University of Wrocław
Department of Mathematics
and Computer Science
Institute of Computer Science

Inverting the lighting transfer for interior scenes

Master's thesis

Paweł Szczepański

Supervisor
dr Andrzej Łukaszewski

Wrocław 2009

Contents

Chapter 1

Introduction

In this work we describe a system for lighting design. The system will be capable of deducing light positions from the final effect painted by the user on the surfaces. As a basis for out work we will use precomputed radiance transfer (PRT) for interiors and attempt to invert the light transfer matrix with suitable constraints.

The second chapter will summarize previous work on precomputed radiance transfer, inverse light transport and lighting design. The methods on which our algorithm is based will be described with more detail.

The third chapter will describe the principles of our method, starting from modifications of PRT algorithm. Nontrivial aspects behind the interactive user interface are also discussed.

The fourth chapter contains the implementation details, including less relevant and more trivial algorithms.

The fifth chapter presents the results, conclusion and highlights avenues for future work.

Chapter 2

Previous work

2.1 Mathematics and physics

Here we will describe mathematical and physical constructs used in the following sections.

2.1.1 Photometry and radiometry

The field of radiometry deals with measuring electromagnetic radiation, including light. Photometry measures specifically light as it is perceived by human visual system. Computing photometric quantity from radiometric one usually requires integrating along the standard human visual system response curves, which is a linear transformation.

Computations are usually carried out in photometric units (luminance expressed in $lumens/sr/m^2$), but the units are almost always omitted. In contrast, the descriptions of the phenomena are most often taken from radiometry (radiance, irradiance). The table 2.1 summarizes the definitions from both photometry and radiometry used through the whole text.

radiometric term	radiometric unit	photometric term	photometric unit	meaning
radiance	$W/sr/m^2$	luminance	$lm/sr/m^2$	light leaving or arriving to a point from a specific direction
irradiance (also radiosity)	W/m^2	illuminance	lm/m^2	light leaving or arriving to a point from all directions
radiant intensity	W/sr	luminous intensity	lm/sr	light leaving or arriving to an area from a specific direction

Table 2.1: Physical quantities used in this text. Specialized literature may describe irradiance as incoming light and radiance (or radiant exitance, emittance) as outgoing light, but we shall use both for either incoming or outgoing light and the meaning will be clear from the context. sr is the measure of solid angle on a sphere.

For example a $100W$ light bulb emits approximately $1700lm$ of light, which for radius $r = 25mm$ gives luminance of $34449lm/sr/m^2$, which when approximated by omnidirectional light, would scale to the luminous intensity of $270lm/sr$.

2.1.2 Spherical harmonics

Spherical harmonics define an orthonormal function basis over a sphere analogous to Fourier basis for one-dimensional functions. Each basis function is characterized by band index $l = 0, 1, 2, \ldots$ and index within a band $m = -l, \ldots, -1, 0, 1, \ldots, l$. l and m respectively are also called degree

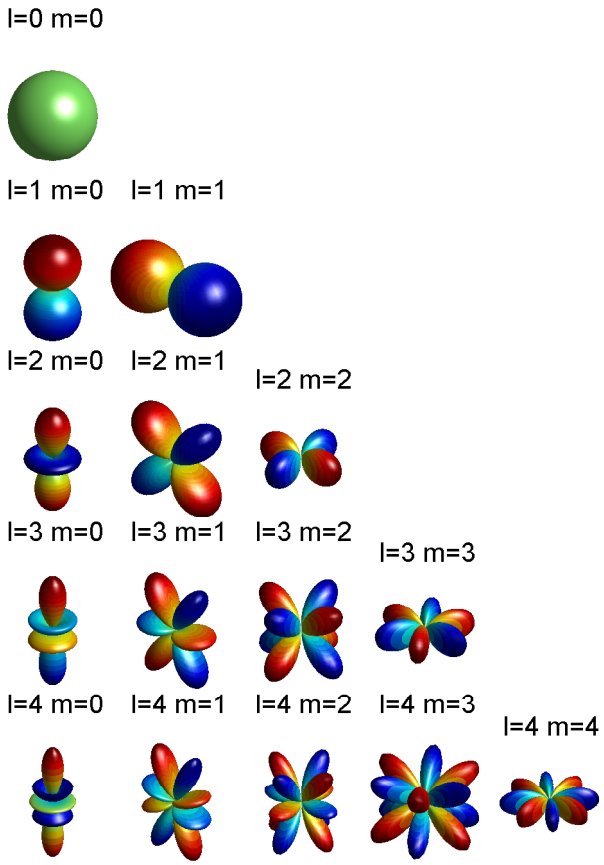


Figure 2.1: Visualization of spherical harmonics. l represents band index (degree), while m is index within a band (order). The plots are for positive m only, negative values are plotted blue.

and order. Figure 2.1 presents first 4 bands for positive m. After [Gre03] we first define Legendre Polynomials using the recursive formulas:

$$\begin{aligned} P_0^0(x) &= 1 \qquad x \in [-1,1] \\ (l-m)P_l^m(x) &= x(2l-1)P_{l-1}^m(x) - (l+m-1)P_{l-2}^m(x) \\ P_m^m(x) &= (-1)^m(2m-1)!!(1-x^2)^{m/2} \\ P_{m+1}^m(x) &= x(2m+1)P_m^m(x) \end{aligned}$$

Legendre Polynomials are used to define SH basis functions:

$$y_l^m(\theta,\phi) = \begin{cases} \sqrt{2}K_l^m cos(m\phi)P_l^m(cos\theta), & m > 0 \\ \sqrt{2}K_l^m sin(-m\phi)P_l^{-m}(cos\theta), & m < 0 \\ K_l^0 P_l^0(cos\theta), & m = 0 \end{cases} \tag{2.1}$$

where $K_l^m = \sqrt{\frac{(2l+1)}{4\pi}\frac{(l-|m|)!}{(l+|m|)!}}$ are normalization factors. These are called the real-valued SH basis functions.

Every function taking arguments in spherical coordinates (including BRDF and radiance) can be represented in this basis as a series of coefficients (we will denote SH representation vector of function f with f^{SH}). If we cut the representation at band l, we arrive at a band-limited representation, which contains only the lowest frequencies of the original function. Therefore SH basis is useful for compression of low-frequency functions. The size of the representation is maximum band index squared, so a representation of maximum degree n requires $(n+1)^2$

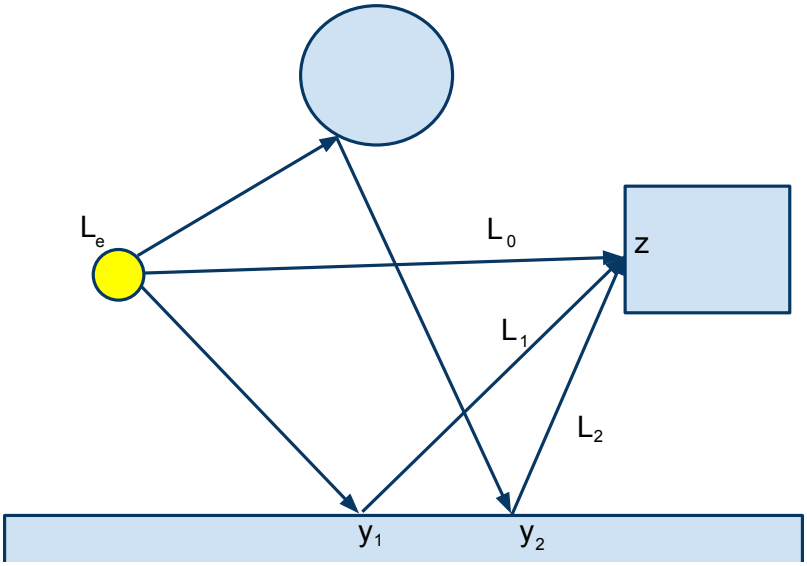


Figure 2.2: Decomposition of lighting arriving to z. L_e is emitted lighting, L_0 is transferred without reflection. L_i is light transferred by rays that bounced i times.

coefficients. We will usually name y using only one index, enumerating all the basis functions from degree 0, then degree 1 and so on.

