even bad configuration a chance to be part of the next generation. To avoid converging into a local minimum it is sometimes necessary to let bad configuration participate in the evolutionary process. Competition is realized in such a way that each configuration gets a probability to participate. The better the quality the higher the possibility. The last mentioned process is reproduction. Reproduction is used in higher order evolution strategies. Through combining the parameters of two or more configurations new ones are derived.

The stochastic method used in this project is a variation of the (1+1) evolution strategy and this evolution strategy will be explained in the following paragraphs. In the (1+1) evolution strategy the stochastic processes mutation and selection are used. The "(1+1)" means that for each generation there is always one parent from which one descendant is derived.

In the original evolution strategy method a random start configuration is chosen within the feasible region. Mutating this initial configuration $\mathbf{x_{parent}}$ is defined as follows:

$$\mathbf{x_{descendant}} = \mathbf{x_{parent}} + \sigma \cdot \mathbf{v} \tag{3.11}$$

whereas $\mathbf{v}$ is a vector of the same dimension as $\mathbf{x_{parent}}$ containing Gaussian distributed values with mean 0 and a standard deviation of 1. The value σ, called step size, is multiplied with the vector $\mathbf{v}$ ensuring that smaller changes (mutations) - in respect to σ - happen more often than big changes. Afterwards the quality values of both configurations are compared (selection). The configuration yielding the better value of the objective function is used as the parent for the next generation.

This process is repeated until a certain stopping criterion is met. Possible stopping criteria are for example quality values less than a predefined value or maximum number of function evaluations or iterations has been exceeded. In case of a (1+1) evolution strategy the parent configuration of the last iteration is chosen to be the solution of the minimization process. However, if an evolution strategy is chosen where deteriorations are allowed, the solution does not need to be the configuration of the last iteration. The overall best configuration must be saved during the iteration process to become the solution.

Nevertheless the step size σ needs to be adapted in respect to the progress of the optimization. This adaption is required to prevent convergence into a local minimum, if such exist. The classical adaption method from the initial implementation of evolution strategies [Rec73] state that σ depends on the ratio of positive mutations[4] $p(pos)$.

$$p(pos) = \frac{\text{\# of positive mutations}}{\text{\# of all mutations}} \tag{3.12}$$

If $p(pos)$, as defined in 3.12, is lower than $\frac{1}{5}$ than the step size σ is decreased by multiplying it with a factor 0.85. Otherwise the step size is increased by dividing it by the same factor 0.85. The chosen values $\frac{1}{5}$ and 0.85 are based on the initial implementation of Rechenberg [Rec73] and can be changed due to personal needs. This adaption of the step size is applied after a predefined set of iterations dependent on the objective function. A flow chart of the whole (1+1) evolution strategy is illustrated in figure 3.3.

One drawback of this method is that no deterioration of the objective function is allowed. In case of the existence of local minima (1+1) evolution strategies cannot guarantee that the global minimum is found. There also exist higher order evolution strategies, like $(\mu|\rho,\lambda)$ evolution strategies for example, that implement such a feature. For more information on higher order evolution strategies see [Rec94].

3.1.3 Barycentric Coordinates

Barycentric coordinates are used repeatedly in this project. Their purpose is to describe all points within a *simplex* as a linear combination of the simplex' vertices. A simplex is the generalization of a triangle in arbitrary dimensions. The number of vertices required to generate a simplex depends on the used simplex' dimension. For a basis of an n-dimensional space n basis vectors are necessary. A single basis vector can be defined by two points. Consequently for defining n basis vectors $n+1$ points are required; an origin and one point per vector. Therefore in general a n-simplex is an n-dimensional polyhedron consisting of $n+1$ vertices. All these vertices lie on the convex hull of this polyhedron, so it is the smallest convex set out of $n+1$ vertices in n dimensions. Consequently a 2-simplex is a triangle, a 3-simplex a tetrahedron and so on. As the geometry of scene is based on triangles this section focuses on the barycentric coordinates of triangles.

[4] mutation that yielded a better quality of the objective function

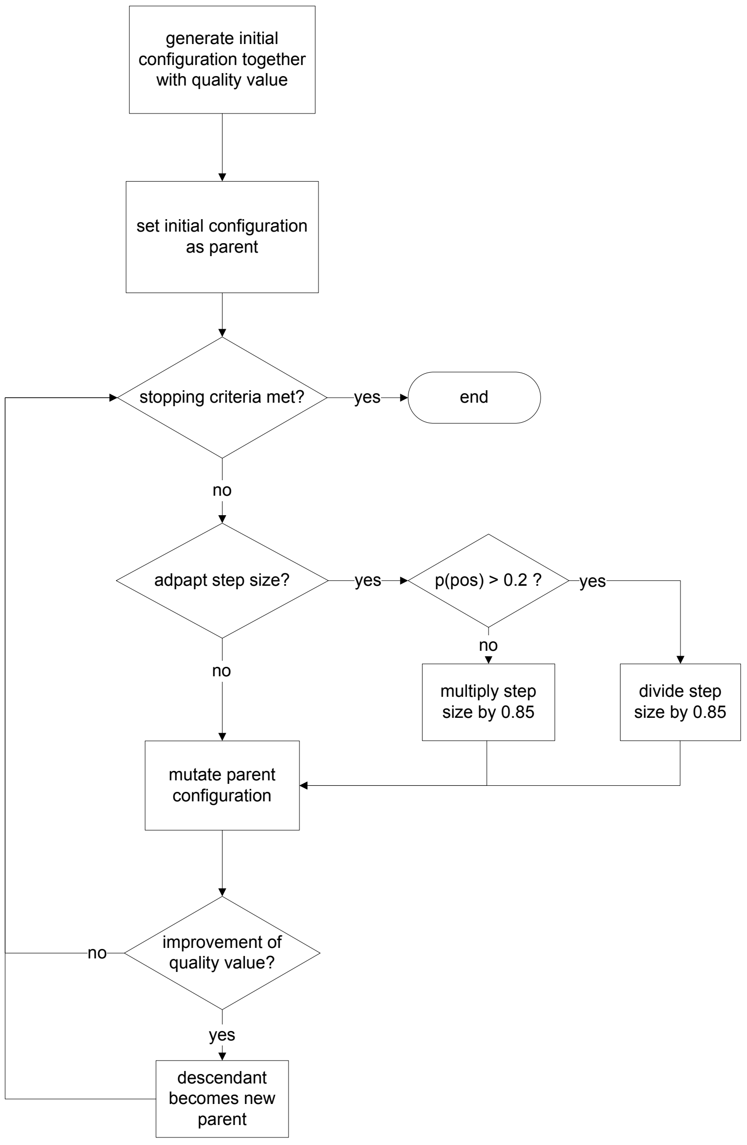

generate initial configuration together with quality value
set initial configuration as parent
stopping criteria met?
yes
end
no
adpapt step size?
yes
p(pos) > 0.2 ?
yes
no
multiply step size by 0.85
divide step size by 0.85
no
mutate parent configuration
no
improvement of quality value?
yes
descendant becomes new parent

Figure 3.3: Flow chart of a (1+1) evolution strategy. Possible stopping criteria are for example quality values less than a predefined value or maximum number of function evaluations or iterations has been exceeded. The chosen values 0.2 and 0.85 are based on the initial implementation of Rechenberg [Rec73] and can be changed according to personal needs. The step size adaption is applied after a predefined set of iterations dependent on the objective function.

Let T be a triangle with its vertices **A**, **B** and **C**. Each point **p** inside this triangle can be expressed as:

$$\mathbf{p} = \lambda_1 \cdot \mathbf{A} + \lambda_2 \cdot \mathbf{B} + \lambda_3 \cdot \mathbf{C} \tag{3.13}$$

with the condition

$$\sum_{i=1}^{3} \lambda_i = 1 \tag{3.14}$$

For a triangle in two dimensions, as for example in texture space, the barycentric coordinates of a point can be calculated as follows. The coordinates of $\mathbf{p} = \binom{x}{y}$ can be expressed in terms of the vertices of the triangle T.

$$x = \lambda_1 \cdot x_A + \lambda_2 \cdot x_B + \lambda_3 \cdot x_C \tag{3.15a}$$
$$y = \lambda_1 \cdot y_A + \lambda_2 \cdot y_B + \lambda_3 \cdot y_C \tag{3.15b}$$

As all three λ_i need to sum up to 1, the third can be expressed in the terms of the others.[5] Therefore λ_3 can be substituted:

$$x = \lambda_1 \cdot x_A + \lambda_2 \cdot x_B + (1 - \lambda_1 - \lambda_2) \cdot x_C \tag{3.16a}$$
$$y = \lambda_1 \cdot y_A + \lambda_2 \cdot y_B + (1 - \lambda_1 - \lambda_2) \cdot y_C \tag{3.16b}$$

What is left in equation 3.16 is a set of two equations in two variables, which can be solved easily. For the sake of completeness the solutions for λ_1, λ_2 and λ_3 are listed here.

$$\lambda_1 = \frac{(y_B - y_C) \cdot (x - x_C) + (x_C - x_B) \cdot (y - y_C)}{(x_A - x_C) \cdot (y_B - y_C) - (y_A - y_C) \cdot (x_B - x_C)} \tag{3.17a}$$
$$\lambda_2 = \frac{(y_C - y_A) \cdot (x - x_C) + (x_A - x_C) \cdot (y - y_C)}{(x_A - x_C) \cdot (y_B - y_C) - (y_A - y_C) \cdot (x_B - x_C)} \tag{3.17b}$$
$$\lambda_3 = 1 - \lambda_1 - \lambda_2 \tag{3.17c}$$

With the help of barycentric coordinates it is easy to determine whether a point lies within a triangle or not. Following conditions must be met for the barycentric coordinates of a point **p** if **p** lies inside a given triangle:

$$0 \leq \lambda_i \leq 1 \quad \text{for} \quad i \in \{1,2,3\} \tag{3.18}$$

Otherwise the point lies outside of the triangle.

For a triangle in the three-dimensional space it is necessary that the point, for which the barycentric coordinates are calculated, lies in the plane of the triangle. Another possibility is to use the normal vector to compute a fourth point and calculate the barycentric coordinates of a tetrahedron. The multiplier of the fourth point has to be zero and the others must fulfill the conditions of equation 3.18 for a point **p** to lie inside the basis triangle. This concludes this short review on barycentric coordinates.

3.2 A Theory for Inverted Radiosity

This section offers a gradual explanation of the development of this thesis' theory. For the purpose of this research a scene with fixed geometry consisting out of lambertian patches is assumed as input. Furthermore, it is assumed that flat circular light sources are illuminating the scene. The light strength is represented as the brightness value in a HSV representation of the light source color. The light source color is always assumed to be white. The unit of the light source size is square meter.

In the first sub-section the basic idea of the used voting scheme as well as its shortcuts are discussed. This introductory section is followed by a section discussing important observations on the form factor equation leading to an improved voting system. Furthermore the form factor distributions resulting from different scene setups are discussed. The third section 3.2.3 is offering a formula of a special ellipse describing the contour lines of the previously discussed form factor distributions. With the information of section 3.2.3 it is possible to describe a set of possible light sources which achieve a desired illumination at a fixed spot. The next section's purpose is to obtain the parameters of such an ellipse. This means for a given setting an ellipse is fitted to represent the set of possible light source positions. The last chapter summarizes the theoretical concepts and leads to an algorithmical formulation.

[5] $\lambda_3 = 1 - \lambda_1 - \lambda_2$

3.2.1 The Basic Idea

The core of our inverse algorithm for solving the presented inverse illumination problem is a patch based voting system. The underlying idea is the fact that each illuminated patch has to be visible from the patch that is illuminating it. Therefore, the patch i, which is illuminating a specific patch k, whether by direct or secondary illumination, has to be visible to patch k. Consequently each patch can cast votes to all patches visible to it because each visible patch could be the light source illuminating it. The information of dark patches can be used as well. Due to the fact that no light source can be visible to a patch, that is not receiving any light, no patch visible to this patch can be a light source.

One problem arising with this method is that it depends severely on the scene's tessellation. This, however, is rooted in the fact that radiosity itself is patch size dependent, but an increase in the scene complexity is also increasing the computational effort. Furthermore the resulting vote distributions might not be reliable enough. Voting for all visible patches is far too complex because only a small set of these patches will really be able to achieve the desired illumination on the observed patch.

One goal of this research is that the resulting algorithm should be able to obtain light sources that are placed in midair. A patch based approach only voting to patches existing in the scene will not be able to achieve this.

Consequently more research in the underlying principles is necessary. The following section will discuss important observations on the form factor equation.

3.2.2 Observations

The main problem of the basic voting system is that only voting for visible patches is not accurate enough. It is important to notice that, for a given brightness of the light source, not all patches visible to an illuminated spot can be a light source producing the desired illumination on that spot. There are far less patches able to do that. Furthermore if only patches are taken into account light sources in midair cannot be obtained. To obtain such light sources changes to the voting system have to be made. And furthermore the one aim of this work is to be independent of the tessellation of the scene. Therefore, instead of a set of patches a set of three-dimensional coordinates have to be solution to the given inverse problem. These coordinates resemble light source positions for a given size and brightness. This section discusses the observations that led to the improved voting system that has been implemented.

There are important observations that can be made by analyzing the form factor equation in 3.4:

- form factor increases by decreasing the angle between the normal of the emitting patch and the connection of the patches
- form factor increases by decreasing the angle between the normal of the receiving patch and the connection of the patches
- luminosity is decreasing with increasing distance

With the help of these observations it is possible to calculate the distance between a light source with known size and brightness and a point with given illumination. For determining a set of possible light source positions for one illuminated point two planes must be given. On one plane the point with the given illumination is located and the light source is to be located somewhere on the other plane. To pinpoint a set of possible light source positions the size and light strength of the light source must be given. As a circular light source is assumed the light source size describes the area of the circle representing the light source. The distance of the light source depends on the constellation of the two planes and on the settings of the desired light source[6].

The form factor is calculated in means of the equation in 3.19, which is an adaption of the formula in 3.4. The adapted form factor equation is the following:

$$F_{i,k} = \frac{\cos(\Theta_i) \cdot \cos(\Theta_k)}{\pi \cdot r^2} \cdot A \tag{3.19}$$

where A denotes the product of light source size and light source strength. This form factor describes the amount of light transferred from a light source with given parameters to a fixed point on a surface. The assumption made here is that the whole light source is visible to the illuminated point. Otherwise an integral over the area of the light source would be necessary in respect to the visibility. This may sound like a strong assumption, but the proposed method relies more on direct illuminated points than on partly or indirect illuminated points. Furthermore this method focuses on finding a solution for a very ambiguous problem therefore cut backs and simplifications are necessary. Therefore this assumption is justifiable as it also decreases the computational effort.

[6] light source size and light strength (brightness)

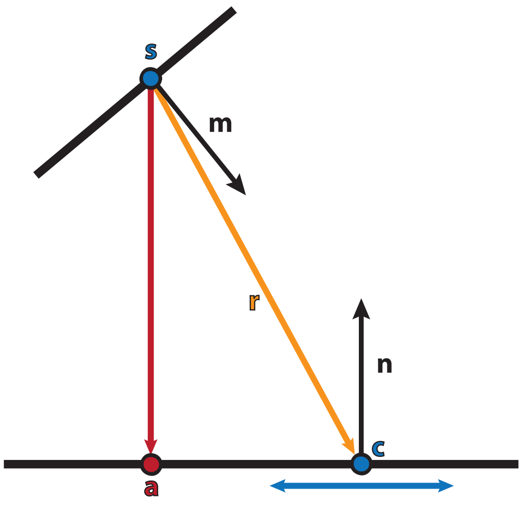


Figure 3.4: A common setting in the observation of form factors. This is a setting of two planes with the respective normals **n** and **m**. The point **s** is fixed and used for observing the form factors produced by the moving light source **c**. Furthermore **s** has a given distance to the light source containing surface along the normal **n**. The length of the vector **r** that connects the point **s** and the position of the light source **c** is used in the form factor equation (see 3.19). The vector **r** can be expressed in terms of **c** and **s**. The point **a** is the foot of the perpendicular of the point **s**. Furthermore **a** is used to express **c** in terms of a set of two basis vectors $\mathbf{b_1}$ and $\mathbf{b_2}$ of the surface with the normal **n**. Consequently **c** is computed by $d \cdot \mathbf{b_1} + t \cdot \mathbf{b_2}$ whereas d and t vary.