The most frequently used property of the SH representations is that the integral of their product can be expressed simply as the dot product of the SH coefficients.

$$\int_\Omega f(\omega)t(\omega)d\omega = \sum_i f_i^{SH} t_i^{SH} \tag{2.2}$$

The proof uses the projection equation $f = \sum_i f_i^{SH} y_i$ and basis orthonormality $\int_\Omega y_i y_j d\omega = \delta_{ij}$ where δ represents the Kronecker delta function.

When we represent local properties in SH basis (like material properties, BRDF) and want to transform these to global space, then we face the problem of SH rotation, which is nontrivial, but thoroughly solved [Gre03]. For small angles there exist fast approximations [Kŏŏ5].

2.2 Precomputed radiance transfer

Precomputed radiance transfer (PRT) was introduced in [SKS02]. It is a method of displaying in real-time *static scenes* with *dynamic lighting*. Let us consider rendering equation, which describes light outgoing from point y in the direction of point z:

$$L_{y\to z} = L_{e,y\to z} + \int_{x\to y\in\Omega} V(x\to y)\rho(y, x\to y, y\to z)cos(n_y, y\to x)L_{x\to y}d(x\to y)$$

$L_{e,y\to z}$ denotes the light emitted from y in the direction of point z. $\rho(y, x\to y, y\to z)$ stands for bidirectional reflectance function (BRDF) in point y for incoming light direction $x\to y$ and outgoing direction $y\to z$ while n_y is surface normal at point y. $L_{e,y\to z}$ represents source light emitted by the point (i.e. if $L_{e,y\to z} > 0$, then y is a point on the light source). The visibility function V is defined as:

$$V(x\to y) = \begin{cases} 0 & \text{if } y \text{ is visible from } x, \\ 1 & \text{else.} \end{cases}$$

The rendering equation is recursive, but it can be expanded.

$$L_{y\to z} = L_{e,y\to z} + L_{0,y\to z} + L_{1,y\to z} + \dots \tag{2.3}$$

This expression called the Neumann expansion and is a sum of the emitted lighting L_e, direct lighting L_0, light that bounced from the scene once L_1, twice L_2 and so on (figure 2.2). Although not every surface is a light source, computing L_0 can be a hard problem itself, mainly due to the visibility term. For point lights as an example, it can be solved using shadow mapping or shadow volumes. Computing further bounces is made simple by using transfer function T, which corresponds to single bounce of light (also L_0 is presented using the transfer function for simplicity).

$$T_f(y \to z, L') = \int_{x\to y\in\Omega} V(x \to y)\rho(y, x \to y, y \to z)cos(n_y, y \to x)L'_{x\to y}d(x \to y) \tag{2.4}$$

where L' denotes any light function of point and direction.

Then we have

$$\begin{aligned} L_{0,y\to z} &= T_f(y \to z, L_e) && (2.5)\\ L_{1,y\to z} &= T_f(y \to z, L_0) && (2.6)\\ L_{2,y\to z} &= T_f(y \to z, L_1) = T_f(y \to z, T_f(y \to z, L_0)) && (2.7)\\ L_{i,y\to z} &= T_f(y \to z, L_{i-1}) && (2.8) \end{aligned}$$

What we will now emphasize are various discretizations of the transfer problem. We will consider different means of storing L_e (the light sources), the incident lighting on geometry L_i, material properties ρ and visibility V.

2.2.1 Distant low-frequency lights, diffuse materials, linear vertex basis

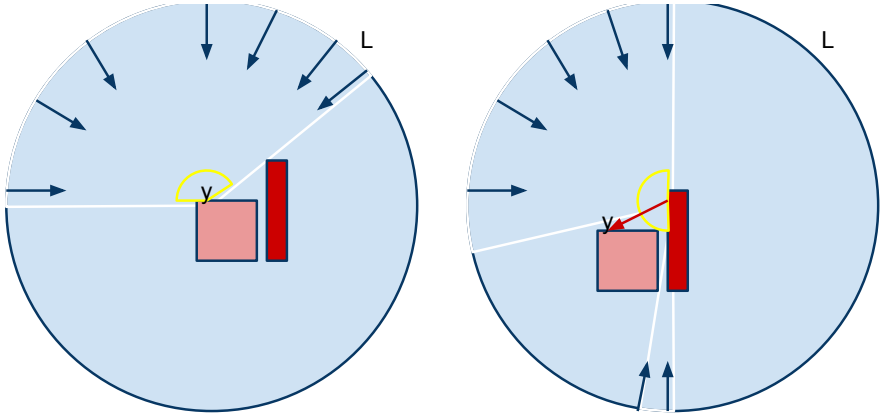


Figure 2.3: Point y is lit by light in infinity, expressed by sphere centered in the gathering point. The left image shows direct lighting, positive visibility is marked white. The right image shows gathering of $L_{1,y}$ (only one gathering point is shown).

This is the classic PRT approach. Light sources are assumed to be distant, therefore can be stored on a sphere at infinity. Frequently spherical harmonics (SH) basis of maximum degree n is used, so we will denote it L_e^{SH} (see section 2.1.2).

Lighting on surfaces is stored in each vertex and interpolated linearly.

Within each vertex lighting is stored for each incoming direction – we use SH basis for this purpose. For example for direct lighting, for single target vertex point we have:

$$\begin{aligned} L_{0,y} &= \int_{x\to y\in\Omega} (V(x \to y) \cdot \rho \cdot cos(n_y, y \to x))L_{e,x\to y}d(x \to y) && (2.9)\\ &= T_{0,y}^{SH} \cdot L_e^{SH} && (2.10) \end{aligned}$$

where $T_{0,y}$ is a vector of coefficients in SH basis representing transfer towards point y. These coefficients are computed using $T^{SH}_{0,y,i} = \int_x (V(x \to y) \cdot \rho \cdot cos(n_y, y \to x)) y_i$. For the purpose of this and next subsection, we will call $T^{SH}_{0,y}$ the single-bounce transfer vector. The resulting dot product is the effect of applying equation 2.2.

This transfer holds regardless of outgoing direction, because of constant BRDF ρ. Equation is linear, so it can be expressed in matrix form. If we define L_0 as vector of source light intensities in all vertices, then it is useful to put the transfer vectors into one matrix T_0. This matrix defines the SH representation of the function T_f:

$$L_0 = T_f(L_e) = \left(T^{SH}_{0,v_1}\ T^{SH}_{0,v_2} \cdots T^{SH}_{0,v_N}\right)^T L^{SH}_e = T_0 L^{SH}_e \tag{2.11}$$

where v_i are scene vertices and L^{SH}_e is a vector of lighting coefficients in the SH basis (the same lighting in infinity). Each subsequent bounce can be decomposed in the same way, only that we integrate over rays that hit the lighting sphere after i bounces with proper weighting factors due to each bounce (fig. 2.3).

We can compute i-th bounce transfer vector more efficiently, by spherical gathering (numerically integrating over hemisphere) the transfer vectors T_{i-1,v_j} from the previous bounce. The update rules are presented in [SKS02].

Again the transfer vectors $\{T_{i,v_j}\}_j$ form the transfer matrix T_i (as in equation 2.11), each representing partial transfer function, where light undergone exactly i bounces. The equation 2.3 simplifies to:

$$L_y = T_0 L^{SH}_e + T_1 L^{SH}_e + T_2 L^{SH}_e + \ldots = (\sum_i T_i) L^{SH}_e = T L^{SH}_e \tag{2.12}$$

Transposition of each row of the full transfer matrix $T = \sum_i T_i$ (corresponding to a target vertex) will be called the (full) transfer vector. i-th element of a transfer vector tells us how i-th SH basis function of lighting affects one output vertex.

2.2.2 Distant low-frequency lights, low-frequency materials, linear vertex basis

When we allow more complex materials (e.g. glossy materials), we must account for the outgoing direction. For each vertex but also for each direction we compute transfer vector $T_{0,v_i,\omega}$. If we parametrize outgoing directions by casting transfer vector to the SH basis, we obtain $(n+1)^2 \times (n+1)^2$ transfer matrix for each vertex $T^{SH}_{0,v_i,\Omega^{SH}}$ (where n is maximum degree of SH expansion). Usually we would use different expansion degrees – higher for spherical lighting L^{SH}_e, lower for each column of the per vector transfer matrix corresponding to one SH basis function spanning the outgoing direction space. The expansion across the outgoing direction can be lower due to the BRDF convolution acting as low-pass filter.

Dynamic algorithm for computing transfer matrices is presented in [SKS02] and carefully derived in [KSL05].

2.2.3 Interior lighting, diffuse materials, linear vertex basis

For interior lighting, the lighting hemisphere is different for each vertex, so it cannot be a function of the same SH-encoded spherical map. Sometimes the solution is to compute different spherical maps for different points in a scene and place those on a spatial grid. When we assume that for moving interior lighting we can produce such grid in real time, then the approach does not differ much from that presented in section 2.2.1.

The other approach is to express the global lighting in a vertex as a linear function of direct lighting in other vertices. From this point the matrix T of size $|V| \times |V|$ will denote this function of intra-vertex transfer and will be called transfer matrix (in contrast to the previous section, where transfer matrix represented lighting-vertex specular transfer in SH basis). Such approach was used in the early radiosity methods [CW98]. The full transfer defined below will henceforth be denoted S.

$$S = \sum_{i=0,1,\ldots} T^i \tag{2.13}$$

To use vertex-vertex transfer, we must first project the source lighting L_e onto the scene. The lights may move, so we may be forced to do this in each frame. There are two ways to project the lighting. In the first scenario we assume that we can compute the direct lighting in all vertices affecting the visible ones (which may unfortunately mean all vertices in a given room area or the scene) in real-time during light motion. The second scenario assumes that the lighting s already projected on th vertices of the scene, so we avoid real-time projection by expressing the lights in vertex basis – as a vector L_e, where $L_{e,y}$ is diffuse lighting in vertex y. This way however the dynamic character of the lighting is limited to attenuating the emitting properties of the static scene vertices. The second scenario is commonly exercised when placing lighting on predesigned surfaces.

$$\begin{aligned} L_0 &= TL_{e,y} \\ L_i &= TL_{i-1} \\ L &= (\sum_{i=0,1,\ldots} T^i)L_e = SL_e \end{aligned} \tag{2.14}$$

2.2.4 Interior lighting, diffuse materials, meshless basis

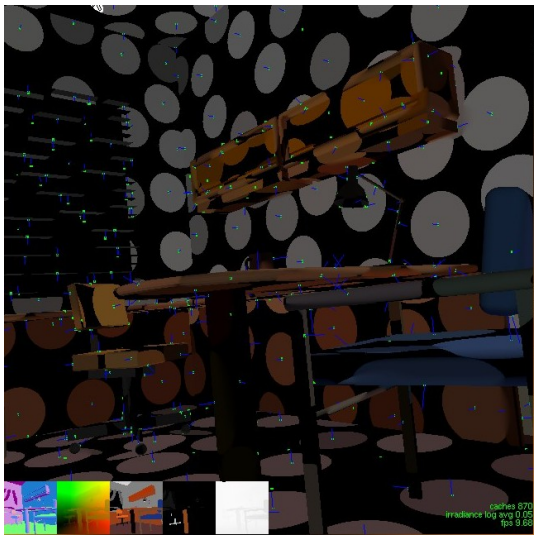


Figure 2.4: Caches with normals. The cache radius has been set substantially smaller for better visualization.

To make the lighting independent from surface model and level of detail, we can store lighting in points other than mesh vertices. This has disadvantage of more complex interpolation between values placed in arbitrary points in 3-dimensional space. We shall call such points *caches* and the basis *meshless*.

Storing lighting in arbitrary points was common in offline rendering – the keywords include irradiance caching [WRC88] and radiance caching [Kŏŏ5]. The authors were storing irradiance scalars or radiance SH representations (vectors) in caches, while dynamically modifying cache density for maximum detail and minimum cache count. Although the accuracy of lighting computed in each cache must be high for correct interpolation, if the lighting basis is hierarchically decomposed, then the rendering times can be reduced substantially [HMS09].

The meshless basis for PRT was presented in [LZT+08]. Again, we seek matrix T, but in contrast to section 2.2.3 and equation 2.14 this matrix represents the intra-cache transfer:

$$L = (\sum_{i=0,1,\ldots} T^i)L_e = SL_e$$

Let L_c contain the outgoing radiance in a cache c (for diffuse case – a scalar, for specular case – a vector of coefficients). We obtain the interpolated radiance value in a point x as follows:

$$L_x = \sum_{c \in C} B_x(p) L_c = \frac{\sum_{c \in C} w_{x,c} L_c}{\sum_{c \in C} w_{x,c}} \tag{2.15}$$

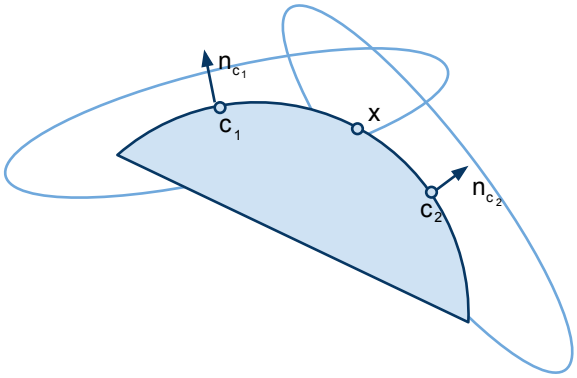


Figure 2.5: Interpolation of irradiance in x from irradiance in c_1 and c_2. The ellipsoids around caches visualize weight squashing in orthogonal planes.