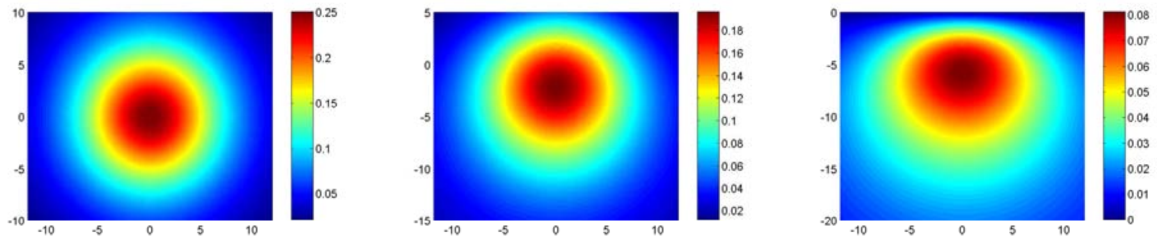
But first it is necessary to examine the distributions of form factors for a given scene. Imagine a scene as shown in figure 3.4. Let **n** and **m** be the normals of both fixed planes in figure 3.4 and s the point where the form factor is evaluated. Furthermore let **c** be the position of the light source with a fixed light strength and size. The position of the light source varies on the surface with normal **n**. The form factor distributions depend on the angle between the normals **m** and **n**. Let α be the angle between these normals. The properties of a distribution with angle $-\alpha$ are the same as the properties with angle α except the fact that the distribution is mirrored. The distributions in figure 3.5 show how the form factors behave for a change of the constellation of the two planes. These distributions are projections along **n** to the respective plane where the light source is located. For each light source position **c** the form factor observed at **s** is recorded at the respective position **c**.

The settings chosen for the plots in 3.5 are the following. Without loss of generality **n** is assumed to be $(0,1,0)$ because each arbitrary setting of two planes can be rotated that one normal is $(0,1,0)$. In the same way the foot of the perpendicular **a** as shown in figure 3.4 is chosen as the origin $(0,0,0)$. Furthermore any two chosen vectors **n** and **m** always lie in one common plane. The distance $|\overline{\mathbf{as}}|$ is fixed at $10m$ as well as the light source size is chosen to be $5\pi m^2$. The light strength is set to 5.

Plot 3.5(a) shows a setting where $\mathbf{m} = -\mathbf{n}$. As mentioned before in the introductory section on radiosity (see section 3.1.1) the maximum form factor is achieved by this configuration of the normals. As seen in the plot the distribution for this setting are concentric circles around the maximum. The other plots are more interesting. The second plot 3.5(b) shows the form factor distribution for a setting where the angle between the normal vectors **n** and **m** is $\frac{\pi}{4}$. The distribution around the maximum still looks like circles, however, these circles seem to be deformed to ellipses the farther away from the maximum. Furthermore these ellipses do not share a common midpoint. The last plot shows a setting for an angle of $\frac{\pi}{2}$. The distribution looks completely different to the both shown before. The contour lines of this distribution are different from circles or ellipses as they are deformed along the minor axis. In the process of the development of this theory they are called *deformed ellipses*. Another important fact is that the amount of received energy decreases for an increasing angle between **n** and **m**. A change in the light source parameters as well as in the distance of the observed point **s** only up- or downscale the form factors and consequently do not effect the distribution. The same distribution of form factors is also achieved on a plane illuminated by a fixed light source located on another fixed plane.

It is important to find a formula representing these deformed ellipses. These ellipses define the set of possible light source positions on a specific surface for one observed form factor. Furthermore it is important that normal ellipses and even circles can also be represented by this formula considering that circles and ellipses describe possible sets of light source positions.

The following section presents and analyzes the formula describing the deformed ellipses.



(a) (b) (c)

Figure 3.5: These plots show the distribution of form factors of an observed point for a giving setting as in figure 3.4. These distributions are projections along **n** to the respective plane where the light source is located. For each light source position **c** the form factor observed at **s** is recorded at the respective position **c**. Another important fact is that the amount of received energy decreases for an increasing angle between **n** and **m**. Without loss of generality **n** is assumed to be $(0,1,0)$ because each arbitrary setting of two planes can be rotated that one normal is $(0,1,0)$. In the same way the foot of the perpendicular **a** is chosen as the origin $(0,0,0)$. The distance $|\overline{\mathbf{as}}|$ is fixed at $10m$ as well as the light source size is chosen to be $5\pi m^2$. The light strength is set to 5. Plot (a) shows a setting where $\mathbf{m} = -\mathbf{n}$, that leads to a configuration of concentric circles around the maximum. The second plot (b) shows the form factor distribution for a setting where the angle between the normal vectors **n** and **m** is $\frac{\pi}{4}$. The distribution around the maximum still looks like circles, however, these circles seem to be deformed to ellipses the farther away from the maximum. Furthermore these ellipses do not share a common midpoint. The last plot shows a setting for an angle of $\frac{\pi}{2}$. The distribution looks completely different to the both shown before. The contour lines of this distribution are different from circles or ellipses as they are deformed along the minor axis. These deformed ellipses describe a set of possible light source positions on one specific surface for one illuminated point.

3.2.3 The Deformed Ellipses

The ellipses representing the contour lines of the form factor distributions are deformed along the minor axis. To obtain ellipses like the ones in the distributions an offset needs to be applied along the minor axis. A choice for the offset for an explicit representation is a function which is zero at the angles representing the major apices and one at the angles for the minor apices. A cosine function with doubled frequency serves our purposes as it fulfills the previously mentioned requirements. Therefore an explicit formula describing such deformed ellipses is the following:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} b \cdot (\cos(t) + d \cdot \cos(2t)) \\ a \cdot \sin(t) \end{pmatrix} \tag{3.20}$$

whereas a and b define the major apex length and the minor apex length. The deforming of the ellipse depends on the *deformation factor d*. As mentioned before it is necessary that this formula can also describe circles and normal ellipses. Ellipses with major apex length a and minor apex length b are defined when the deformation factor d is set to 0. Consequently circles with radius r are achieved when $r = a = b$ and $d = 0$.

Figure 3.6(a) shows such a deformed ellipse with the relation $a : b : d$ set to $12 : 10 : 1$. The figure 3.6(b) shows a sequence of such deformed ellipses with increasing factors a, b and d. That figure demonstrates the similarities to the form factor distribution of the plot in 3.5(c). It is important to notice that all ellipses of figure 3.6(b) share the same minor axis and all according major axes are parallel to each other, therefore sharing the same direction.

Figure 3.7 expresses the differences between a normal ellipse with major apex length a and minor apex length b (shown in blue) and a deformed ellipse (shown in red) with a and b set to the same value as in the normal ellipse. The minor axis of the deformed ellipse is defined as the axis of symmetry. Consequently the minor apices are defined as the intersection points of the minor axis with the ellipse. The remaining major apices of the deformed ellipse are defined as the points with the farthest distance to the axis of symmetry. The major axis is defined as the straight line connecting both major apices. In this figure the major axis is the horizontal axis and therefore the minor axis is the vertical one. The red and blue spots indicate the respective major and minor apices of the according ellipses.

Both ellipses share the same point from which they are drawn (green spot) with the help of their explicit formulation[7]. What is interesting is that the intersection point of the major and minor axis of both ellipses is different. Furthermore the green spot indicates the intersection point of both axes of the normal ellipse. The orange spot is the intersection point of minor and major axis of the deformed ellipse. This displacement of the intersection point is due to the deformation of the

[7]explicit formulation of normal ellipse: $\binom{x}{y} = \binom{b \cdot \cos(t)}{a \cdot \sin(t)}$

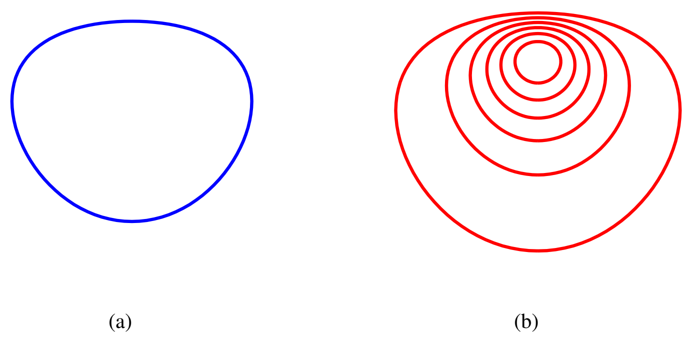


Figure 3.6: Figure (a) shows a deformed ellipse with the relationship $a : b : d$ set to $12 : 10 : 1$. The second Figure shows a sequence of deformed ellipses with increasing factors a, b and d and demonstrates many similarities to the form factor distribution of the plot in 3.5(c). It is important to notice that all ellipses of figure (b) share the same minor axis and all according major axes are parallel to each other, therefore sharing the same direction.

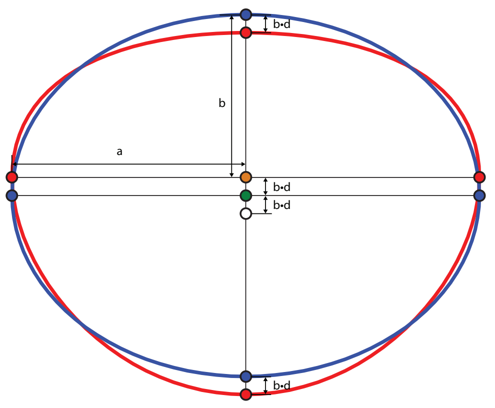


Figure 3.7: Comparison of a normal ellipse (in blue) and a deformed ellipse (in red) with the same major and minor apex lengths a and b. The red and blue points represent the major and minor apices. The minor axis of the deformed ellipse is defined as the axis of symmetry. The according major axis is the connection of both points with the farthest distance to the minor axis. The apices are defined as the intersection points of the ellipse with the respective axis. Both ellipses are drawn out of the same midpoint (green spot). Furthermore the green spot indicates the midpoint of both minor apices and the intersection point of the minor and major axis in case of the normal ellipse. The orange point is the intersection point of the axes of the deformed ellipse and the white point is the midpoint of both minor apices of the deformed ellipse. These points as well as all apices are displaced by $b \cdot d$ in case of the deformed ellipse.

ellipse along the minor axis. The midpoint of the minor apices is displaced as well. This midpoint is represented as the green spot in case of the normal ellipse and as the white spot in case of the deformed ellipse.

The point from which the ellipses are drawn, the intersection point of the axis, and the midpoint of the minor apices are the same as in the case of the normal ellipse. These points as well as all apices are displaced for the deformed ellipse in respect to the deformation factor d. The displacement takes place along the minor axis, therefore the displacement also depends on b. Hence all mentioned points are displaced by $b \cdot d$.

For fitting purposes an implicit formulation of such a deformed ellipse is required. To reformulate an explicit formulation into an implicit formulation, it is necessary to express the varying factor t in terms of a single coordinate. The parameter t can then be substituted in the other coordinate. The expression of t in terms of y and the reformulated cosine terms of the x-coordinate are presented in the equations in 3.21.

$$t = \arcsin\left(\frac{y}{a}\right) \tag{3.21a}$$

$$\cos(t) = \sqrt{1 - \frac{y^2}{a^2}} \tag{3.21b}$$

$$\cos(2t) = 1 - 2\frac{y^2}{a^2} \tag{3.21c}$$

The implicit formulation then looks as follows:

$$x = b\left(\sqrt{1 - \frac{y^2}{a^2}} + d\left(1 - 2\frac{y^2}{a^2}\right)\right) \tag{3.22}$$

and can be simplified to:

$$x^2 - 2x \cdot b \cdot d + \frac{4x \cdot b \cdot d \cdot y^2}{a^2} + \frac{(b^2 - 4b^2 \cdot d^2) \cdot y^2}{a^2} + \frac{4b^2 \cdot d^2 \cdot y^4}{a^4} - b^2 + b^2 \cdot d^2 = 0 \tag{3.23}$$

For describing ellipses that are not located in the origin, the midpoint of the ellipse (x_m, y_m) must be taken into respect leading to the finale implicit representation of the deformed ellipse:

$$\begin{aligned}(x - x_m)^2 - 2(x - x_m) \cdot b \cdot d + \frac{4(x - x_m) \cdot b \cdot d \cdot (y - y_m)^2}{a^2} \\ + \frac{(b^2 - 4b^2 \cdot d^2) \cdot (y - y_m)^2}{a^2} + \frac{4b^2 \cdot d^2 \cdot (y - y_m)^4}{a^4} - b^2 + b^2 \cdot d^2 = 0\end{aligned} \tag{3.24}$$

The next step in defining a set of possible light source positions is determining the parameters of a deformed ellipse in a given form factor distribution. The process of fitting such deformed ellipses is explained in the following section.

3.2.4 The Fitting Process

Assume a setting as shown in figure 3.4 is given. In addition to this setting the value A, which is the product of the desired light source strength and size, is also given. This time the illumination at the point $\mathbf{s}$ is given and the set of light source positions $\mathbf{c}$ is desired. From the observation and the analysis above it is known that $\mathbf{c}$ has the form of a deformed ellipse. Therefore we are interested in the parameters a, b and d of the deformed ellipse describing the positions of light sources with given light strength and size that yield in the given illumination in s. What is furthermore required is the midpoint of this ellipse, therefore two additional parameters are necessary x_m and y_m. These two parameters are expressed as the distance from the maximum of the distribution along the minor and major axis of the ellipses.

To perform a fitting of a specific ellipse the minor and major axis direction of the according form factor distribution needs to be found first. These two directions are also basis vectors of the plane where the light source is located. The major axis direction is achieved as the normal vector of the plane spanned by the normal vectors $\mathbf{m}$ and $\mathbf{n}$. Consequently the direction can be inferred from the outer product of $\mathbf{m}$ and $\mathbf{n}$. As the minor axis is normal to the major axis and lies in the same plane defined by $\mathbf{n}$, the minor axis direction is the result of the outer product of $\mathbf{n}$ and the major axis direction. The formulas in 3.25 show the respective calculations.

$$\text{major axis direction} = \mathbf{n} \times \mathbf{m} \tag{3.25a}$$

$$\text{minor axis direction} = \mathbf{n} \times \text{major axis direction} \tag{3.25b}$$

These axes directions also serve as the two basic vectors $\mathbf{b_1}$ and $\mathbf{b_2}$ of the plane containing the light source. What is required next is the coordinate as well as the value of the maximum of the according form factor distribution. The position of the maximum is necessary because this position defines the minor axis because the minor axis goes straight through the maximum. The value, however, is an indicator whether any light source on this plane can ever achieve an illumination that has been observed at $\mathbf{s}$. If the maximum of the distribution is less than the observed illumination than no light source can achieve this desired illumination.

To achieve the coordinates of the maximum a reformulation of the form factor equation in 3.19 is necessary. A formulation in terms of multiplies of $\mathbf{b_1}$ and $\mathbf{b_2}$ is necessary. As mentioned before the vector $\mathbf{r}$ can be expressed in terms of $\mathbf{s}$ and $\mathbf{c}$. Furthermore, $\mathbf{c}$ can be expressed in terms of $\mathbf{a}$, the foot of the perpendicular, and multiples of $\mathbf{b_1}$ and $\mathbf{b_2}$. This yields to the formulation of the vector $\mathbf{r}$:

$$\mathbf{r} = \mathbf{a} + d \cdot \mathbf{b_1} + t \cdot \mathbf{b_2} - \mathbf{s} \tag{3.26}$$

The form factor equation can be reformulated to:

$$f(d,t) = \frac{\frac{\langle \mathbf{n}, -\mathbf{r} \rangle}{||\mathbf{r}||} \cdot \frac{\langle \mathbf{m}, \mathbf{r} \rangle}{||\mathbf{r}||}}{\pi \cdot ||\mathbf{r}||^2} \cdot A \tag{3.27}$$

Due to the fact that $\mathbf{r}$ is in general not normalized it is necessary that each dot product gets divided by the length of $\mathbf{r}$. This formula can then be simplified to:

$$f(d,t) = \frac{\langle \mathbf{n}, -\mathbf{r} \rangle \cdot \langle \mathbf{m}, \mathbf{r} \rangle}{\pi \cdot ||\mathbf{r}||^4} \cdot A \tag{3.28}$$

This formula can now be maximized in terms of d and t by using either a deterministic or a nondeterministic optimizing strategy. To achieve good results with any optimizing strategy a good starting point for the optimization is required. The foot of the perpendicular $\mathbf{a}$ turned out to be a good choice.

With the coordinates of the maximum of the distribution and the directions of the major and minor axis the deformed ellipse for a specific value can be fitted. To do so the implicit formula formulation of the deformed ellipse is used as a quadratic error measure. Equation 3.29 shows the corresponding quadratic error measure:

$$\begin{aligned} f(a,b,d,x_m,y_m,x,y) = \Bigg(& (x-x_m)^2 - 2(x-x_m) \cdot b \cdot d + \frac{4(x-x_m) \cdot b \cdot d \cdot (y-y_m)^2}{a^2} + \\ & \frac{(b^2 - 4b^2 \cdot d^2) \cdot (y-y_m)^2}{a^2} + \frac{4b^2 \cdot d^2 \cdot (y-y_m)^4}{a^4} - b^2 + b^2 \cdot d^2 \Bigg)^2 \end{aligned} \tag{3.29}$$

With equation 3.29 as error measure all that is left is a set of samples of the desired deformed ellipse. These samples are acquired via binary search. The maximum is the highest value in the distribution consequently the values decrease in any direction. As mentioned before it is necessary that the desired value is less than the maximum to obtain a set of potential light source position on the current surface. The binary search for the desired value is done around the maximum for a predefined amount of times. Let k be the amount of samples $(x_i, y_i), 1 \leq i \leq k$ taken. The mean squared error is then for a set of ellipse parameters (a,b,d,x_m,y_m) is then computed as:

$$mse(a,b,d,x_m,y_m) = \frac{1}{k} \sum_{i=1}^{k} f(a,b,d,x_m,y_m,x_i,y_i) \tag{3.30}$$

The error function in 3.30 needs to be minimized in respect to the ellipse parameters (a,b,d,x_m,y_m). Because of the identity of

$$\nabla mse(a,b,d,x_m,y_m) = \frac{1}{k} \sum_{i=1}^{k} \nabla f(a,b,d,x_m,y_m,x_i,y_i) \tag{3.31}$$

it is sufficient to derive $f(a,b,d,x_m,y_m,x,y)$ by a, b, d, x_m, and y_m if a deterministic optimizing approach is chosen. The variables x and y are constant because they resemble the coordinates of the sample points.