The right-hand side stands for weighted interpolation with weight $w_{x,c}$ for each cache position c. Weight depends on position of x in relation to c and normals n_x, n_c in both points:

$$w_{x,c} = K_c(dist(x,c)) max(0, n_c^T n_x)^2 \tag{2.16}$$

$$K_c(r) = K(r/r_c) \quad K(r) = \begin{cases} 2r^3 - 3r^2 + 1, & r \leq 1 \\ 0 & r > 1 \end{cases} \tag{2.17}$$

$$dist(x,c) = ||x - c + \sigma\ ((x-c)^T n_c)\ n_c|| \tag{2.18}$$

where K is the kernel smoothing within the cache radius r_c, and $dist$ is the 5-dimensional distance function (of also normals in the cache center). Squash factor σ increases the distance when x is not on the plane tangent to n_c at c. For $\sigma = 0$ we receive euclidean distance. We use $\sigma = 2$. This distance function is used for interpolation.

Then symmetric distance between two caches is defined as:

$$D(p,q) = \frac{min(dist(p,q), dist(q,p))}{max(0, n_p, n_q)^2} \tag{2.19}$$

The symmetric distance is used to distribute caches in the scene according to Poisson disk distribution. The more curved the surface, the more caches it will receive, because of supposedly higher lighting frequency.

Sadly, we are not able to guess the true lighting frequency, because precomputed radiance transfer is done before source lighting is known. [WRC88] report that knowing the lighting gradient (first order expansion) in a cache point can greatly benefit interpolation and sampling. Analytical methods for computing lighting gradients for simple scenarios (non-global lighting) are known ([RMB07]).

In this work we extend the idea presented in [LZT+08]. The details are given in section 3.2.

2.3 Inverse rendering

Our work falls into the area of inverse rendering. Inverse rendering itself is much broader term: given final scene looks (e.g. in a form of an image) we may reconstruct numerous parameters set before the rendering process, like BRDF [RH01], texture and atmospheric parameters. We focus on inverse lighting, and strive to reconstruct light positions and intensities. Most of the work in inverse rendering reconstructs these parameters from images (image-based methods), running a certain optimization process with pixel similarity criterion.

Some of these methods are sufficiently general to be easily reformulated for object space. [Mar98] for example chooses the lighting basis as discrete set of area lights on a distant sphere. For each light in the basis an image is rendered (with the assumption that 3D models are also available). The set of all pixels of this image can be thought of as matrix columns from section 2.2.4. The optimization then follows to compute suitable linear combination of basis images.

2.4 Lighting design

The lighting design interfaces range from CAD systems where the user inputs the light parameters explicitly, to intuitive systems where the effect is painted directly on the object surface. We want to introduce the system of the second kind, that enables the user to paint lighting in the interior scenes and reconstructs the most desired light source locations within the scene while taking the global lighting effects into account.

The work in [OMSI07] is nearest to what we want to achieve, but it concentrates around designing lighting surrounding a single (though arbitrary complex) object – lighting is not a part of the scene and illuminates it from the outside. While useful for most situations, we extend this approach for lighting of the interior scenes by using different PRT model and optimization methods. Moreover, instead of spherical harmonics used previously, spherical radial basis functions (SRBF, introduced in [TS06]) are used. Worth-mentioning details are selective tone mapping (for region of interest specified by user) and used Tikhonov regularization terms for least squares optimization to better handle underconstrained cases. The software behind [OMSI07] allows painting both specular and diffuse lighting with arbitrary colors, we allow only white lights.

[PBMF07] considers the renderer as a black box: on input we have the lights' parameters (position, direction, color, distance cutoffs, distance falloffs, barn shape, shadow density and more) and on the output we have one rendered image as a set of pixels. The approach allows plugging in arbitrarily complex rendering algorithm (including for example non-photorealistic cartoon renderer or pathtracer), but requires nonlinear optimization procedure to yield the parameters' values, which may require even hundreds of objective function evaluations (i.e. renderings). Slow renderer leads to slow convergence, but the approach could benefit from PRT renderer, as well as we do. Interesting ideas include brush design (optionally weighting brush intensity by normal difference or steering specular reflections using texture preservation brush) and building importance map guided by the painting history. In contrast we consider only discrete (however very large) set of positions and fixed light parameters except intensity. Moreover we optimize for lighting on the whole scene, whereas [PBMF07] optimization is performed on an image.

[OKP+08] does not strive to reconstruct the lighting parameters, but presents an approach to store edits of the light transfer. The transfer is stored for a certain viewport as two matrices per channel (transfer modulation and offset) that describes the final gathering relation between points on the scene. Each matrix row is compressed using wavelets. Incorporating any final gathering renderer is straightforward, but edited transfer is limited to indirect and diffuse.

[SDS+93] presents the system similar to ours, although does not set the solution within the PRT framework explicitly. The optimization problem is solved using modified Gauss-Seidel algorithm on vertices.

Chapter 3

Inverting the lighting transfer

Here we will describe the mathematical principles and main algorithms behind the implemented system for lighting design. The technical details will be given in the next chapter.

3.1 Workflow

The stages of program execution are listed below:

1 load a scene
2 generate caches on the surfaces
3 precompute radiance transfer function
4 load preset lights (optional)
5 **repeat**
6 | transfer radiance from lights (compute $L = SL_e$)
7 | display the radiance transferred from the current light set
8 | get target radiance from the user
9 | compute lights
10 **until** *the user quits*

The PRT framework is based on the model presented in section 2.2.4. The modifications are described in section 3.2. The second step determines cache positions as described in the section 4.1.1. The result of the third step is the full transfer matrix S, such that

$$L = SL_e$$

The matrix can be saved with the scene for future executions.

The fourth step takes the lights from the loaded scene and casts these onto the PRT caches to fit into the model. Details are presented in section 4.1.1. The step is optional and the initial lighting may be zero. To aid the user in recognizing the objects, scene with no defined lights will be rendered with minor shading from auxiliary camera light.

The steps $6 - 9$ are iterated by the user until the result is satisfactory. First the PRT matrix S is applied to the light vector L_e and the resulting radiance vector L is displayed to the user as lit scene (see section 4.2). Then the user can paint the desired lighting (section 3.3) and run the optimizer (section 3.4). The optimizer computes the light vector L_e that is again transformed and displayed to the user.

The light intensities and positions can be viewed and exported at any time.

3.2 PRT

Due to the ability to describe lighting in interiors we use the system described [LZT+08] as a basis for modifications. The original paper does not deal with some drawbacks of the approach:

- memory requirements for SH basis,
- no error metrics,
- deferred shading prevents complex material support.

We extend the original idea towards the glossy material support.