As before an optimal start configuration needs to be found. Approximations for a and b are necessary to achieve good fitting results. It is possible to calculate b accurately by taking the half of the distance between both samples along the minor axis. This is possible because the maximum is located on the minor axis. A good approximation for a is the length of the distance between the maximum and the result of a binary search along the major axis direction. The parameter d can be set to 0. Good values for x_m and y_m are based on the coordinates of the maximum.

Each point in the three-dimensional space can be expressed as a linear combination of basis vectors of the three-dimensional space as shown as follows:

$$\mathbf{p} = \lambda \cdot \mathbf{e_1} + \rho \cdot \mathbf{e_2} + \tau \cdot \mathbf{e_3} \tag{3.32}$$

whereas $\mathbf{p}$ is the three-dimensional point expressed by the three basis vectors $\mathbf{e_1}$, $\mathbf{e_2}$ and $\mathbf{e_3}$. In general these basis vectors are $(1,0,0)$, $(0,1,0)$ and $(0,0,1)$. For points located on our plane we can also choose $\mathbf{n}$, $\mathbf{b_1}$ and $\mathbf{b_2}$. By using these three vectors each point on our plane has the same multiple for the normal n. The multiples (λ, ρ, τ) can be calculated by solving the following linear equation:

$$\begin{pmatrix} \mathbf{n} & \mathbf{b_1} & \mathbf{b_2} \end{pmatrix} \begin{pmatrix} \lambda \\ \rho \\ \tau \end{pmatrix} = \mathbf{p} \tag{3.33}$$

Therefore starting values for x_m and y_m can be found by taking the multiples of the minor and major axis of the linear combination representing the maximum. The midpoint of the ellipse is also located on the minor axis therefore y_m should not change during the minimization process.

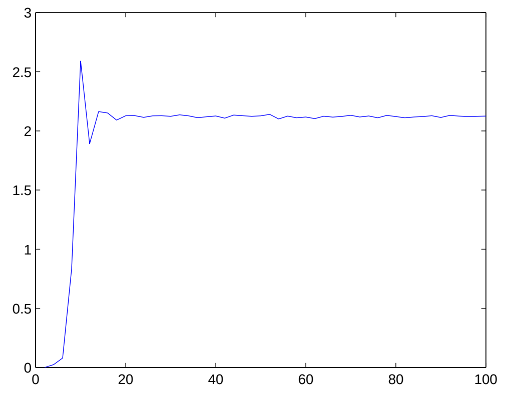


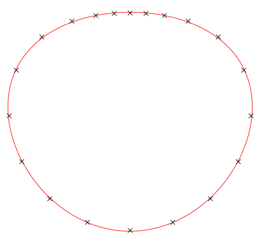
Figure 3.8: Evolution of the mean squared error for fitting a specific ellipse for a number of samples going from 2 to 100. The low error value in the interval $[2, 20[$ is due to the insufficient amount of samples and is therefore underfitting. Because the taken samples are accurate and do not contain any noise overfitting is not taking place. The mean squared error is almost constant for a sample count greater than 20.

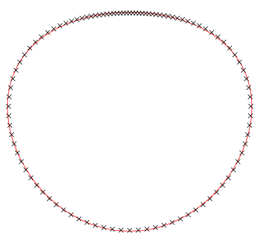
Figure 3.8 displays the evolution of the mean squared error for fitting a specific ellipse for a number of samples reaching from 2 to 100. The low error value in the interval $[2, 20[$ is due to the insufficient amount of samples and is therefore underfitting. Because the taken samples are accurate and do not contain any noise overfitting is not taking place. The mean squared error is almost constant for a sample count greater than 20. Therefore in this case 20 is the best choice for the amount of samples.

Figures 3.9(a) and 3.9(b) show the difference between 20 and 100 samples. It is important to notice that the distribution of samples is denser in the upper part of the ellipse. This is due to the fact that we take the samples based on the position of the maximum that lies not in the midpoint of the ellipse. The resulting ellipses are almost the same therefore in this case there is no need in taking more than 20 samples. Figure 3.9(c) shows the evolution of the fitted ellipse in terms of increasing samples. The red ellipse is taken with 2 samples, the blue with 4 samples, the green with 6 samples, the orange with 10 samples, and the black with 20 samples. The orange ellipse is close to the black one and therefore hardly visible.

Figure 3.10 shows the evolution of the ellipse parameters for fitting ellipses in certain intervals in the distribution where the angle between m and n is $\frac{\pi}{2}$ (see figure 3.5(c)). The maximum of this distribution is at 0.081 and all ellipses are fitted using 20 samples. The horizontal axis shows the value for which the deformed ellipse is fitted. The vertical axis shows the value of the respective parameter. Plot 3.10(a) shows the evolution of a, plot 3.10(b) the evolution of b and plot 3.10(c) the evolution of d. The process of those three evolutions is very similar with only differences in the values. The fourth plot shows the distance between the maximum of the distribution and the midpoint of the fitted ellipse for decreasing fitting value.

Note that in case of a setting $\mathbf{m} = -\mathbf{n}$ the fitting process except for the maximum search is not necessary. The distribution resulting from that setting is a set of concentric circles and the maximum of the distribution is the midpoint of the circles. Consequently no minor and major axes are required and any basis vectors of the light source containing


(a)


(b)

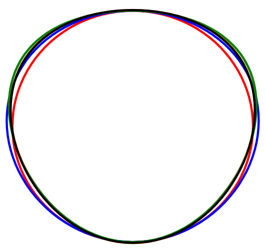
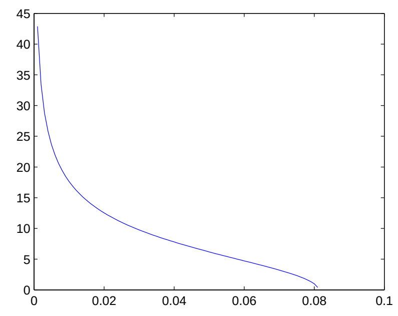
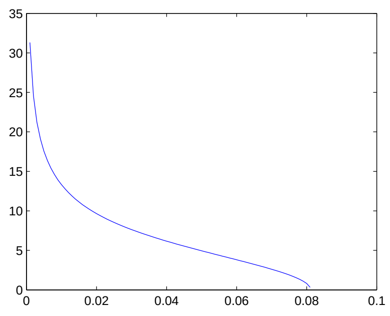

(c)

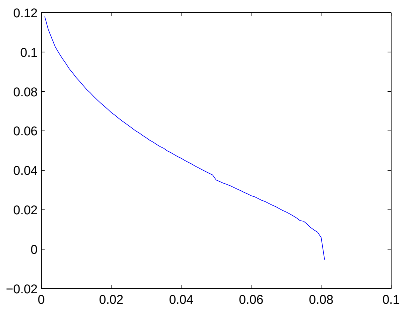
Figure 3.9: The first two figures show a comparison of the ellipse fitting in taking 20 or 100 samples. The result is almost the same therefore 20 samples are sufficient. The third figure shows the evolution of the fitted ellipse in terms of increasing samples. The red ellipse is taken with 2 samples, the blue with 4 samples, the green with 6 samples, the orange with 10 samples, and the black with 20 samples. The orange ellipse is close to the black one and therefore hardly visible.



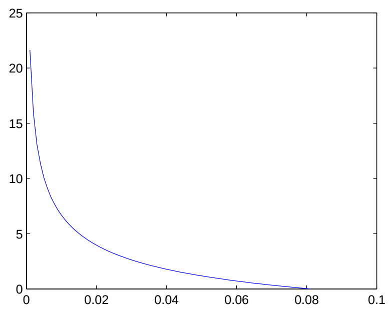
(a)



(b)



(c)



(d)

Figure 3.10: These plots show the evolution of the ellipse parameters for fitting ellipses in certain intervals in the distribution where the angle between m and n is $\frac{\pi}{2}$. The maximum of this distribution is at 0.081 and all ellipses are fitted using 20 samples. The horizontal axis shows the value for which the deformed ellipse is fitted. The vertical axis shows the value of the respective parameter. Plot (a) shows the evolution of a, plot (b) the evolution of b and plot (c) the evolution of d. The process of those three evolutions is very similar with only differences in the values. The fourth plot shows the distance between the maximum of the distribution and the midpoint of the fitted ellipse for decreasing fitting value.

plane are sufficient. The radius of the circle describing the set of potential light source positions is achieved by one single binary search.

The final section in this chapter summarizes all observations and findings discussed so far and formulates the improved voting system.

3.2.5 The Improved Voting System

With a representation of the deformed ellipses and a method to calculate their parameters for a given setting it is possible to process votes from illuminated points. An illuminated point needs to vote to each existent plane in the scene. However, there are some exceptions. It is not necessary to vote to the plane the voting plane belongs to because planes cannot illuminate themselves directly. It is furthermore not necessary to vote to a plane that has the same normal vector as the voting point. That plane is not visible to the voting point, consequently no vote is necessary.

Thus each illuminated point associates a set of light source positions in form of a deformed ellipse for each plane. Light sources placed on the exact border of this ellipse produce exactly the illumination observed at that point. Is a light source placed inside of an ellipse it produces a higher illumination at that point and light sources placed outside will produce a lower illumination instead. As a desired lighting configuration can also be obtained from more than one light source each point actually needs to vote for all points outside and on the border of the fitted deformed ellipse. However, it is hard to say how much of the observed illumination is produced by what light source, there are infinite possibilities. Therefore most attention is directed to the points mostly illuminated by one light source. As a consequence a vote from one point to one plane only consists out of the border of the ellipse. So all points are voting only for that light sources that will exactly produce their illumination (without considering secondary illumination).

Figure 3.11 illustrates the voting process. A vote on a plane is based on the fitted ellipse for that plane to the point's specific illumination value. For illustration purposes just three points cast votes in this example. The red, green and blue point vote inside the illustrated box with their illumination. These points vote to all planes except their own. The three ellipses intersect in one common point in the back of the box. There is no other intersection point with more than two ellipses intersecting therefore this intersection point (shown in white) is the most likely light source position.

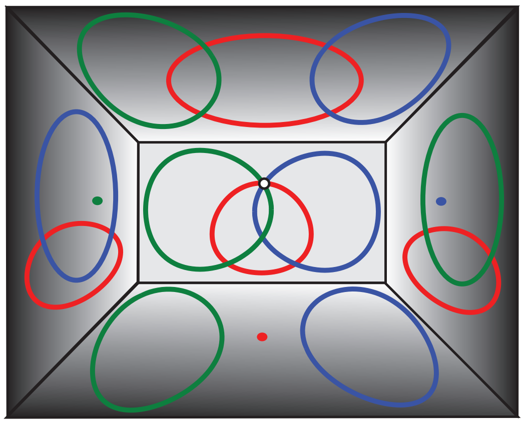


Figure 3.11: This figure illustrates the voting process. A vote on a plane is based on the fitted ellipse for that plane to the point's specific illumination value. For illustration purposes just three points cast votes in this example. The red, green and blue point vote inside the illustrated box with their illumination. These points vote to all planes except their own. The three ellipses intersect in one common point in the back of the box. There is no other intersection point with more than two ellipses intersecting therefore this intersection point (shown in white) is the most likely light source position.

To obtain a set of potential light source positions it is necessary to process more than one illuminated point. For a complete vote distribution a well distributed set of illuminated points in the scene has to vote to each existent plane in the scene. Accumulation points in the resulting vote distribution are indicators for potential light source positions. With the partly illuminated scene as input it is no problem to calculate all plane equations for all present surfaces. To control the density of votes a predefined "votes per square meter" factor is crucial.

Based on the area of each patch a set of voting points is distributed among the surface of this specific patch. All these points vote to all existent planes in the scene based on their illumination value. This process will produce a vote distribution for all planes of the scene. With such a vote distribution only light sources placed on the geometry can be processed. For obtaining light source positions in midair additional planes to vote to are necessary. These additional planes are called *dummy planes*.

In reality most lamps that are located in midair are lamps hanging from the ceiling. Based on that fact a predefined amount of dummy planes are used with predefined spacing between them. That sequence of dummy planes is placed in layers parallel to the floor beginning at the ceiling. The normal of these planes are directed towards the floor. The amount and position of dummy planes can be adapted if the need arises. Figure 3.12 demonstrates the use of dummy planes. With the help of these dummy planes processing light sources located in midair becomes possible.

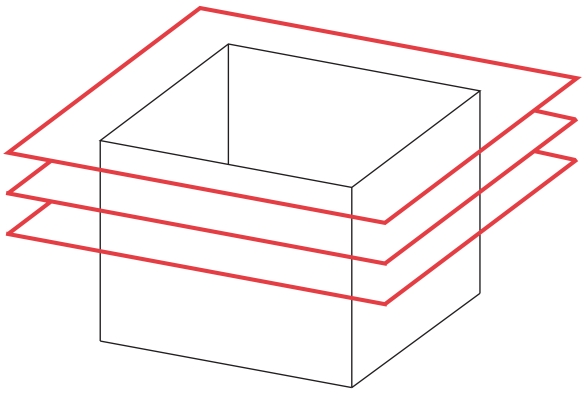


Figure 3.12: Demonstration of the use and purpose of dummy planes. These additional planes are used to obtain light sources that are located in midair. These planes are placed in layers parallel to the floor beginning at the ceiling. The normal vector of these planes faces towards the floor.

To prevent the vote distribution from becoming too ambiguous several techniques are applied. It is obvious that it is necessary to apply a weighting to the votes. Two different methods are discussed in the following paragraphs. One possible weighting strategy is a weighting based on the illumination value. Points with a higher illumination are consequently placed closer to the light source than ones with a lower illumination value. As a consequence patches placed close to a light source are more important than the ones farther away or the ones being illuminated indirectly. Furthermore points with a low or no illumination value have hardly any influence on the result. To ensure an equally distributed weighting the weighted votes are scaled by the maximum illumination value of the scene.

Another weighting strategy is a weighting based on the distance between voting point and surface it votes to. With this strategy votes to nearby surfaces are enhanced and votes to faraway surfaces are reduced. A *normalized form factor*[8] is an aid for this purpose, as it already takes distance and arrangement of the two involved planes into account.

A further way to avoid misleading accumulation points in the resulting vote distribution is to process occlusion during the vote casting. Therefore only the not occluded part of the deformed ellipse is voted. Consequently no votes to not visible surfaces are processed.

One remaining question is how those resulting vote distributions are interpreted. Figure 3.13 shows common vote patterns around potential accumulation points indicating a light source position. As the size of the vote ellipses depends on the chosen light source size and light source strength. These parameters need to be adapted based on the resulting vote distribution. This adaption has to be done manually. The first sub-figure 3.13(a) shows a case where most vote ellipses intersect each other in a common point. Consequently the light source parameters are chosen right. The following figure 3.13(b) shows a case where the light source parameters are chosen too small. The ellipse fronts from the sides do not intersect each other. Increasing the light source size or strength means that the light source needs to be farther away from the voting point to achieve the given illumination at that point. Due to the fixed plane and the fixed point the only way for the light source to increase the distance to the voting point is to increase the ellipses. Bigger ellipses can be achieved by increasing the light source parameters. With bigger ellipses a intersection will happen. The third picture 3.13(c) shows a case where the ellipses intersect each other, but there is no point where intersection points gather. The ellipses are too big because the light source has been chosen too big or too bright. There is empty space between the ellipse fronts, an indicator for wrongly chosen settings. Decreasing the light source parameters means that the light source needs to decrease the distance to the voting point. The only solution is to change the ellipses, in this case to shrink them. Consequently it is necessary to lower the light source parameters. The last picture shows a case where the light source parameters are chosen far too big. The empty space between the ellipse fronts is bigger, meaning the ellipses need to be much smaller to intersect in a single point.