3.2.1 Incoming radiance and outgoing radiance

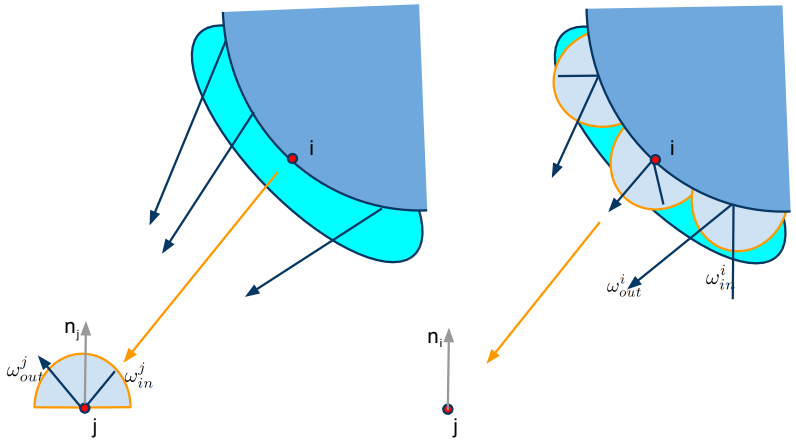


Figure 3.1: Light transfer from cache to cache premultiplied with target BRDF (left) and only with source BRDF storing incoming light only (right). n_i and n_j are normals in i and j, while ω^i_{in}, $\omega^i_{out} = -\omega^j_{in}$ and ω^j_{out} are vectors directed towards or outwards i and j respectively.

Cache interpolation was defined in equation (2.15) with $B_c(p)$ as normalized weight of cache c in point p.

In the original paper the transfer function between caches i and j is defined as a spherical integral over incoming direction to point j denoted w^j_{in}:

$$T_{i\to j} = \int_{\Omega^j_{in}} B_i(r(x_j, \omega^j_{in}))f_j n_j^T \omega^j_{in} d\omega^j_{in}$$

In the above equation we use raycast function $r(x, \omega)$, which returns the closes point to x in the direction ω. We are gathering irradiance from the cache i by interpolating its irradiance in points visible from j, then we multiply these by BRDF f in j to finally get irradiance outgoing from j. Irradiance in j can be expressed solely in terms of irradiance in i.

If the materials are glossy, then we cannot premultiply by BRDF and have to store radiance incoming to j. Fundamental difference is that we gather on the previous point and multiply by BRDF of i (figure 3.1). Then radiance is transferred by the integral

$$L_{i\to j,\omega^j_{in}} = B_i(r(x_j, \omega^j_{in})) \int_{\Omega^i_{in}} L_{i,\omega^i_{in}}(f_i(\omega^i_{in}, \omega^i_{out})(n_i^T \omega^i_{in}))d\omega^i_{in} \tag{3.1}$$

$$= \int_{\Omega^i_{in}} L_{i,\omega^i_{in}} \left(B_i(r(x_j, \omega^j_{in}))f_i(\omega^i_{in}, \omega^i_{out})(n_i^T \omega^i_{in}))\right) d\omega^i_{in} \tag{3.2}$$

To get the transfer function, we leave out the L_i (this is again possible by expressing the functions in a conveniently integrable basis like SH) and use equation 2.2:

$$L_{i\to j,\omega^j_{in}} = L_i^{SH} \cdot T^{SH}_{i\to j,\omega^j_{in}} \tag{3.3}$$

$$T^{SH}_{i\to j,\omega^j_{in},k} = \int_{\Omega^i_{in}} B_i(r(x_j, \omega^j_{in}))f_i(\omega^i_{in}, \omega^i_{out})(n_i^T \omega^i_{in})y_k(\omega^i_{in})d\omega^i_{in} \tag{3.4}$$

We can represent the function $T^{SH}_{i\to j,\omega^j_{in},k}$ itself in SH basis to get rid of the ω^j_{in}, and finally we arrive in the $T^{SH}_{i\to j}$, which we will call transfer submatrix in this case (it is analogous to transfer matrix in section 2.2.2):

$$T^{SH}_{i\to j,k} = \int_{\Omega^j_{in}} T^{SH}_{i\to j,\omega^j_{in},k} y_k(\omega^j_{in}) d\omega^j_{in} \tag{3.5}$$

This is the most general case, when we allow every surface on the way to be glossy and reflect different lighting in every direction. For SH degree n, i.e. $(n+1)^2$ coefficients, $T_{i\to j}$ is itself a matrix $(n+1)^2 \times (n+1)^2$. The memory usage for SH degree n is therefore $O(n^4)$ times the normal usage with one huge transfer matrix per color channel. If we only allow for the last (direct) bounce to be glossy, then we can use vectors instead of matrices, and memory usage becomes $O((n+1)^2)$ times more than diffuse-only case.

For fully diffuse case we may remove ω^j_{in} dependence from the equation 3.2. Also we can store the lighting in i – $L_{i,\omega^i_{in}}$ as a scalar, L_i/π being the integrated average incoming radiance, because BRDF is no longer directional. After integrating 3.2 over ω^j_{in}:

$$L_{i\to j} = \int_{\Omega^j_{in}} \int_{\Omega^i_{in}} L_{i,\omega^i_{in}} B_i(r(x_j, \omega^j_{in})) f_i (n_i^T \omega^i_{in})(n_j^T \omega^j_{in}) d\omega^i_{in} d\omega^j_{in} \tag{3.6}$$

We move the unaffected terms in front of the inner integral:

$$L_{i\to j} = \int_{\Omega^j_{in}} L_i B_i(r(x_j, \omega^j_{in})) f_i \frac{\int_{\Omega^i_{in}} L_{i,\omega^i_{in}} (n_i^T \omega^i_{in}) d\omega^i_{in}}{\pi} (n_j^T \omega^j_{in}) d\omega^j_{in} \tag{3.7}$$

We will define L_i as the value of the inner integral and move it outside the outer integral:

$$L_{i\to j} = L_i \int_{\Omega^j_{in}} B_i(r(x_j, \omega^j_{in})) f_i (n_j^T \omega^j_{in}) d\omega^j_{in} \tag{3.8}$$

The integral will thus be our scalar transfer coefficient of the single-bounce transfer matrix T:

$$T_{i\to j} = \int_{\Omega^j_{in}} B_i(r(x_j, \omega^j_{in})) f_i (n_j^T \omega^j_{in}) d\omega^j_{in} \tag{3.9}$$

This result differs from the one in the original paper only by the factor f_i instead of f_j since we moved the gathering from j to i.

3.2.2 Initial transfer matrix

After equation 2.14, the full transfer for infinite ray is:

$$S = (0 + T + TT + TTT + \ldots) S_0 \tag{3.10}$$

S_0 is a matrix of, which elements describe how much source light from light cache arrives into receiver cache and will be called initial transfer matrix. We use separate matrix for the first bounce to incorporate some light characteristics, which are not present in inter-surface transfer, for example BRDF is replaced by uniform emission. Notice that we use 0 instead of identity I, because the light sources themselves (surface emission properties) are rendered as a separate pass and are not to be used as objective during the inverse transfer. When regarding light source caches as point lights, the initial transfer matrix' elements simplify to:

$$S_{0,i\to j} = V(i \to j)$$

The further transfer T remains area-to-point transfer and requires numerical integration.

3.2.3 Towards glossy materials

The biggest obstacle we have to account for is the memory limit. Transfer matrices are sparse and stored as column-major matrix: vector X of row indices, which contain nonzero values, vector V of corresponding nonzero values and mapping Y from column index j into X and V, such that values in X and V are from $j-th$ column from $V[Y[j]]$ to $Y[C[j+1]]$. This gives $4(2e+|C|)$ as the size in bytes, where e is nonzero element count and $|C|$ represents number of caches and we assume 32-bit data and indices. The matrix sizes for sample scenes are grouped in a table 5.2 and easily become over 20MB per channel. For diffuse transfer we need three such matrices. For spherical harmonics: the expansion of degree 7 to 10 is usually sufficient, which gives 64 to 121 matrices per channel. Storing such amounts of data is feasible only in the presence of compression and caching matrices on the hard drive.

3.3 Illumination brush

We chose painting as the method for designing lighting, because it is natural, sufficiently accurate and general. Just as our method of representing lighting, the painting process does not put any assumptions on scene geometry.

Another method for modifying the scene is to explicitly set the desired outgoing radiance to selected polygons and objects. Editing selections of polygons can be easily incorporated into our framework.

The user drags the mouse over the screen producing a stroke which is a sequence of samples S in the object 3-dimensional space. The stroke itself may have radius assigned, which is useful for selecting caches and painting large objects. We want to elevate (or reduce) the outgoing radiance along the stroke to a new value t. The desired luminance t is the luminance of the first clicked pixel after adding or subtracting some fixed amount. This luminance is then imposed upon the following pixels in a stroke.

In the following sections we will explore some of the methods of imposing desired luminance. The method described in section 3.3.3 (with similarity term on outgoing radiance) is the one currently used in the system.

3.3.1 Simple thick brush

t	sample outgoing radiance (luminance – wavelength independent)
C	the set of all caches
$c \in C$	a cache
ρ_c	BRDF in cache position (incorporated cosine term!)
$p_c > 0$	cache incoming irradiance
w_c	weight of the cache c contributing to the current sample
$l_c = p_c\rho_c$	cache outgoing radiance

If the brush stroke is thick (has large radius with respect to cache density) then the fastest way to impose the desired luminance between the caches is just setting the incoming irradiance stored in caches within a stroke to $p_c = t/\rho_c$.

This approach will not work well for glossy surfaces, in the case when we receive strongest reflections between the caches. Sampling the pixels on these glossy reflections and not only in the caches is inevitable.

3.3.2 Point-by-point relighting

For greater control or when the brush is significantly thinner than cache radius, we impose the desired luminance on each pixel in the stroke center, expressing the luminance in pixel as weighted incoming irradiance from caches. Assume that we process only one sample at a time.

We define new incoming irradiance $p_c, i+1$ multiplying the old incoming irradiance by a heuristics of how it would change. t/l_c estimates how did the outgoing irradiance change, because the change in incoming irradiance is proportional. Let us express new incoming irradiance as:

$$p_{c,i+1} \quad = \quad p_{c,i} \times (t/l_c) \tag{3.11}$$

This gives (from interpolation equation 2.15 multiplied by reflectances ρ_c):

$$\sum_{c \in C} \rho_c w_c p_{c,i+1} = \tag{3.12}$$

$$\sum_{c \in C} \rho_c w_c p_{c,i} t / l_c = \tag{3.13}$$

$$t \sum_{c \in C} \rho_c w_c p_{c,i} / l_c = \tag{3.14}$$

$$t \sum_{c \in C} w_c = t \tag{3.15}$$

which is what we want.

3.3.3 Stroke relighting

When doing a brush stroke and modifying several pixels, the point-by-point method modifies caches for local effect each time a pixel is added, which may destroy the outcome for previous points. Stroke relighting solves this problem with optimization.

Let us define S as set of target points (stroke) and C as set of caches affecting these points. We will describe several approaches to the stroke relighting.

With similarity term on cache incoming irradiance

We will derive optimization scheme for the left-hand side of (3.12). We want the values in sample points to increase, as before which is described by first term of equation below, but we also want the cache values to change as slightly as possible to account for not sampled values, which do not change. The second term therefore is a similarity term between the new and the old incoming irradiance.

$$E_f = \sum_s (\sum_c \rho_s w_{s,c} p_c - t_s)^2 + \lambda \sum_c (p_c - p_c^{old})^2 \quad p_c > 0$$

This is an example of the multi-objective (two-objective) least squares problem, where λ represents tradeoff between the two objectives. The solution reduces to one least squares problem as described below. Let us define:

$$\phi_{s,c} = \rho_s w_{s,c} = (w_{s,c} l_{s,c}) / p_c^{old}$$

$$\begin{aligned} \Phi &= [\phi_{s,c}] \in \mathbb{R}^{S \times C} \\ C &= [p_c] \in \mathbb{R}^{C \times 1} \quad \text{(unknown)} \\ C' &= [p_c^{old}] \in \mathbb{R}^{C \times 1} \\ Y &= [t_s] = [t] \in \mathbb{R}^{S \times 1} \\ I &\quad \text{identity matrix} \in \mathbb{R}^{C \times C} \end{aligned}$$

$$A = \begin{bmatrix} \Phi \\ \sqrt{\lambda} I \end{bmatrix} \in \mathbb{R}^{(S+C) \times C} \tag{3.16}$$

$$Z = \begin{bmatrix} Y \\ \sqrt{\lambda} C' \end{bmatrix} \in \mathbb{R}^{(S+C) \times 1} \tag{3.17}$$

Then minimizing E_f is equivalent to solving the simple least squares problem:

$$min_C ||AC - Z||^2$$

With similarity term on outgoing radiance

We can express similarity term in terms of radiance in cache points. Getting the outgoing radiance from incoming irradiance in cache points involves trivial computations and with the assumption that the brush does not pass exactly through cache points, the following energy function should provide useful solution.

$$E_f = \sum_s (\sum_c \phi_{s,c} p_c - t_s)^2 + \sum_{c'} (\sum_c \phi_{c',c} p_c - \sum_c \phi_{c',c} p_c^{old})^2 \tag{3.18}$$

$$= \sum_s (\sum_c \phi_{s,c} p_c - t_s)^2 + \sum_{c'} (\phi_{c',c'} p_{c'} - \phi_{c',c'} p_{c'}^{old})^2 \tag{3.19}$$

$$= \sum_{s,c'} (\sum_c \phi_{s,c} p_c - t_s)^2 \tag{3.20}$$

In the equation 3.18 we use the property that when c' and c are both cache positions, then $\phi_{c',c}$ is zero when $c \neq c'$. We then sum over both sample points on a stoke and cache positions, so that all sums can be expressed as one matrix multiplication. After properly extending the matrix ϕ and vector Y, we get a system like in the case without similarity term. Thus, for new rows within the matrix:

$$\begin{aligned} \phi'(c',c') &= \rho_{c'} \quad \text{diagonal entries of the submatrix} \\ \phi'(c',c) &= 0 \quad \text{non-diagonal entries of the submatrix} \quad (c' \neq c) \\ t_{c'} &= \phi_{c',c'} p_{c'}^{old} = \rho_{c'} p_{c'}^{old} \end{aligned}$$