[8]form factor without considering the area of the receiving patch: $\frac{\cos(\Theta_i)\cdot\cos(\Theta_k)}{\pi\cdot r^2}$

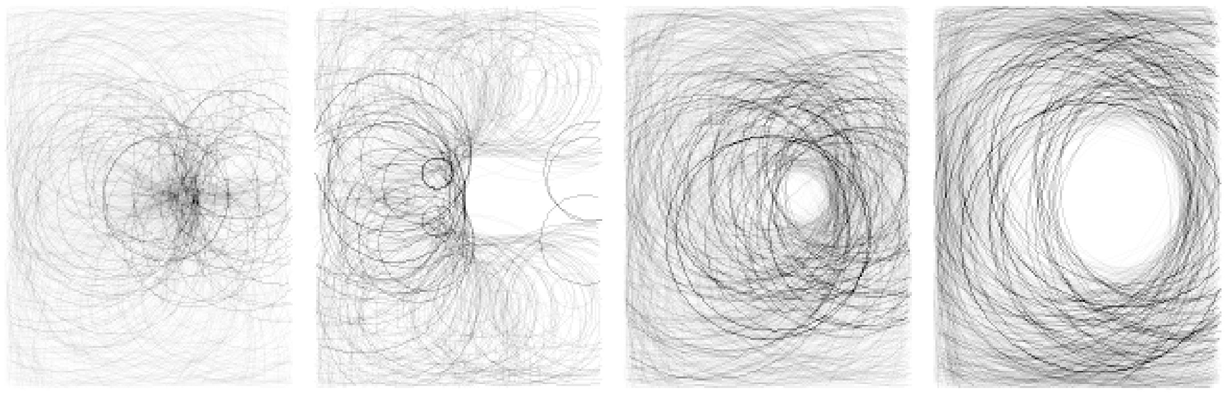


Figure 3.13: This sequence of figures demonstrates how vote distributions need to be interpreted. The first sub-figure (a) shows a case where most vote ellipses intersect each other in a common point. Consequently the light source parameters are chosen right. Figure (b) shows a case where the light source parameters are chosen too small. The ellipse fronts from the sides do not intersect each other. Increasing the light source size or strength means that the light source needs to be farther away from the voting point to achieve the given illumination at that point. Due to the fixed plane and the fixed point the only way for the light source to increase the distance to the voting point is to increase the ellipses. Bigger ellipses can be achieved by increasing the light source parameters. With bigger ellipses a intersection will happen. The third picture (c) shows a case where the ellipses intersect each other, but there is no point where intersection points gather. The ellipses are too big because the light source has been chosen too big or too bright. Decreasing the light source parameters means that the light source needs to decrease the distance to the voting point. The only solution is to change the ellipses, in this case to shrink them. Consequently it is necessary to lower the light source parameters. The last picture shows a case where the light source parameters are chosen far too big.

Figure 3.14 shows the vote distribution for five consecutive dummy planes. Those five dummy planes are layered as shown in figure 3.12. The first distribution belongs to the plane with the greatest distance to the floor and consequently the last distribution has the smallest distance to the floor. The distribution of the first dummy plane shows no clear accumulation point, but a ring of approximately the same value. Consequently the ellipses do not intersect in one point and the distance of the plane to the voting points is too big. To increase the ellipses the light parameters need to be increased or the distance to the voting points decreased. This time we are given the freedom to change the position of the plane, therefore we look at the distribution of the next dummy plane which is located closer to the floor. The ring closes in the second distribution, but not completely. The third dummy plane shows then a clear accumulation point, which should be chosen as the light source's position. The accumulation point begins to disperse in the fourth and the fifth distribution because those planes are placed too close and the ellipses become too big. This dispersion is due to the intersection points of the ellipses being scattered and this is an indication of too big ellipses.

What is left is to detect accumulation points in the resulting distributions. The following algorithm is pixel based and uses the vote distribution that is written to a texture as input. The derived voting distributions looks similar to Hough transforms. Therefore a similar approach for peak detection in Hough transforms is applied. One of these voting distributions is seen in figure 3.15(a). For a better illustration of the process, a lower resolution of figure 3.15(a) is used as seen in figure 3.15(b). First to enhance local maxima a Laplace filter is applied. The resulting picture of the distribution in figure 3.15(b) is seen in figure 3.15(c). The next step is applying a so called *non-maxima suppression*. The value of a pixel is suppressed, respectively set to 0, when there exists a neighboring pixel with a higher value. Otherwise the pixel is conserved with its own value. After applying the non-maxima suppression the dynamic range of the image is mapped to the range $(0,255)$. This mapping is, on the one hand, applied to be able to visualize the result within a gray scale image and, on the other hand, to ensure a fixed maximum value of 255 and a fixed minimum value of 0 for the next step. Finally a threshold is applied and all pixel with a value greater than the threshold value are candidates for a light source position. The non-maxima suppressed image of the laplace filtered image in figure 3.15(c) is seen in figure 3.15(d). The single red pixel indicates the most likely position of a light source of chosen size and brightness. The threshold is set to 254 consequently only the most likely positions are returned.

This ends the chapter about theory. The following chapter focuses on implementation strategies as well as on performance issues.

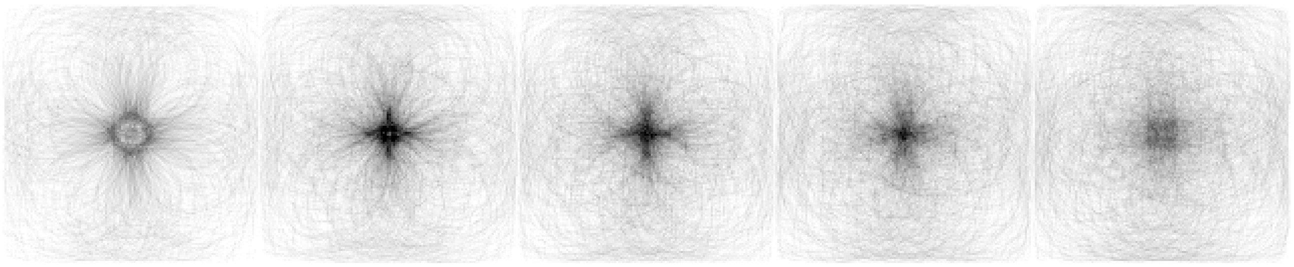


Figure 3.14: This figure shows the vote distribution for five consecutive dummy planes layered as in figure 3.12. The first distribution belongs to the plane with the greatest distance to the floor and consequently the last distribution has the smallest distance to the floor. The distribution of the first dummy plane shows no clear accumulation point, but a ring of approximately the same value. Consequently the ellipses do not intersect in one point and the distance of the plane to the voting points is too big. To increase the ellipses the light parameters need to be increased or the distance to the voting points decreased. This time we are given the freedom to change the position of the plane, therefore we look at the distribution of the next dummy plane which is located closer to the floor. The ring closes in the second distribution, but not completely. The third dummy plane shows then a clear accumulation point, which should be chosen as the light source's position. The accumulation point begins to disperse in the fourth and the fifth distribution because those planes are placed too close and the ellipses become too big. This dispersion is due to the intersection points of the ellipses being scattered and this is an indication of too big ellipses.

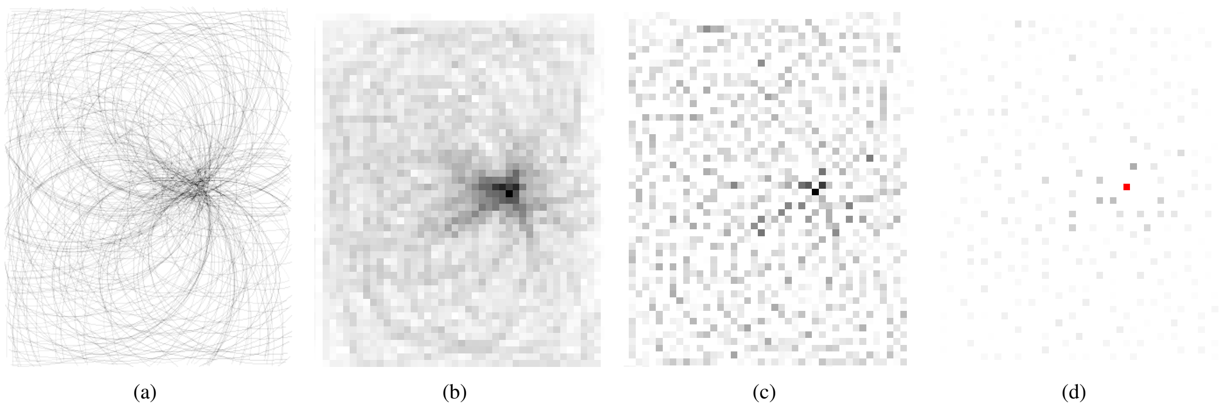


Figure 3.15: This figure illustrates the process of detecting potential light source positions out of a derived voting distribution. Figure (a) shows the resulting vote distribution texture. For a better illustration of the process, a lower resolution of figure (a) is used as seen in figure (b). The third picture (c) shows the result after applying a Laplace filter to enhance local maxima. The last figure (d) shows the non-maxima suppressed image to locate real local maxima and suppress everything else. After the non-maxima suppression the pixel values are mapped to the range $(0,255)$ and a threshold is applied. The single red pixel is the highest likely position of a light source of chosen size and brightness and is achieved after applying a threshold of 254.

Chapter 4

Implementation

Contents

This chapter focuses on the software that has been developed during this thesis. This software has been developed using the programming language C++. The graphical user interface is designed with the help of QT [QT]. As scene graph library Coin3D [Coi] is used. Coin3D is a OpenGL based scene graph rendering library and is based on Open Inventor [Ope]. For displaying scene graph content in QT widgets Quarter [Qua] is used. Quarter provides a special widget named QuarterWidget for displaying Coin3D based scene graphs. All these libraries are open source and are platform independent.

All input scenes are modeled with Autodesk Maya [May] and are illuminated with the help of the mental ray renderer included in Autodesk Maya. These models are exported to VRML format as input for the developed program. The target illumination of the scene is stored in a texture for the whole scene.

This chapter first discusses several performance issues. Afterwards the new algorithm with respect to the previously explained performance issues will be described. At the end of this chapter a tutorial for the software developed throughout this thesis is presented together with examples.

4.1 Performance Issues

4.1.1 Precalculations

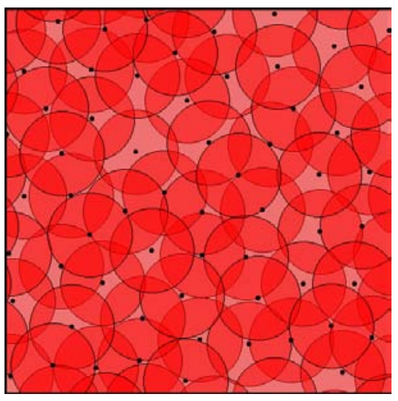
To decrease the amount of calculations during the runtime parts of the algorithm are precalculated. On the one hand, the distribution of the sample points and on the other hand the parameters of the fitted deformed ellipses for different settings are precalculated. All precalculations must be loaded on start up of the program. This results in a longer start up process, but minimizes the amount of time required for a single computation.

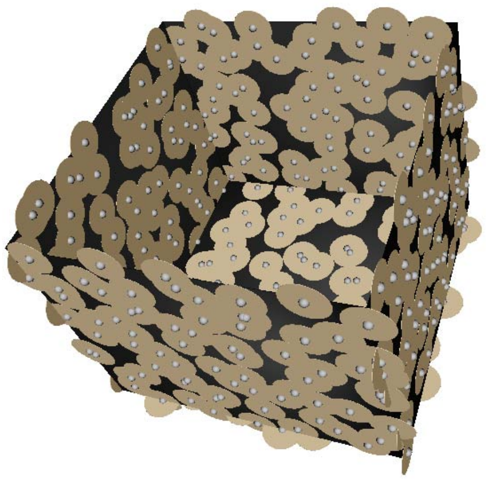
4.1.1.1 Poisson Disk Sampling

To guarantee well distributed voting points throughout the scene a *poisson disk* sampling is applied. Another reason to apply a poisson disk sampling is to reduce aliasing in the resulting vote distribution textures. Aliasing as well as unwanted accumulation points occur if a regular sampling is applied. A poisson disk sampling is a sampling for which each random generated point has at least a distance r to each other generated point. So each generated point has a circle containing no other points with radius r around it. This circle is called the poisson disk. Figure 4.1 shows such a poisson disk sampling of a square. The black dots represent the sample points and the red circles the respective poisson disks. The sampling is figure 4.1(a) is tileable because it is calculated on the surface of a sphere. Consequently a disk leaving the right side is continued at the left and vice versa. Of course the same property applies to the upper and the lower side.

The poisson disk sampling is done for each triangle of the scene. Consequently the distance between the sample points cannot be guaranteed along the borders of the triangles. To prevent calculation during the execution the coordinates of the poisson points inside a triangle are precalculated. The sample points inside of the triangles are calculated in means of barycentric coordinates. The sample points are precalculated and the according barycentric coordinates are saved as red, green and blue values inside a texture. During execution a random lookup in this precalculated texture is necessary to receive poisson disk distributed samples for each triangle. A poisson distribution of a simple scene is shown in figure 4.1(b).

Figure 4.1(c) shows the comparison between vote distributions obtained with different samplings. A regular sampling is used in the upper picture and the poisson disk sampling in the lower one. The result obtained with the regular sampling shows aliasing artifacts that lead to additional local maxima that are not present in the non regular sampling.


(a)


(b)

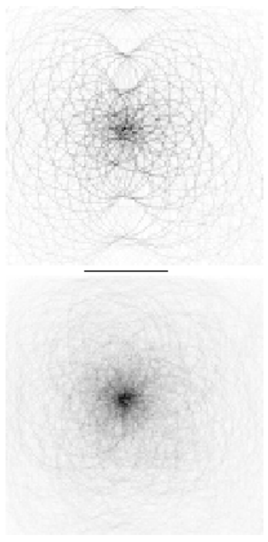

(c)

Figure 4.1: Figure (a) shows a poisson disk sampling of a square. The black dots represent the sample points and the red circles the respective poisson disks. This sampling is tileable, therefore it is calculated on the surface of a sphere. Leaving the right side means appearing on the left side and vice versa. Of course the same property applies to the upper and the lower side. Figure (b) shows a poisson disk sample of a simple scene. The spheres represent the poisson points and the golden cylinders the poisson disks. The minimum distance between two poisson points cannot be guaranteed along the borders of triangles. The last figure (c) shows the comparison between vote distributions obtained with different samplings. A regular sampling is used in the upper picture and the poisson disk sampling in the lower one. The result obtained with the regular sampling shows aliasing artifacts that lead to additional local maxima that are not present in the non regular sampling.

4.1.1.2 Voting Table

Processing one vote is very time intensive. For one vote two different optimizing calculations are necessary. One the one hand, the maximum of the form factor distribution needs to be determined. And on the other hand, the according ellipse needs to be fitted based on the before determined maximum of the form factor distribution. These optimizing processes need a lot of time independently of the chosen optimization strategy.Both optimizing strategies are too time consuming to yield an interactive system. The long execution time is mostly due to the number of processed votes. However, a lot of votes are necessary to guarantee an appropriate result.

To prevent the calculation of the optimizers for each vote, a table containing all necessary voting information is precalculated. This three-dimensional *voting table* contains following information for different arrangements of voting point and plane to vote to:

1. value of the maximum of the according form factor distribution

2. angle for calculation of the direction which leads to the position of the maximum
3. value of the form factor distribution on which the ellipse parameters are based
4. deformed ellipse parameters a, b, d, x_m and y_m

To draw the according deformed ellipse it is necessary to calculate the midpoint of the ellipse. To achieve the position of the midpoint of the ellipse the position of the maximum is required first because:

$$\text{midpoint} = \text{maximum} + x_m \cdot \text{minor axis direction} + y_m \cdot \text{major axis direction} \tag{4.1}$$

where x_m and y_m are the offsets stored within the voting table. The position of the maximum is calculated with the angle stored in the voting table. A rotation of the voting point's normal vector is necessary to achieve the direction pointing to the maximum's position on the plane to vote to. The normal vector needs to be rotated in the plane spanned by itself and the normal of the plane to vote to. The according rotation axis is the major axis direction of the underlying form factor distribution. Consequently the normal vector of the voting point gets rotated around this major axis direction by the angle specified in the according voting table entry. The position of the maximum is then achieved by simply intersecting the vector specified by voting point and the rotated direction with the plane to vote to. The maximum leads to the position of the ellipse's midpoint and this midpoint is used to draw the specific deformed ellipse. Consequently with this voting table it is possible to replace the calculation of two optimizing strategies with one matrix multiplication and one vector - plane intersection.

The distance between voting point and plane to vote to along the planes normal is mapped to the first dimension of the voting table. The second dimension represents the angle between the normal vector of the plane the voting point belongs to and the normal of the plane to vote to. As both plane's normals always lie in one plane and the distribution just gets mirrored when the angle is negated no additional angle sampling is necessary. The third dimension represents the distance between the value to fit and the maximum's value. The ellipses parameters are sampled in predefined steps for a predefined number of times. The maximum's value is required for computation of the entry where the according ellipse parameters are stored. Therefore, the entry $(i, j, 0)$ for any i and j always stores only value of the maximum. Consequently the third dimension has one additional entry.

The precomputed values of the maximum and the values where the ellipse parameters originate from are all normalized and independent of light source size and brightness. Therefore lookup in the table must be done with a normalized values, which are independent of the chosen light source parameters. This normalized illumination value of the voting point is simply acquired by dividing the illumination value by the product of light source size and brightness.

In this thesis' implementation the first dimension covers the distance from 0.1 meters to 10 meters in 10 centimeter steps. The second dimension samples an angle of π with a step size of $\frac{\pi}{36}$. The third dimension samples the ellipse parameters starting by the value of the maximum 250 times with a step size of 0.0002. The dimension of the voting table therefore is $(100, 36, 251)$.

Figure 4.2 illustrates the choice of the three dimensions. Dimension one and two influence the resulting form factor distribution and dimension three samples the ellipses that belong to that distribution.

The speedup achieved by the lookup in the voting table instead of computing the optimization processes is more than 200%.

4.1.2 GPU Calculation

The initial implementation of this algorithm has been implemented in Python, a scripting language for Maya[May]. The advantage of this implementation is that all internal data structures of Maya are accessible to the programmer. Therefore there is no need to process any input. Nevertheless, the execution time for one calculation of a scene is at least half an hour. That execution time was not acceptable.

Consequently the implemented serial algorithm for processing the votes is too time intensive. As all votes can be processed independently of each other a parallel designed algorithm will yield shorter execution times. For a efficient parallel implementation the graphics card is a useful device. NVIDA CUDA is a parallel computing architecture that is only accessible on recent NVIDA graphics cards. The high number of processing units on the GPU allows parallel execution. The full computational power is accessible to the programmer and CUDA provides this access with a standard C interface. With the help of CUDA it is possible to reduce the execution time.

The advantage of parallel computing can be used at two points in the program. First for the calculation of the poisson distributed points and afterwards for the vote processing. Scenes that took over half an hour or even an hour to compute can be performed in less than second with the CUDA based implementation.. The details of the CUDA implementation are shown in the following section where the new algorithm is presented.

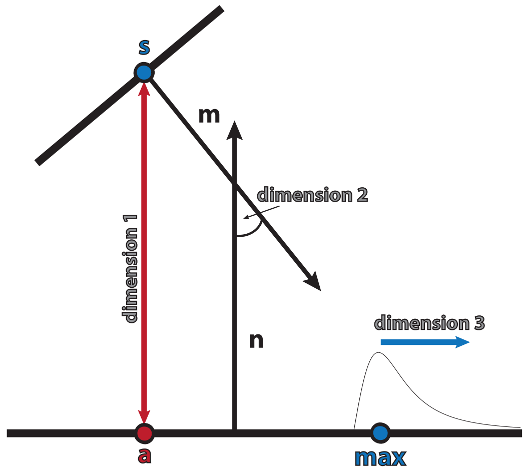


Figure 4.2: This figure illustrates the choice of the three dimensions of the precalculated voting table. Dimension one is the distance between voting point and plane to vote to along the planes normal. Dimension two represents the angle between the two respective normal vectors. The third dimension samples the ellipses of the form factor distribution produced by choices of dimension one and two in predefined steps.

4.2 The Inverse Radiosity Algorithm

In this section the implemented algorithm is presented. For this section basic CUDA knowledge is required. An explanation of memory allocation, memory freeing, and texture generation in the graphics card's memory is not provided. For an detailed information on how CUDA works, see the latest CUDA programmer's guide available on NVIDIA's homepage. Listing 4.1 shows the outline of the algorithm. Figure 4.3 shows a flow chart of our algorithm.

Listing 4.1: Basic Algorithm Structure

```
//PREPROCESSING
loadPrecalculations();
readInputScene();
calculatePlanes();

//POISSON DISK SAMPLING
for all faces f
        calculatePoissonPoints();

//VOTE PROCESSING
for all poisson points p
        for all planes k
                processVote();

//POSTPROCESSING
findAccumulationPoints();
```

The four main parts of the algorithm are: preprocessing, poisson disk sampling, vote processing, and postprocessing. In the following, each part of the algorithm is covered in a dedicated section. The important parts of each section are explained and in certain cases source code is provided.

4.2.1 Preprocessing

During start up of the program it is necessary to load all precalculations like the voting table and the random texture for the poisson disk sampling. This will lead to a longer start up time of 15 seconds, but the execution time of single calculation decreases by more than 99%. To decrease the lookup time in the precalculated voting table on the GPU the voting table is stored in three-dimensional textures. As a texture can only store up to four 32 bit floating point values per entry, two textures are required to store the entire voting table.

To calculate potential light source positions an input scene is required. After loading the input scene and the according illumination texture, several things need to be done before the actual computation can start. For further processing it is

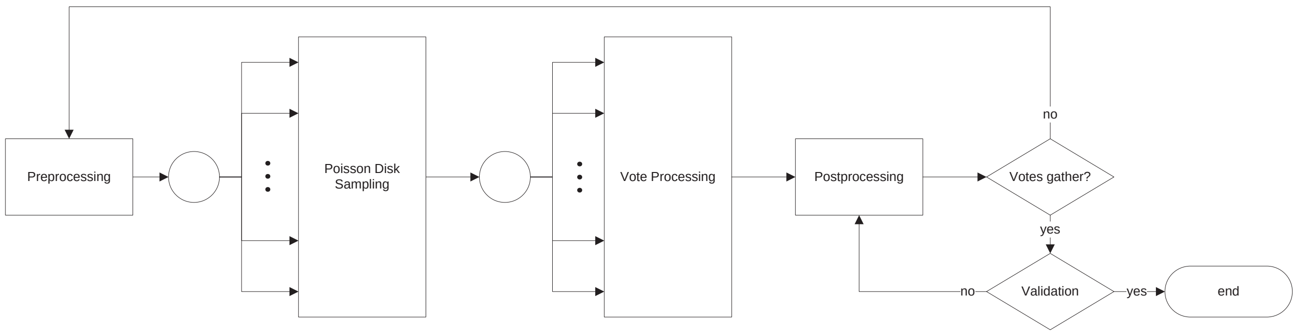


Figure 4.3: Flow chart of the Inverted Radiosity algorithm. The four main parts of the algorithm are: preprocessing, poisson disk sampling, vote processing, and postprocessing. The poisson disk sampling as well as the vote processing are processed parallel. Should the vote distributions do not contain any significant accumulation points then the whole calculation needs to be repeated with adapted settings. In the other case it is possible that the resulting light source positions are not extracted perfectly therefore the postprocessing step can be repeated with other settings.

necessary to extract arrays of all vertices and all faces as well as their normals. With the help of this information it is possible to calculate all different planes present in the scene. It is important that all calculated planes hold a list of all faces being a part of this plane.

For future weighting of the votes it is helpful to know what the brightest value in the inherent illumination of the scene is. The weights of the votes will be scaled according to the value of the brightest spots. A brightest spot is considered to have the overall biggest gray value in the illumination texture. To decrease the amount of planes to vote to and therefore the computational effort it is often convenient to calculate all planes visible to the brightest spots. For calculation of these visible planes it is necessary to find the three-dimensional coordinates that belong to the texture coordinates of the brightest spots. This is done with the help of barycentric coordinates. All triangles in texture space that represent faces in the scene's geometry are checked whether they include any of the brightest spots or not. There has to be exactly one triangle containing one brightest spot. The barycentric coordinates of a brightest spot based on the respective texture coordinates of the face's vertices can also be used to calculate the according three-dimensional position with the coordinates of the face's vertices. A plane is considered to be visible to a brightest spot if one vertex of one face is visible. In that context visible means that the angle between normal of the brightest spot and connection between brightest spot and vertex is less than $\frac{\pi}{2}$. Otherwise the vertex lies behind the plane and is therefore not visible. The position of the light source is always visible to the brightest spot, however, if more than one light source is contained in the solution this simplification might fail.

Before the computation of the set of potential light source positions several options can be set. All options available to the user will be discussed in the remaining part of this section.

- **Light Strength:** Brightness of the lamp for that potential positions are calculated.
- **Light Size:** Size of the lamp for that potential positions are calculated.
- **Resolution:** Resolution of the resulting vote distribution texture based on the size of the input illumination texture. The smaller the texture the more likely votes will gather on single pixels.
- **Vote Strength:** Basic weighting of each vote.
- **Votes per Square Meter:** This setting defines the amount of votes per square meter. The higher this setting is chosen the higher the computational effort becomes because more poisson disk samples are taken.
- **Weighting:** Possible vote weighting strategies are in respect to the illumination value or in respect to the voting distance.
- **Occlusion:** This setting defines whether occlusion should be processed or not. This setting increases the computational effort but avoids light sources, which lead to a wrong result.
- **Midair Vote:** This setting defines whether dummy planes should be used to process potential light source positions. With this setting light sources in midair can be realized. The amount and position of the dummy planes are predefined.

- **Vote to all Planes:** This setting defines whether votes should be cast to all planes or only to planes visible to the brightest spots. This setting increases the computational effort, but enables accumulation points to be located on planes not visible to the brightest spots. If only one light source is suspected to illuminate the scene it is suggested to turn this option off. Otherwise votes to all planes are necessary.
- **Threshold:** Threshold for the non-maxima suppression in the postprocession step. The higher the threshold the less local accumulation points are accepted. The range of possible values goes from 0 (all detected local maxima) to 254 (the likeliest positions).
- **Neighborhood:** Neighborhood radius of the non-maxima suppression step in the postprocession. The higher the radius the more local maxima are suppressed.

4.2.2 Poisson Disk Sampling

The poisson disk sampled points are calculated per face. To calculate the amount of sample points required for the whole scene the area of each face as well as the "votes per square meter" factor are necessary. For an efficient GPU calculation the *prefix sum* of the amount of voting points per face is a necessity because the number of points is defining the number of executed threads. Let the scene contain n triangles and let the entry i of the array pointAmount contain the amount of sample points of face i. The entries of pointAmount are based on the area of the respective triangles. The prefix sum pointsPerFace of pointAmount contains in entry i: $\sum_{k=0}^{i} \mathsf{pointAmount}[k]$. The last entry of the prefix sum consequently contains the sum of all poisson disk sample points of the scene. Henceforth the sum of all poisson points is called num_poisson.

The sampling is calculated using CUDA. For each poisson point a CUDA thread is started. The amount of blocks is computed by $\lceil \frac{\mathsf{num\_poisson}}{\mathsf{THREADS\_PRO\_BLOCK}} \rceil$, whereas THREADS_PRO_BLOCK is a constant defining the amount of threads per block[1]. The unique index of each thread calculated by $\mathsf{idx} = \mathsf{blockIdx.x} \cdot \mathsf{THREADS\_PRO\_BLOCK} + \mathsf{threadIdx.x}$ defines the number of the sample point that is calculated by this thread. The index falls in the range $(0, \mathsf{num\_poisson} - 1)$. Should the index be greater or equal to num_poisson no calculation is necessary. This can occur due to the calculation of the amount of blocks using the ceiling operation.

For the coordinate, normal and illumination computations of the sample point it is necessary to know to which face it belongs to. With the help of the prefix sum and the according index it is possible to pinpoint the face index of the face the point belongs to. An efficient way to do this is by using binary search within the prefix sum array. With the information of the according face it is possible to calculate the required data. The normal of the sample point is the normal of the located face. The coordinates of the sample point are calculated by a random lookup in the precalculated random texture containing the barycentric coordinates of the poisson disk sampling of a triangle. The coordinates are therefore calculated by $\lambda_1 \cdot A + \lambda_2 \cdot B + \lambda_3 \cdot C$ whereas λ_1 to λ_3 indicate the barycentric coordinates and A, B, and C indicate the coordinates of the face's vertices. The same barycentric coordinates can be used for calculating the position of the sample point in texture space. Therefore $\lambda_1 \cdot A_{tex} + \lambda_2 \cdot B_{tex} + \lambda_3 \cdot C_{tex}$ defines the texture coordinates of the sample point, whereas A_{tex}, B_{tex}, and C_{tex} are the respective texture coordinates of A, B, and C. The according illumination value of the poisson point is calculated by a lookup in the illumination texture.

The following listing shows pseudo code for the parallel computation of the poisson disk sampling.

Listing 4.2: Pseudo Code for Poisson Disk Sampling

```
calculatePoissonPoints(...)
{
  idx = blockIdx.x * THREADS_PRO_BLOCK + threadIdx.x;
  if ( idx >= num_poisson)
    return;

  currentFace = binarySearch(pointsPerFace, idx);

  //RANDOM LOOKUP
  [lambda1, lambda2, lambda3] = tex2D(poissonSample_tex, random);

  poissonPoints[idx].normal = currentFace.normal;
  poissonPoints[idx].coord = lambda1 * currentFace.A +
                             lambda2 * currentFace.B +
                             lambda3 * currentFace.C;
```

[1] for the efficiency of the GPU implementation a multiple of 32 is recommended

```

  texcoord = lambda1 * currentFace.A_tex +
             lambda2 * currentFace.B_tex +
             lambda3 * currentFace.C_tex;

  poissonPoints[idx].illumination = tex2D(illumination_tex, texcoord);
}
```

4.2.3 Vote Processing

After the calculation of all poisson disk sample points the votes can be cast. Each sample point has to vote to all planes of the scene, which has been calculated before. If the option 'use midair vote' is set dummy planes are processed too. It is necessary to distinguish between regular planes and dummy planes.

To allow for processing all votes efficiently once again a CUDA implementation is used. This time a block calculates the vote of one point to one specific plane. The fixed number of threads ELL_SAMPLES per block is required to separate the vote ellipse into distinct line segments. The resulting polygon of the ellipse with ELL_SAMPLES vertices is written to the resulting texture by all threads. Each thread writes one segment of the polygon. Therefore the number of blocks num_blocks is $\text{num\_poisson} \cdot (\text{num\_planes} + \text{num\_dummy})$ whereas num_poisson denotes the number of poisson sample points, num_planes the number of planes and num_dummy the amount of dummy planes. Because of the limitation of 2^{16} of one grid dimension a two dimensional block grid is necessary if a lot of sample points are used. The size of each dimension of the grid therefore is $\lceil\sqrt{\text{num\_blocks}}\rceil$. The unique index of each block is therefore calculated by $\text{idx} = \text{blockIdx.x} + \text{gridDim.x} \cdot \text{blockIdx.y}$. Due to the ceiling operator the execution of all blocks with index greater or equal to num_blocks needs to be rejected. Let us assume a setting of 100 points and 5 planes out of which 3 are regular planes and 2 are dummy planes. Table 4.1 illustrates the mapping of the unique block index to sample point and plane to vote to.

Index	Point	Plane	Type
0	0	0	regular
1	0	1	regular
2	0	2	regular
3	0	0	dummy
4	0	1	dummy
5	1	0	regular
6	1	1	regular
7	1	2	regular
8	1	0	dummy
⋮	⋮	⋮	⋮
497	99	2	regular
498	99	0	dummy
499	99	1	dummy

Table 4.1: Illustration of the mapping between the unique block index and according sample point and plane to vote to. A setting of 100 points and 5 planes out of which 3 are regular planes and 2 are dummy planes is assumed.

Consequently the sample point based on the unique index is calculated by $\lfloor\frac{\text{idx}}{\text{num\_planes}+\text{num\_dummy}}\rfloor$ and the plane index by $\text{idx} \bmod (\text{num\_planes} + \text{num\_dummy})$. If the derived plane index is greater than num_planes then it belongs to a dummy plane and the dummy plane index can be derived by subtracting num_planes from the previously derived plane index. The following listing of pseudo code shows the initialization step of the vote processing.

Listing 4.3: Pseudo Code for Vote Processing - Initialization

```
processVotes(...)
{
  idx = blockIdx.x +
        gridDim.x * blockIdx.y;

  total_planes = num_planes;
  if(processMidairVote)
    total_planes += num_dummy;
```

```

if(idx >= num_points * total_planes)
  return;

plane_idx = idx % total_planes;
point_idx = floor(idx/total_planes);
currentPoint = poissonPoints[point_idx];

if(plane_idx < num_planes) //REGULAR PLANE
  currentPlane = regularPlanes[plane_idx];
else if(processMidairVote) //DUMMY PLANE
  dummy_idx = plane_idx - num_planes;
  currentPlane = dummyPlanes[dummy_idx];
```

After identifying which vote needs to be processed in the current block the parameter of the deformed ellipse representing the vote can be retrieved. This is done via a lookup in the precalculated voting table. As mentioned before, to minimize the lookup time the voting table is split into two three-dimensional textures. But before the lookup it is necessary to perform declining tests because certain arrangements of voting point and plane to vote to do not allow casting a vote. Afterwards the indices of the required entry, based on the arrangement of voting point and plane to vote to, can be calculated.

The first declining test is a check whether the illumination of the voting point is 0 or not. A point with no illumination contains no information on any position of a light source therefore its vote can be suppressed. Should the normalized normal of the voting point and of the plane to vote to coincide, no vote is necessary because the according plane is not visible to the voting point. The same applies to an arrangement where the computed distance along the normal of the plane to the voting plane is negative or zero. In such a setting the voting point lies behind or on the according plane and no vote is possible because the plane is not visible to the voting point.

If the arrangement of voting point and plane satisfies the previously mentioned conditions and is therefore not declined the ellipse parameters can be fetched. However, it is still possible that the setting is declined because it is not contained in the voting table. Should the distance of the point to the plane be less than the value of the first sample[2] then it is set to the distance of the first sample. Based on the explanation in section 4.1.1.2 the according entries in the voting table are retrieved. First the indices based on the distance of point and plane and angle between their normals are computed. Based on these indices the maximum of the according form factor distribution can be determined. With the help of the value of the maximum and the normalized illumination value of the poisson point the index of the third dimension can be computed. Listing 4.4 shows pseudo code which retrieves the desired deformed ellipse parameters.