The new problem is

$$min_C ||AC - Z||^2$$

with

$$A = \begin{bmatrix} \Phi \\ \sqrt{\lambda}\Phi' \end{bmatrix} \in \mathbb{R}^{(S+C)\times C} \tag{3.21}$$

$$Z = \begin{bmatrix} Y \\ \sqrt{\lambda}C' \end{bmatrix} \in \mathbb{R}^{(S+C)\times 1} \tag{3.22}$$

$$\Phi = [\phi_{s,c}] \in \mathbb{R}^{S\times C} \tag{3.23}$$

$$\Phi' = [\phi'_{c',c}] \in \mathbb{R}^{C\times C} \tag{3.24}$$

$$C = [p_c] \in \mathbb{R}^{C\times 1} \quad \text{unknown} \tag{3.25}$$

$$C' = [\phi_{c',c'} p_{c'}^{old}] = [\rho_{c'} p_{c'}^{old}] \in \mathbb{R}^{C\times 1} \tag{3.26}$$

$$Y = [t_s] = [t] \in \mathbb{R}^{S\times 1} \tag{3.27}$$

Solution is straightforward. A sample result is presented in figure (3.2). The method is wider known as least squares regularization and was used for example in [Mar98]. Usually additional empirical parameter λ is used to steer regularization strength. All new rows in the extended matrix are then multiplied by λ.

3.4 Inverse PRT

Here we will describe the possibility of inverting the light transfer, i.e. computing source lighting L_e when the effect $L = SL_e$ is given. The S matrix is the full transfer matrix defined in 3.10. Of course when no constraints are present, the problem is simply solving the system $L = SL_e$.

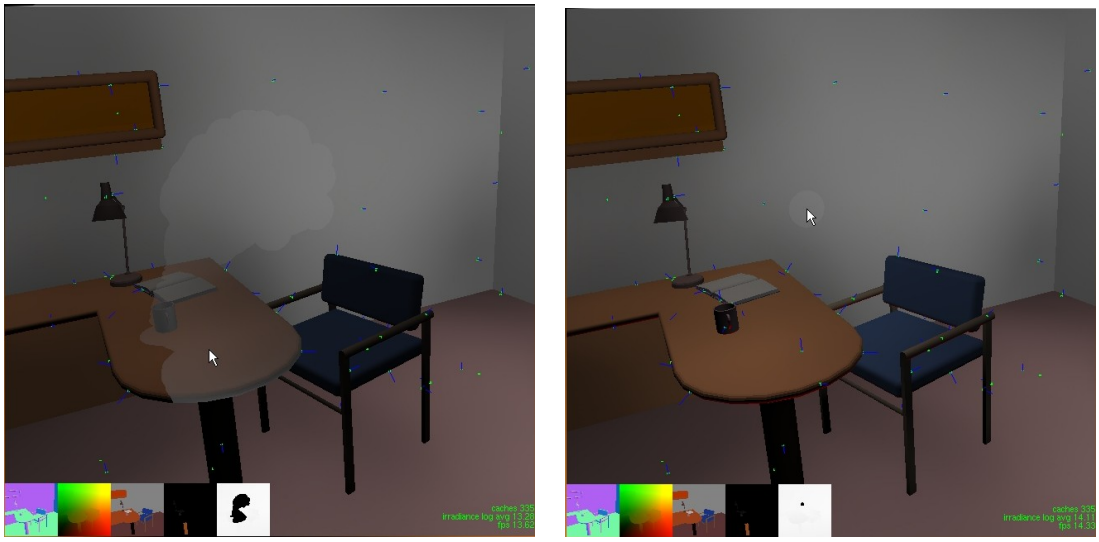


Figure 3.2: Painting the lighting. The left image shows brush stroke. The right image shows the result interpolated from existing caches using method from section 3.3.3.

3.4.1 The transfer matrix

We define first norm of a matrix A as $||A||_1 = max_j \sum_i |A_{ij}|$, which is the maximum absolute column sum of A. This norm is induces by the $||\cdot||_1$ Manhattan norm of the vector. For each scene we estimate the condition number of a matrix based on its first norm $cond(A)_1 = ||A||_1||A||_1^{-1}$. We use the iterative algorithm for $||\cdot||_1$ estimation without computing A^{-1} explicitly from [HT00] (algorithm 2.1).

[RH01] notes that for purely diffuse case the inverse rendering problem is badly conditioned (underconstrained). The results for the sample scenes confirm this statement and are summarized in table 5.2.

3.4.2 Unconstrained inversion

For any scene we compute the lighting using the following equation:

$$SL_e = L \tag{3.28}$$

where S is light transfer matrix is a function of scene geometry. Lighting design involves painting intensities on a scene, which are then stored as elements of vector L, therefore light source intensities L_e (caches acting as point lights) can be obtained by solving the above system. As the matrix is badly conditioned for diffuse case, a set of constraints is necessary.

3.4.3 Constraints on lights' positions

The system enables the user to select caches that are allowed to emit light. Marking the caches as not-emitting is the same as deleting rows of L_e and corresponding columns of S and usually yields an overdetermined system and the following optimization problem:

$$min||SL_e + L|| \tag{3.29}$$

We use nontrivial implementation of nonnegative linear least squares (NNLS) algorithm from [LH74] to solve it (see details in section 4.3).

3.4.4 Constraints on other lights' properties

Assume we want to place only specific lights on a scene. Every constraint not related to light power can be realized before the precomputation phase. Constraints on position (arbitrary 3D point) can

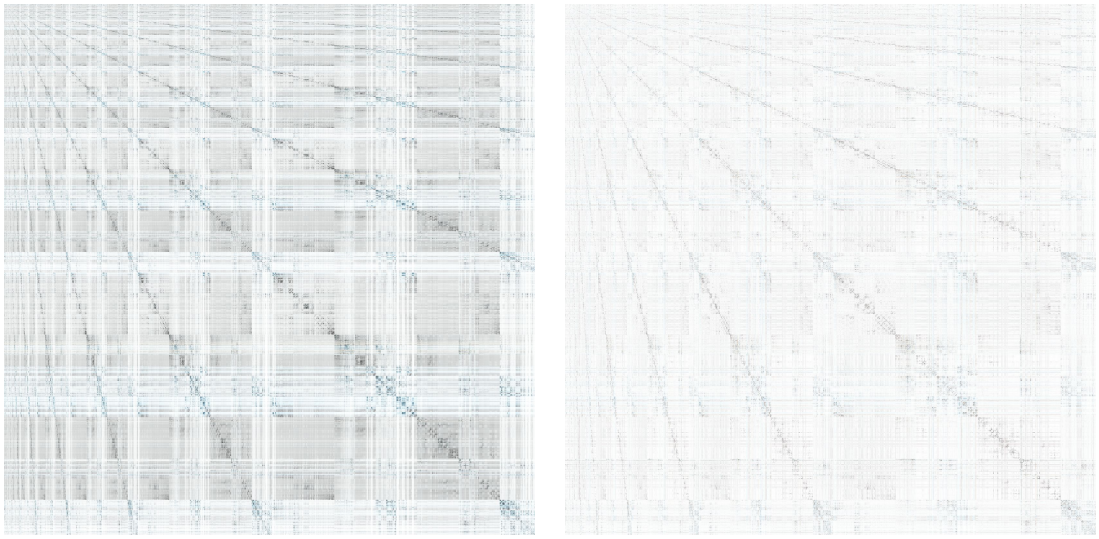


Figure 3.3: The visualization of the full transfer matrix for office scene (left). The entries in range from 0 to 2.0 were mapped to the display range (black for large magnitude). Strong block character can be observed. The right image represents the difference between the matrix and its transposition (black for large difference), showing that the matrix is not symmetric, although the left image strongly suggests this.

be added as additional caches. Constraints on light shape and directionality can be incorporated into S_0 initial transfer matrix and if lights are point lights, then applying such constraints involves one sparse matrix multiplication (it does not involve raytracing in particular).