Listing 4.4: Pseudo Code for Vote Processing - Voting Table Lookup

```
A        = lightStrengh * lightSize;
angle    = arccos(dot(currentPoint.normal, currentPlane.normal));
distance = dot(currentPlane.normal, currentPoint.coordinate) + currentPlane.offset;

//DECLINING TESTS
if(currentPoint.illumination == 0                   ||
  currentPoint.normal        == currentPlane.normal ||
  distance                   <= 0)
  return;

if(distance < votingTable.distanceInc)
  distance = votingTable.distanceInc;

x = round((distance - votingTable.distanceInc) / votingTable.distanceInc);
y = round(angle / votingTable.angleInc) % (Pi / votingTable.angleInc);

if(y >= votingTable.DimensionY || x > votingTable.DimensionX)
  return;

maximumValue = tex3D(votingTable1, x, y, 0).maximum;

normalizedIllumination = currentPoint.illumination / A;
z = round((maximumValue - normalizedIllumination) / votingTable.distanceFromMaxInc);
```

[2]in the case of this implementation it is 10 centimeter

```

  if(z <= 0 || z >= votingTable.DimensionZ)
    return;

  //GET DEFORMED ELLIPSE PARAMETERS FROM VOTING TABLE
  table1Values = tex3D(votingTable1, x, y, z);
  table2Values = tex3D(votingTable2, x, y, z);
```

After retrieving the parameters of the deformed ellipse representing the vote, it is necessary to calculate sample points on the ellipse. The amount of sample points is based on the number of threads per block because each thread calculates one sample point depending on its index. For the calculation of these sample points other information is required beforehand, such as the minor and major axis direction and the position of the midpoint of the ellipse. First the minor and major axis directions are calculated as explained in equation 3.25. In the case of circles, where the normal of the point is the negative of the normal of the plane, any direction lying in the plane can serve as major axis direction. The next step is calculating the position of the distribution's maximum. A rotation matrix is calculated based on the major axis direction[3] and the angle stored in the voting table. Rotating the normal of the voting point leads to the direction which in turn leads to the position of the maximum. With the position of the maximum the position of the midpoint can simply be calculated by $\text{maximum} + \text{x\_m} \cdot \text{minor} + \text{y\_m} \cdot \text{major}$ whereas x_m and y_m are the offsets stored in the voting table. Based on the thread index a sample point of the ellipse can now be calculated. To provide all threads of the current block with the information on the ellipse sample points, an array of these sample points is allocated in shared memory. A sample point is calculated using the explicit representation of the deformed ellipse (see equation 3.20). The angle used for calculation depends on the thread's index. After calculating the sample points it is necessary to synchronize the threads because in the upcoming calculations these sample points are required. The pseudo code in listing 4.5 shows the calculation of these ellipse sample points.

Listing 4.5: Pseudo Code for Vote Processing - Sampling of the Ellipse

```
  if(currentPoint.normal == -currentPlane.normal) //CIRCLE CASE
    //CALCULATE ANY MAJOR AXIS
    ...
  else //REGULAR CASE
    major = cross(currentPlane.normal, currentPoint.normal);

  minor = cross(major, currentPlane.normal);

  //ROTATE NORMAL TO GET DIRECTION IN WHICH THE MAXIMUM LIES
  rotMatrix = getRotationMatrix(major, table1Values.angle)
  max_direction = rotMatrix * currentPoint.normal;
  maximum = currentPlane.rayIntersect(currentPoint.coordinate, max_direction);

  //ELLIPSE PARAMETERS
  a   = table1Values.a;
  b   = table2Values.b;
  d   = table2Values.d;
  x_m = table2Values.x_m;
  y_m = table2Values.y_m;
  midpoint = maximum + x_m * minor + y_m * major;

  //SHARED ARRAY FOR ELLIPSE SAMPLE POINTS; EACH THREAD CALCULATES ONE.
  __shared__ samplePoints[ELL_SAMPLES];

  //THREAD INDEX BASED CALCULATION
  ell_angle = threadIdx.x * 2 * Pi / ELL_SAMPLES;
  samplePoints[threadIdx.x].coordinate = midpoint
                                       + a * sin(ell_angle)                        * major
                                       + b * (cos(ell_angle) + d * cos(2 * ell_angle)) * minor;

  __syncthreads();
```

It finally needs to be assured that the ellipse sample points lie on an existing face and are not occluded. The test for occlusion is only processed if the according option is set. As it is already known on which plane the sample points are

[3]the major axis direction is the rotation axis.

located, not all faces need to be checked. For all faces of the according plane a test via barycentric coordinates is carried out. If the barycentric coordinates state that the point lies inside the face then the texture coordinates of the point can be computed with the same barycentric coordinates. If the point is not located on a face the occlusion test is not necessary because no texture coordinate belongs to this point. Dummy planes do not contain any faces and therefore they need to be treated differently. The quad resembling the dummy plane is simply divided into two triangles. The test via barycentric coordinates can be processed within these two triangles. As the votes to dummy planes are written to another texture the texture coordinates $(0,0)$, $(0,1)$, $(1,0)$, and $(1,1)$ are assigned to the four vertices of the dummy plane.

The occlusion test is processed with all planes except the one the point is located on.[4] The connection between voting point and ellipse sample point is intersected with all other planes. If the intersection point lies between voting point and ellipse sample point and is within a face of the according plane, the point is occluded. The test whether the intersection point lies within a face or not is also done via barycentric coordinates. If no intersection point in the space between voting point and ellipse sample point is located on a face the point is not occluded.

After acquiring this information for the sample points the according polygonal line can be drawn to the result textures. Each thread draws the line from its current point to the next point in the samplePoints array. If either of these two points is occluded or not on a plane the whole line is not drawn. This will cause that ellipses, which are drawn to faces that are located on an edge, are not drawn until the edge of the according face. With a sufficiently high ellipse point sample density, however, this is not an issue. The lines are drawn with the Bresenham algorithm [Bre65]. Votes drawn to the dummy plane texture are offset based on the dummy plane index so that the areas resembling a dummy plane are placed side by side in the resulting texture.

It is important that no pixel is drawn twice therefore the starting point of each line is not drawn. Furthermore it is important to mention that votes can be lost if all blocks are writing simultaneously into the textures. Since CUDA compute capability 1.2 it is possible to lock single variables. The according function is called atomicAdd and is necessary to obtain correct results. Listing 4.6 shows the final piece of pseudo code for drawing to the result texture.

Listing 4.6: Pseudo Code for Vote Processing - Writing to the Result Texture

```
  samplePoints[threadIdx.x].isOnPlane = false;
  samplePoints[threadIdx.x].isOccluded = false;

  //CHECK WHETHER SAMPLE POINT IS ON A FACE OR NOT
  for all faces f of currentPlane //IN CASE OF DUMMY PLANE THERE ARE ONLY TWO
    [lambda1, lambda2, lambda3] = getBarycentricCoords(f, samplePoints[threadIdx.x].coordinate);
    if(isInside(lambda1, lambda2, lambda3))
      samplePoints[threadIdx.x].isOnPlane = true;
      samplePoints[threadIdx.x].texCoordinate = lambda1 * f.A_tex +
                                                lambda2 * f.B_tex +
                                                lambda3 * f.C_tex;
      break;

  //CHECK WHETHER SAMPLE POINT IS OCCLUDED OR NOT
  if(samplePoints[threadIdx.x].isOnPlane && processOcclusion)
    vector = currentPoint.coordinate − samplePoints[threadIdx.x].coordinate;
    for all planes p //NO DUMMY PLANES
      if(p != currentPlane)
        intersectionPoint = p.rayIntersect(samplePoint.coordinate, vector);
        if intersectionPoint is between the coordinates of currentPoint & samplePoints[threadIdx.x]
          for all faces f of p
            [lambda1, lambda2, lambda3] = getBarycentricCoords(f, intersectionPoint);
            if(isInside(lambda1, lambda2, lambda3))
              samplePoints[threadIdx.x].isOccluded = true;
              break;
          if(samplePoints[threadIdx.x].isOccluded) //FOUND ONE OCCLUDING PLANE, THAT IS ENOUGH
            break;

  __syncthreads();

  if(samplePoints[threadIdx.x].isOnPlane && !samplePoints[threadIdx.x].isOccluded)
    //DRAW LINE TO ACCORDING TEXTURE
    ...
}
```

[4] dummy planes are not included in this test because they are no occluders.

A problem occurring with CUDA is that a kernel execution is canceled after a timeout. CUDA stops the execution of the kernel call if it takes more than five seconds. In other words, a high density of votes in a rather complex scene is not possible because of the large amount of computation time required. Therefore, it is necessary to start the kernel several times with smaller workload. To guarantee that all votes are processed the calculated index of the block must be unique throughout all kernel calls. Therefore an offset for the index is necessary: idx = offset + blockIdx.x + gridDim.x * blockIdx.y; where offset is a variable containing the number of processed blocks in the previous kernel executions.

This concludes the section about the vote processing. The next section deals with the processing of the resulting textures.

4.2.4 Postprocessing

The postprocessing step deals with finding accumulation points within the resulting textures. One texture is generated for the scene and one additional texture is generated for the dummy planes if the according setting is chosen. For visualizing the resulting texture the dynamic range needs to be adapted to the maximum grey value present in both textures. It is important that the same scaling is applied to both textures to keep the relation between the pixel values of both textures. If only the regular texture is computed a simple dynamic range adaption is sufficient. As the resulting textures look similar to Hough transforms a technique for peak detection in Hough transformed images can be applied.

The first step for the peak detection is applying a convolution based on a Laplacian kernel. Applying a Laplacian kernel is enhancing local maxima in the underlying image. A 3×3 Laplacian kernel is sufficient for our purposes. Afterwards a simple non-maxima suppression is applied. If a pixel has a neighboring pixel with a higher value than itself than its value is suppressed[5]. The neighborhood for the non-maxima suppression is chosen by the user in the preprocessing step. To keep the relation between the maximum values of the two resulting textures, each pixel value is afterwards scaled with the according value of the texture gained after applying the dynamic range conversion.

The last step is applying the threshold filter with the user-defined threshold. Each pixel with a value greater than the threshold is considered to be a possible light source position. The explained technique guarantees that the most likely light source positions correspond to pixels with value 255. Therefore a threshold of 254 leads just to the most likely light source positions.

The last step of the presented algorithm is to find the three-dimensional coordinates corresponding to the pixels indicating a light source position. The applied technique is the same that is used for finding the three-dimensional coordinates for the brightest spots in the illumination texture. For the regular texture all triangles in texture space correspond to faces of the geometry are checked whether they contain one of the resulting pixels or not. With the according barycentric coordinates the three-dimensional position can be retrieved. As the dummy planes are containing just two triangles finding the according three-dimensional position is easy because it is contained in one of these triangles.

With the three-dimensional coordinates of the calculated light source positions a solution to the ambiguous inverse lighting problem is found. A tutorial how to use the software developed with this thesis is presented in the next section. This tutorial is accompanied by several examples.

4.3 Examples and Tutorial

This section provides a tutorial for the program that has been developed with this thesis. Required software, hardware and libraries are:

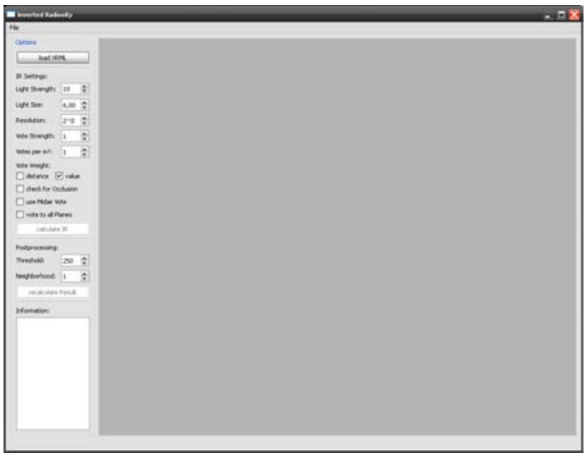
- QT version ≥ 4.5, [QT]
- Coin3D version > 3.0, [Coi]
- Quarter version 1.0.1, [Qua]
- CUDA version ≥ 2.3, [CUD]
- CUDA templates, [CTe]
- CUDA enabled NVIDA graphics card; compute capability ≥ 1.2 recommended

After starting a splash screen is displayed while all precalculations are loaded. The splash screen looks like the screenshot in figure 4.4(a). After the start up the program looks like the screenshot in figure 4.4(b).

All options listed on the left side of the user interface are discussed in section 4.2.1. To load a VRML file the according button on the left side needs to be clicked. The software automatically tests if the file is structured the way it needs to

[5]set to 0

(a)



IR Settings:
Light Strength: 10
Light Size: 6,00
Resolution: 2^8
Vote Strength: 1
Votes per m²: 1
Vote Weight:
☐ distance ☑ value
☐ check for Occlusion
☐ use Median Vote
☐ vote to all Planes
Postprocessing:
Threshold: 250
Neighborhood: 1
Information:

(b)

Figure 4.4: Figure (a) shows the splash screen that is visible on the screen during start up. While this screen is visible all precalculations are loaded. Figure (b) shows a screenshot of the developed program after start up. All options available on the left side are discussed in section 4.2.1.

be. If the geometry is not only composed out of triangles, or if the according illumination texture file is missing or is not related to the geometry then an according error message is shown. If the file is accepted the "Calculate IR" button becomes active and an input window showing the textured geometry contained in the file is displayed. Furthermore information related to the input file is displayed in the "Information" text field such as value of the brightest spot, number of calculated planes, and number of visible planes to the brightest spots.

After loading an input file the options on the left side need to be specified. With appropriate options a click on the "Calculate IR" button produces the according output window. The output window contains the same textured scene as the input window but red spheres with radius based on the chosen light source size are placed at the calculated light source positions.

The generated voting textures are saved in the folder containing the executable file. In the case the calculated light source positions are unnatural or inappropriate, the generated textures need to be examined. If no real accumulation points are present the light source parameters have to be changed. As already discussed in the previous theory chapter, increasing the light source parameters also increase the ellipses. Consequently, decreasing the light source parameters also decreases the ellipses. Increasing the density of votes might also be an aid in achieving accumulation points, as more different illuminated points are voting.

In case the resulting vote distribution is too ambiguous changing the weighting of the votes might help. Turning off both weight options leads to a uniform weight for all votes, this might lead to undesired accumulation points. If occluders are present in the scene turning on the occlusion check is also a possibility to decrease the ambiguity.

After finding proper light source parameters a little fine tuning is necessary to extract single light source positions from light source position cluster. In most cases such light source position cluster are sufficient because they indicate that a light source has to be placed in the middle of it. If an extracted light source position is required, however, changes to the other settings are necessary. A way to achieve an extracted position is to increase the neighborhood of the non-maxima suppression because more pixel values get suppressed if the neighborhood radius increases. Another way is decreasing the resolution of the resulting textures. It is more common for ellipses to intersect if the pixel resolution is smaller. Extracting one single light source per cluster is not always possible respectively it is hard to find the correct setting for achieving this. Figure 4.5 shows a cluster of possible light source positions in the first picture and the extracted light source position in the second one.

To allow more light source positions, for example in a case where more light sources are to be expected, a change in the threshold property is necessary. A lower threshold will allow more local maxima of the voting texture to be interpreted as possible light sources. Decreasing the threshold will often lead to a clustering of light source positions at the most likely positions.

Changes to the threshold as well as to the neighborhood property can be done without recalculating the vote distribution. A click on the button with the name "recalculate Result" in the postprocessing section of the options interprets the latest calculated result with the changed threshold and neighborhood parameters. By invoking the Inverted Radiosity calculation the chosen threshold and neighborhood settings are considered as well. Consequently the "recalculate Result" is only available after calculating the Inverted Radiosity once for a newly loaded input scene.

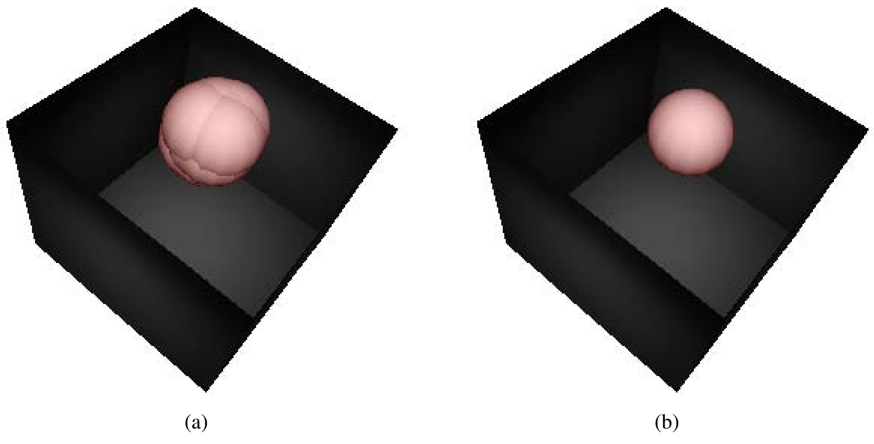


Figure 4.5: Figure (a) shows a cluster of possible light source positions. That cluster indicates that a light source has to be placed in the midst of this cluster. Figure (b) shows the extracted light source position.

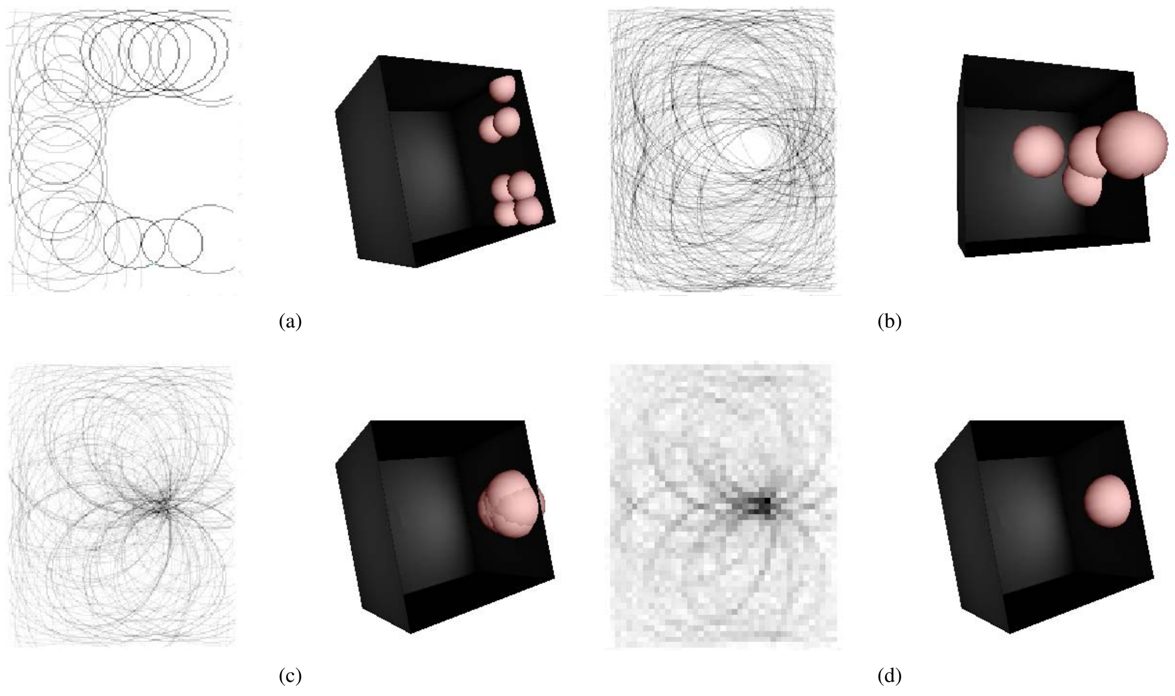
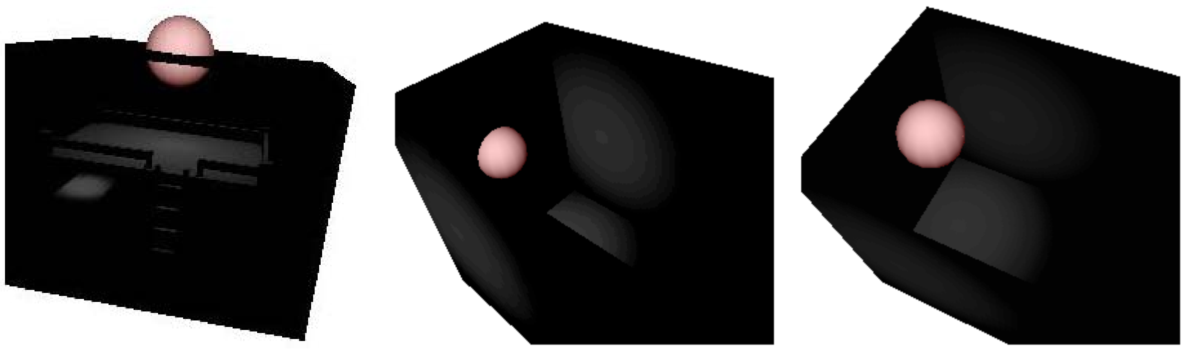


Figure 4.6: This figure shows images according to the first example presented in this tutorial. The accompanying vote distribution images are always the vote distribution of the plane where the light source is placed in the end. Figure (a) shows the result of the initial guess of light source parameters. The parameters are $2m^2$ for the light source size and 10 for the brightness. The according vote distribution indicates that the light source size or strength is chosen too small because the ellipses do not intersect in one common point. The second image (b) shows the situation where the light source size is chosen too big ($8m^2$). The ellipses intersect in many points but not in one single point and are therefore too big. The right guess of a light source size of $6m^2$ is shown in figure (c). The ellipses intersect in one common point, but the resulting light source positions are clustered. To extract one single light source the texture resolution is decreased. The result of this extraction is shown in the last image (d).

The remaining part of this section provides examples. Figure 4.6 shows the images to the first example. The accompanying vote distribution images are always the vote distribution of the plane where the light source is placed in the end. Let us assume we have already loaded an input scene and are about to give our initial guess for the light source parameters. Figure 4.6(a) shows the result for a chosen light source size of $2m^2$ and brightness of 10. The according vote distribution shows a set of deformed ellipses that do not intersect in a common point. Those ellipses are too small to intersect in one common point, therefore the light source parameters need to be increased. The next guess is with an increased light source size of $8m^2$. Figure 4.6(b) shows the according result. This time the deformed ellipses are too big. They intersect in many points but not in one common point. Consequently the right light source size has to lie between $2m^2$ and $8m^2$. Figure 4.6(c) shows the result for a light source size of $6m^2$. The ellipses intersect in one common point, but the result shows a cluster of light source positions. To extract one single light source the threshold is set to 254 and the resolution of the resulting texture is decreased. The local maximum in the vote distribution gets enhanced by decreasing the texture resolution because now almost all ellipse intersect in one single pixel. The result of the light source extraction is shown in the last sub-figure 4.6(d).

The example shows results for scenes where the target illumination is complete. The following examples demonstrate cases where the illumination is actually painted onto the scene's surface. It is difficult to imitate a real illumination by manually painting it onto surfaces. It is necessary to paint bright spots onto parts that need to be illuminated. Figure 4.7 shows results for two scenes. A bright spot formed like concentric circles on a surface implies that a light source is placed somewhere above it. One example for this is seen on the bed in figure 4.7(a). Since the bright spot is also illuminated by two further spots at the adjacent walls, a light source on a third adjacent wall is also a possible solution as seen in figure 4.7(b). Nevertheless a light source directly above the bright spot on the floor is also a possible solution and will be achieved with other settings of the light source (see figure 4.7(c)).



(a) (b) (c)

Figure 4.7: Two examples for incomplete target illuminations. Figure (a) shows an example where the illumination painted on the bed's surface implies a light source directly above it. The second image (b) shows an example where a light source position on a wall is strengthened. However, a light source placed above the spot on the floor in midair is also possible as seen in the third figure.

This ends this tutorial further examples are shown in the next chapter.

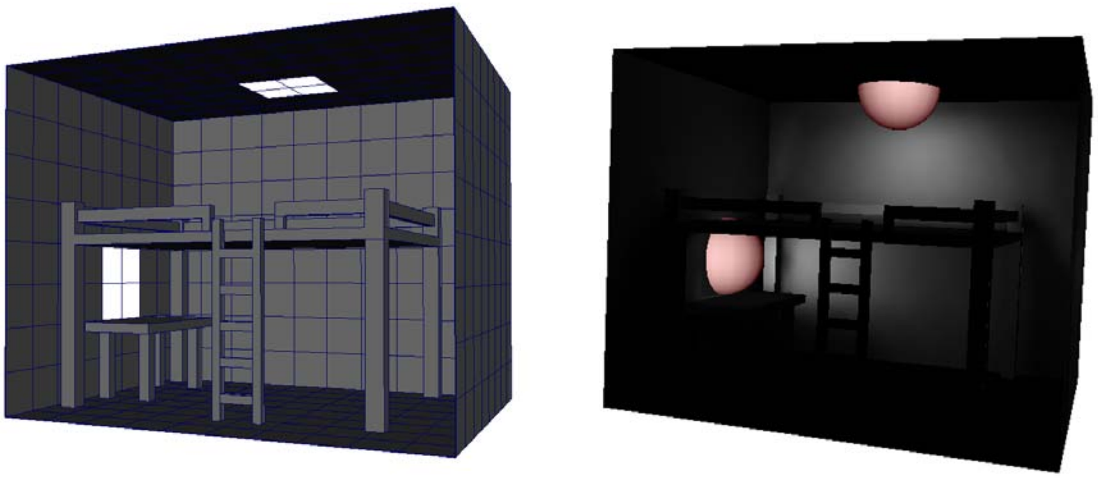
Chapter 5

Results and Discussion

This chapter deals with the discussion of benchmark tests that show the integrity of our method. Furthermore strengths as well as limitations of the proposed method are discussed.

5.1 Benchmarks

The best way to prove the proposed algorithm works correctly is to calculate light source positions known a priori. Consequently the best benchmarks are scenes that are completely illuminated by a set of light sources of which the parameters are known. The following examples show scenes where light sources of same size and brightness are placed in the scene. Due to the same size and brightness of the light sources all positions can be calculated at once. With known light source parameters the settings for the respective options are given. All following figures show the according geometry with patches set as light sources and the result of the light source position calculation. Figure 5.1 shows a complex scene that is inspired by a test scene of [M.02]. Both light source position can be extracted at their correct positions. Figure 5.2 shows that extraction is not always possible when more than one light source illuminates the scene, but the result shows clearly where the light sources need to be placed. Figure 5.3 shows a test scene where angles other than 0 and $\frac{\pi}{2}$ occur.

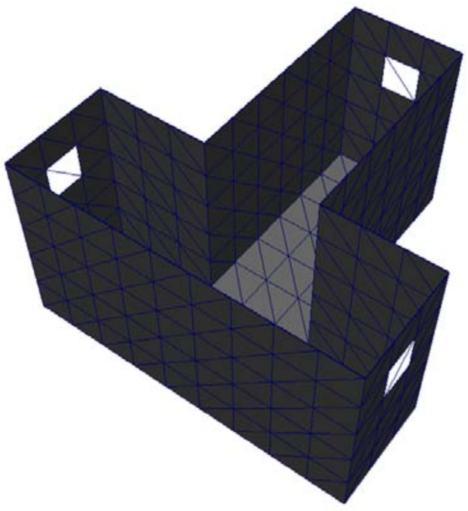


(a) (b)

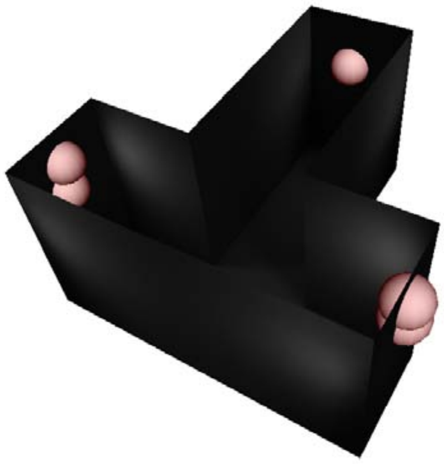
Figure 5.1: A complex test scene inspired by a scene from the paper [M.02]. This scene is completely illuminated by the light sources shown in figure (a). Both light sources can be extracted at their correct positions as seen in figure (b).

5.2 Discussion: Strengths and Limitations

The proposed method is still not perfect. There are still situations, where it is very difficult to get a clear result. The main problem resides within overlapping illuminations of different light sources. Even full illuminations with known light source parameters cause difficulties when the illuminations of the single light sources overlap too much. The main

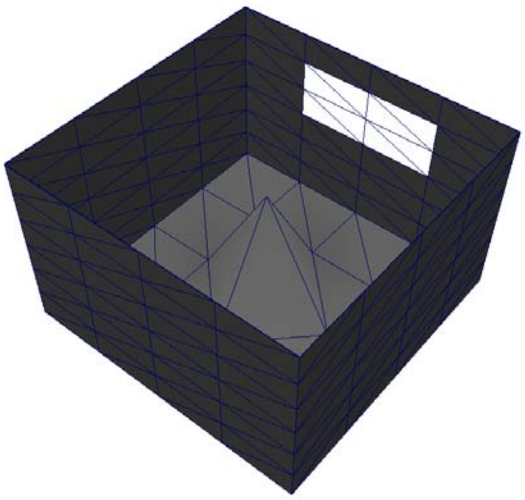


(a)

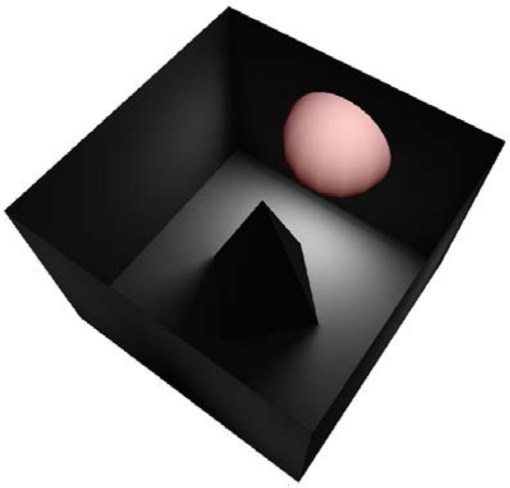


(b)

Figure 5.2: A test scene to illustrate that light source positions cannot always be extracted when more than one light source is present in the scene. Figure (a) shows the positions of the light sources that illuminate the scene and figure (b) shows the calculated result.



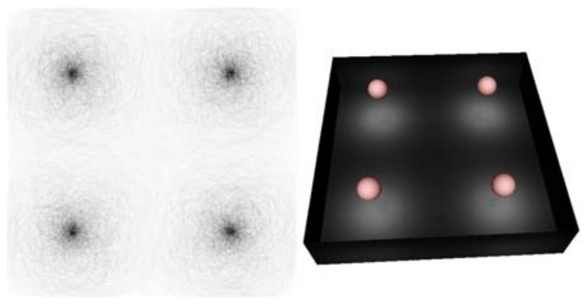
(a)

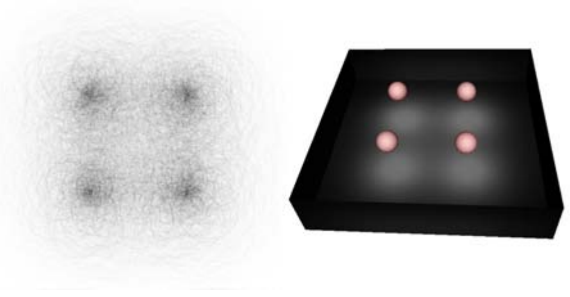


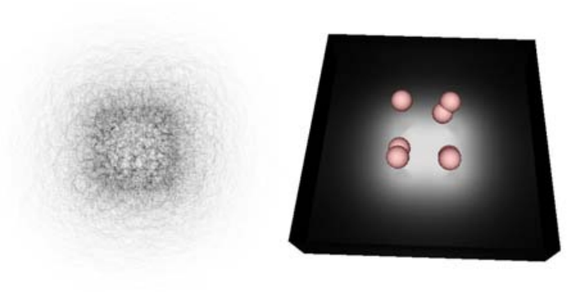
(b)

Figure 5.3: A test scene to demonstrate other arrangements of voting point and planes to vote to. In this test scene other angles than 0 or $\frac{\pi}{2}$ occur. As before figure (a) shows the geometry with the defined light source and figure (b) shows the light source position that is calculated with the illumination produced by the light source in (a) as input.

problem lies within the fact that for a single illuminated point the amount of light provided by each light source is not known. Even assumptions for these distributions are meaningless because the distribution is different at each illuminated point. Our method relies on the points that receive most of the illumination from one light source. Figure 5.4 illustrates that problem and a workaround. Each sub-figure shows the vote distribution with the according result. In this example the four light sources move towards each other and the illuminated regions overlap. In the first sub-figure 5.4(a) the illuminated regions do not overlap and the result is clear. The according vote distribution also shows four clear accumulation points. In the second case in figure 5.4(b) the regions overlap a bit, even though the result is clear. Again, the vote distribution shows four clear accumulations points. In the third figure 5.4(c) the problems begin to occur. The vote distribution does not show any clear accumulation points, but more a contour line with approximately same voting value. Consequently the positions of the four light sources are hard to determine. Nevertheless the fourth sub-figure 5.4(d) shows a workaround. Instead of trying to fit four separate light sources that are close to each other it is better to find a position of a single light source that is four times greater than the original ones. The voting distribution of figure 5.4(d) shows a clear accumulation point and therefore the position of the bigger light source is determined.


(a)


(b)


(c)

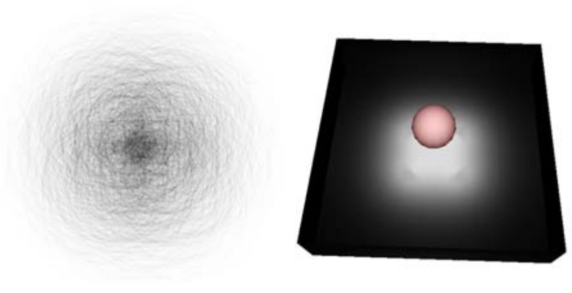

(d)

Figure 5.4: This figure illustrates the problem with overlapping illuminations of different light sources. Each sub-figure shows the vote distribution with the according result. In this example the four light sources move towards each other and the illuminated regions overlap. In the first sub-figure (a) the illuminated regions do not overlap and the result is clear. The according vote distribution also shows four clear accumulation points. In the second case in figure (b) the regions overlap a bit, even though the result is clear. Again the vote distribution shows four clear accumulations points. In the third figure (c) the problems start. The vote distribution does not show any clear accumulation points, but more a contour line with approximately same voting value. Consequently the positions of the four light sources are hard to determine. Nevertheless the fourth sub-figure (d) shows a workaround. Instead of trying to fit four separate light sources that are close to each other it is better to find a position of a single light source that is four times greater than the original ones. The voting distribution of figure (d) shows a clear accumulation point and therefore the position of the bigger light source is determined.

Another example of the problem occurring with overlapping illuminations is shown in figure 5.5. The scene has been illuminated by two light sources of the same size. One of these light sources is placed in the middle of the ceiling and another on a wall. Let us suppose we know the exact size and brightness of the placed light sources. Computing the vote distributions with these settings will not yield in a desirable result. This configuration will not even lead to accumulation points at the positions where the light sources have been located before. The resulting distribution on the wall indicates that the parameters are chosen too small. The result is a position somewhere between the original positions as seen in figure 5.5(a). This position produces an illumination close to the target illumination, but the original light source positions are not found. This is due to the overlapping illumination.

Our approach searches for a single position that will illuminate the scene as it is given. This is because all points vote with their illumination value and no distinction between light sources is made. It is not possible for a single light source of the known size and brightness to illuminate the scene in such a way. Due to two light sources with overlapping illuminations placed in the scene, the illuminated spot is twice as bright. Therefore if a single light source should achieve this target illumination it must also be twice as bright or twice as big as the original ones. A setting with the doubled parameters retrieves the original positions approximately as shown in figure 5.5(b). This behavior is due to the fact that our method calculates one single light source achieving the target effect. If the two light sources with the doubled parameters are placed on those two positions the illumination will be too bright. As both light sources illuminate the same spot, their brightness or size needs to be halved to obtain the target illumination. So basically if we search for two light sources of size X illuminating the same spot, it is necessary to search for two positions of light sources with size $2X$ that approximately reach the target effect. Both light sources can then be places with size X at the positions calculated before. This problem does not occur if the illuminated spots do not completely overlap.

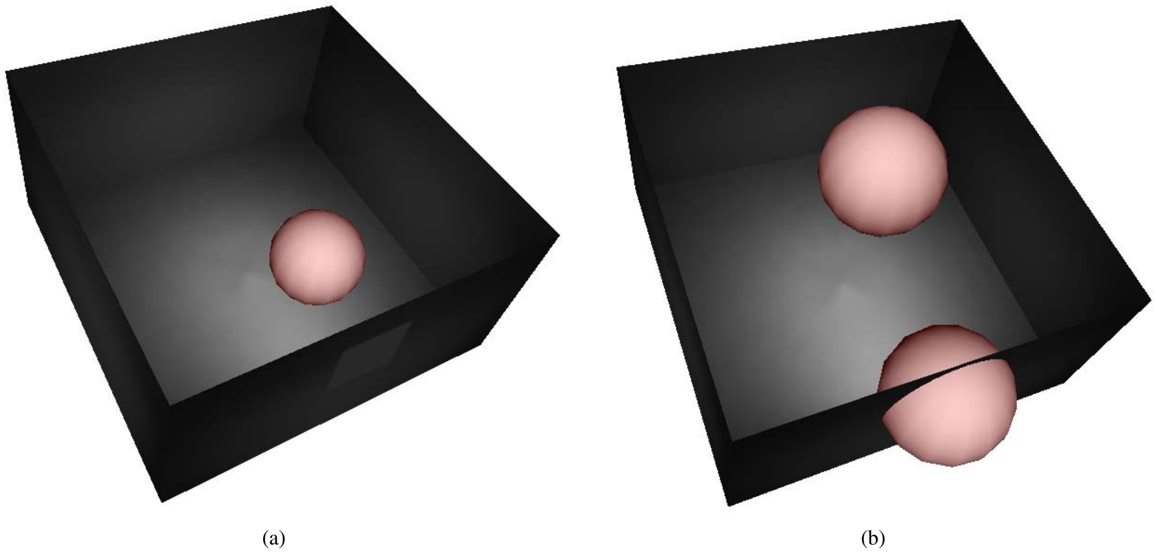


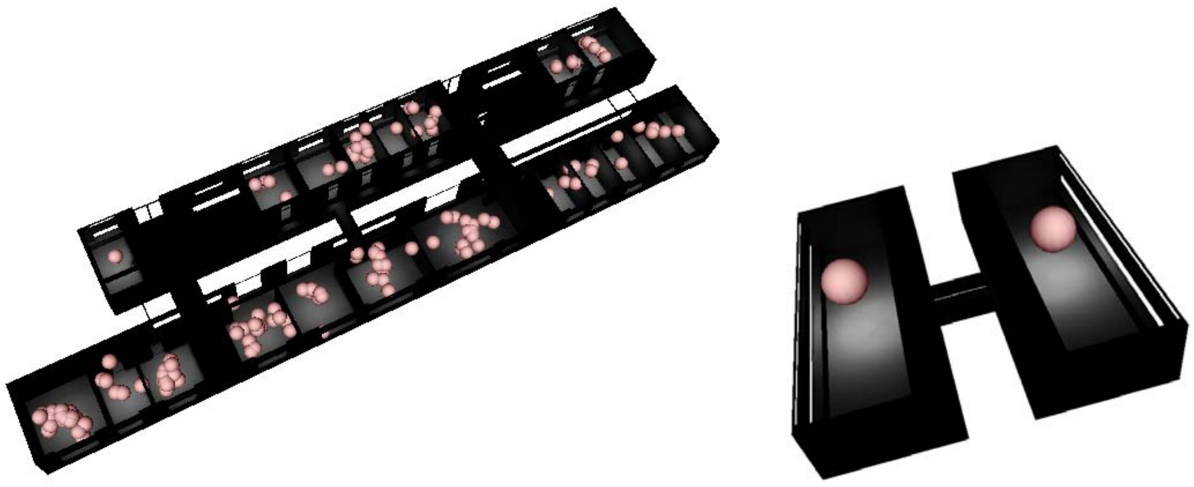
Figure 5.5: This figure illustrates problems of overlapping illuminations. This scene has been illuminated by two light sources of the same size. One of these light sources is placed in the middle of the ceiling and another on a wall. Let us suppose we know the exact size and brightness of the placed light sources. Sub-figure (a) shows the result of a computation with these parameters. This position produces an illumination close to the target illumination, but the original light source positions are not found. Our approach searches for a single position that will illuminate the scene as it is given. This is because all points vote with their illumination value and no distinction between light sources is made. Due to two light sources with overlapping illuminations are placed in the scene the illuminated spot is twice as bright. Therefore if a single light source should achieve this target illumination it must also be twice as bright or twice as big as the original ones. With a setting with the doubled parameters the original positions can approximately be retrieved as shown in figure (b). If the two light sources with the doubled parameters are placed on those two positions the illumination will be too bright. As both light sources illuminate the same spot their brightness or size needs to be halved to obtain the target illumination.

Other problems arise with the use of incomplete illumination setups as input. The problem is rooted in the fact that in real world scenarios light sources of different sizes illuminate a scene. Furthermore it is hardly possible that an illumination setup, which is painted on a scene can guarantee that light sources of same size can achieve this lighting. Therefore several independent calculations with different light source settings are necessary to identify all light source positions.

Nevertheless, we also want to highlight the advantages of the approach. The most important feature is certainly the ability to calculate light source positions in midair. Until now it was necessary to provide geometry. However, with provided geometry the number of possible light source positions is strongly limited. Furthermore the algorithm is independent of the tessellation of the geometry. The short execution time of one single computation is an advantage. Due to this short execution time fine tuning of the settings becomes possible. Thus, our approach can be used to guide the designer to a desired result.

5.2.1 Out Takes

Large scenes cannot be processed very well. Figure 5.6(a) shows an example for such a large scene. This scene resembles a floor of a building located at the Graz University of Technology. Each office on that floor has been illuminated by one light source placed at the ceiling. All placed light sources have the same size and brightness. The problem lies within the resulting vote distribution. All accumulation points have different values and extracting all of them is impossible in such large scenes. By lowering the threshold a lot of light source position cluster are produced. This diminishes the significance of the result. Another problem with such larges scenes is the amount of memory required on the graphics card that is required for the execution.



(a) (b)

Figure 5.6: Figure (a) shows a 3D model of a floor of a building located at the Graz University of Technology. Processing the whole floor leads to a lot of light source position cluster because the accumulation points in the result textures do not share the same value. To generate meaningful results, such large models need to be split into smaller parts that can be processed separately. For models of buildings this is often the case because rooms are in general illuminated without taking other rooms of the building into account. Figure (a) demonstrates that smaller parts of the building can be processed without any problems.

However, such scenes can be split into smaller parts as most of the illuminations are independent of each other. The algorithm works for smaller sets of offices without a problem as demonstrated in figure 5.6(b). Consequently a division into smaller parts is necessary to produce meaningful results. In real life buildings each room is illuminated separately, therefore each room can be processed without taking the other rooms into account. Our algorithm would produce the right result if each office of the model in figure 5.6(a) is processed separately.

Chapter 6

Conclusion and Future Work

This last chapter is the concluding chapter. First it gives an outlook for possible places of action. At the end of this chapter important future work is listed.

6.1 Conclusion

A method for solving ambiguous inverse lighting problems is presented in this thesis. To calculate a set of potential light source positions from a user-defined target lighting it is necessary to provide information about the light source that is to be placed. The results calculated by our method can be used in various situations. The main place of action will certainly be in architecture. The results should provide help for architects that design indoor environments. The positioning of lamps can be automatically calculated just by defining a target illumination. The space-light co-design will also profit from our application. If the desired lighting is for example not possible, perhaps a rearrangement of the indoor equipment will resolve the problem. Architecture, however, is not the only field of application. Another possible use of our program would be in computer games. Desired lighting effects to hide or highlight special items or locations can also be achieved with the help of our program.

Another interesting application would be retrieving light source positions from photographs of an illuminated room. With the help of photographs it is possible to build a 3D model of the room. With a separate set of photographs for which light source positions are known, it is possible to retrieve the material constants for all surfaces in the room. To calculate these material properties the method presented in [YDMH99] can be used. With the geometry and the material properties the light source position can be calculate for any set of photographs.

For the use in the above mentioned environments a lot of work is still necessary. Future work is listed in the following section.

6.2 Future Work

Automation The need of an automatic method is obvious. However, an interpretation method for the resulting vote distributions is necessary. One possible method would be a machine learning approach where the computer interprets the distributions on its own. Another method would be minimizing an error function that calculates the error between target illumination and illumination produced by the calculated light sources. A higher order evolution strategy could be used for minimizing this error function. It is possible that the user gives hints to accelerate the optimization because it is much easier for the user to interpret the resulting distributions than for the computer. Furthermore, the method to handle overlapping illuminations needs to be improved. A solution for the problem with light sources of different parameters needs to be found as well. The automatic method should also be able to return light source positions belonging to light sources with different size and brightness settings as result.

Non-Lambertian Materials Assuming that all surfaces are lambertian is rather restrictive. Therefore a need to process reflecting as well as refracting surfaces is present. As these surfaces cannot be light sources, votes onto these surfaces need to be processed differently. A vote to a reflecting surface is either rejected or it produces another vote to determine the light source that is illuminating that mirroring spot. Refracting surfaces need to be processed in a similar way. Furthermore, a decrease in vote strength for each reflection or refraction is necessary. A maximum number of reflections and refractions needs to be defined as well.

Independence of Maya To get independent from Autodesk Maya [May] an interface for light painting is necessary. It should provide an easy way for producing and reconfiguring the target lighting. For this the interface needs to provide specific brushes to imitate real light distributions. The target illumination could then be adapted in the developed program.

Furthermore an option to paint shade onto surfaces is necessary. Shaded points nullify all votes to the faces they belong to as well as all votes to all faces visible to those shaded points. This is due to the fact that no light sources and therefore no light source positions are visible to not illuminated points.

Moreover an option to calculate forwards radiosity with the calculated light source positions is necessary to provide confirmation of the target illumination and visualization of the according side effects, such as illumination on other spots. The user is then able to switch between forwards and backwards radiosity calculation to achieve the desired results. Another goal would also be a support for different input formats.

Additional Dummy Plane Features and Placement Techniques Another feature the improved user interface should provide is the option to manually place dummy planes throughout the scene. With manually placed dummy places light sources can be processed everywhere the user wants to. Additionally the user is also able to interactively move the dummy plane for achieving a single accumulation point. Furthermore, options for computer-aided placement of dummy planes are necessary to decrease the amount of work for the user. Such an option should be for example: "layer X dummy planes with distance Y to each other from the ceiling". This kind of option should be available for all sides of an oriented bounding box of the scene. Until now dummy planes can only receive votes from points lying in front of the plane. A different kind of dummy plane that accepts votes from both sides is also part of the future work. With such dummy planes vote distributions that are not close to any boundary of the scene become more realistic.

Bibliography

[Bre65] BRESENHAM J.: Algorithm for computer control of a digital plotter. *IBM Systems Journal 4*, 1 (1965), 25–30. 34

[Coi] Coin3D. a high-level, retained-mode toolkit for effective 3D graphics development, www.coin3d.org/. 25, 35

[CTe] CUDA templates. a collection of C++ template classes and functions which provide a consistent interface to CUDA, cudatemplates.sourceforge.net/. 35

[CUD] NVIDIA CUDA. a parallel computing architecture, www.nvidia.com/object/cuda_home_new.html. 2, 35

[Fle87] FLETCHER R.: *Practical Methods of Optimization*, second ed. John Wiley & Sons, New York, 1987, ch. 8.7 : Polynomial time algorithms, pp. 183–188. 7

[Gla94] GLASSNER A. S.: *Principles of Digital Image Synthesis*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1994. 5, 7

[GMW81] GILL P. E., MURRAY W., WRIGHT M. H.: *Practical optimization*. Academic Press Inc. [Harcourt Brace Jovanovich Publishers], London, 1981. 7

[ISC*99] II I., SOUSA A. A., COSTA A. C., FERREIRA F. N., III F.: Lighting design: A goal based approach using optimisation, 1999. 3

[KPC93] KAWAI J. K., PAINTER J. S., COHEN M. F.: Radioptimization: goal based rendering. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1993), ACM, pp. 147–154. 3

[Lev44] LEVENBERG K.: A method for the solution of certain non-linear problems in least squares. *Quarterly Journal of Applied Mathmatics II*, 2 (1944), 164–168. 8

[M.02] M. C.: Inverse lighting problem in radiosity. *Inverse Problems in Engineering 10* (1 January 2002), 131–152(22). 4, 39

[Mar63] MARQUARDT D.: An algorithm for least-squares estimation of nonlinear parameters. 431–441. 8

[May] Maya. a 3D modelling environment, usa.autodesk.com/. 25, 27, 46

[Ope] Open Inventor. an object-oriented 3D toolkit, oss.sgi.com/projects/inventor/. 25

[PF92] POULIN P., FOURNIER A.: Lights from highlights and shadows. In *SI3D '92: Proceedings of the 1992 symposium on Interactive 3D graphics* (New York, NY, USA, 1992), ACM, pp. 31–38. 3

[PRJ97] POULIN P., RATIB K., JACQUES M.: Sketching shadows and highlights to position lights. In *CGI '97: Proceedings of the 1997 Conference on Computer Graphics International* (Washington, DC, USA, 1997), IEEE Computer Society, p. 56. 3

[QT] QT. a cross-platform application and UI framework, qt.nokia.com/products. 25, 35

[Qua] Quarter. a light-weight glue library that provides seamless integration between Kongsberg SIM's Coin high-level 3D visualization library and Nokia's QT 2D user interface library, www.coin3d.org/. 25, 35

[Rec73] RECHENBERG I.: *Evolutionsstrategie: optimierung technischer systeme nach prinzipien der biologischen evolution*. Frommann-Holzboog, 1973. 8, 9, 10

[Rec94] RECHENBERG I.: *Evolutionsstrategie '94*. Frommann-Holzboog, 1994. 8, 9

[SDS*93] SCHOENEMAN C., DORSEY J., SMITS B., ARVO J., GREENBERG D.: Painting with light. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1993), ACM, pp. 143–146. 3

[TG97] TENA J. E., GOLDBERG I. R.: An interactive system for solving inverse illumination problems using genetic algorithms, 1997. 3

[YDMH99] YU Y., DEBEVEC P., MALIK J., HAWKINS T.: Inverse global illumination: recovering reflectance models of real scenes from photographs. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1999), ACM Press/Addison-Wesley Publishing Co., pp. 215–224. 4, 45

List of Tables

Listings

List of Figures

Index