If we allow only discrete set of light powers with constant step (i.e. $p, 2p, 3p, \ldots, kp$), then we receive model exemplar of the finite integer least squares problem, which is NP-complete. Both approximate and exact algorithms exist [HV03].

3.4.5 Importance

The initial results confirmed the need of using the importance map as in [PBMF07]. Importance map is generated automatically given two parameters – importance v of not painted caches and importance δ of caches painted by previous strokes. The resulting importance w_c is assigned to each cache c. Initially $w_c = v$. When a stroke is painted, the cache importance is updated: for caches affected by stroke, importance of one is assigned ($w_c = 1$), for caches unaffected the importance is mixed with importance for not painted caches using $w_c := (1 - \delta)v + \delta w_c$ (importance decreases). We can incorporate the importance into the existing least squares optimization as the weighting function, thus introducing the weighted least squares. To do so we divide both the matrix rows and the target illumination vector entries by respective importance values squared (as w_c corresponds to allowed relative standard deviation). The new matrices' entries are defined as follows:

$$\begin{aligned} (S_w)_{ij} &= S_{ij}/w_i^2 \\ (L_w)_i &= L_i/w_i^2 \end{aligned}$$

Then the minimization 3.29 becomes:

$$min||S_w L_e + L_w|| \tag{3.30}$$

By using low initial importance v, the user can run the successful optimization after painting only the interesting areas, and letting the optimizer decide what is the lighting to be on the remaining surfaces.

3.4.6 Regularization

After adding importance weighting of the solution, we discover that the optimizer tends to boost lights lighting unimportant scene areas. We therefore add least-norm regularization. The new objective function is stated by

$$min\ ||S_w L_e + L_w|| + \mu||L_e|| \tag{3.31}$$

and solved by running optimizer on an equivalent problem

$$min\ || \begin{bmatrix} S_w \\ \sqrt{(}\mu)I \end{bmatrix} L_e + \begin{bmatrix} L_w \\ 0 \end{bmatrix} || \tag{3.32}$$

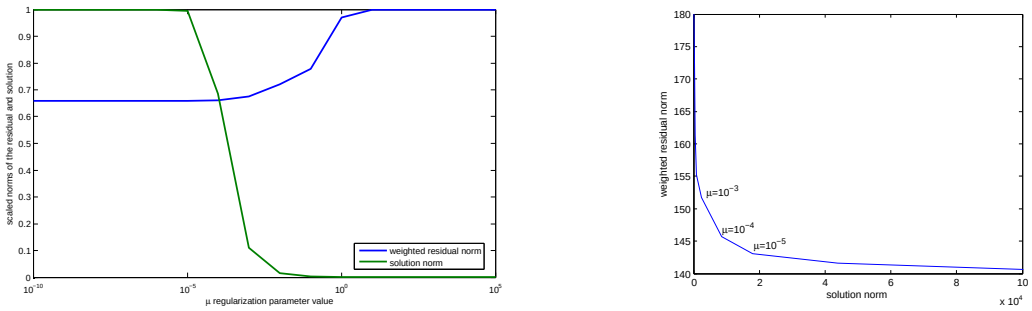


Figure 3.4: The choice of regularization weights may be driven by these plots drawn for office scene. The left plot suggests that the interesting interval for μ lies between 10^{-5} and 1. The right plot (called usually the tradeoff curve) presents solution quality as a function of solution magnitude. We pick μ corresponding to the solution norm of 10^4 as further decreasing of μ does not lead to significantly better residual, but much larger solution.

Figure 3.5: Sample scene rendered with different regularization parameters. From left: $\mu = 0, 1, 10^4$. Higher regularization makes more local lighting by dimming any unnecessary lights at the cost of worse fit in important regions. Note that the tone mappings for the images are different.

The choice of how much the magnitude of the result will affect the optimization is steered by μ, which is usually chosen empirically. Figure 3.4 shows the plots that were used for determining μ for one of the sample scenes.

Regularization (with average, not very small μ) has a useful side-effect – it makes the problem 3.32 matrix full rank.

Chapter 4

Implementation details

Here we will describe technical details behind our system.

4.1 PRT implementation

4.1.1 Positioning caches on surfaces

As in [LZT+08], in order to place caches on the surfaces we shoot rays through the scene. The rays are shot from initial camera position and each bounce produces a cache if only it is distant enough from the previously added caches. This minimum cache distance is configured by the user and the metric used is defined by equation 2.19. To get more uniform distribution we omit first three bounces.

The sampling strategy described is called Poisson disk sampling, but is of course also biased towards points more visible from the initial view point. Even omitting the first bounces, won't not help the algorithm pick samples from the other side of a narrow gap. Other idea would be to fill the gap with a portal (not visible surface) and generate samples on the portal. Irradiance or radiance in the caches on side A of the portal would be treated as light sources on side B and vice versa. This approach would be iterative for hopeful convergence.

4.1.2 Searching for the nearest cache

During cache position generation, after each bounce, we add a new cache if it is distant enough from the previously added caches. We therefore seek for the nearest neighbour of the candidate. To perform the search fast we place each added cache into a kd-tree, which is a data structure partitioning the 3-dimensional space subsequently using axis aligned hyperplanes. We would like the tree to be balanced, but balancing it during each addition is costly. We do a lazy reconstruction of the whole tree every u iterations, while keeping the unsorted tail of the new cache samples as an array.

Cost analysis of u iterations following i-th iteration:

binary search of the nearest neighbour in a tree	$u\ \log_2(i+au)$
linear search of the nearest neighbour in an array	$u^2/2$
balancing	$(i+u)\ \log_g(i+u)$

Cost of one iteration is therefore approximately (we omit u terms under $\log_2$):

$$\log_2 i + u/2 + \frac{i+u}{u}\log_2 i$$

Let us fix i and compute minimum over u. Minimum is achieved for

$$u_{min} = \frac{\sqrt{2}\sqrt{i \cdot ln(2) ln(i)}}{ln(2)} \approx 1.698\sqrt{i\ lni}$$

which gives the optimal number of iterations before reconstructing the tree. More detailed analysis would require measuring the constant factors related to each operation.

4.1.3 Converting area lights to caches

Our system optionally reads lights from the scene. Lights are converted to caches in the following way: each light-emitting polygon is replaced by a set of caches belonging to this polygon. If no cache belongs to this polygon, then one nearest cache is chosen. Each cache behaves as a point light. If polygon has radiance E and area A, then each cache receives radiance of EA/n where n is the number of caches within this polygon.

When lights are given and are relatively small and floating in empty space, then on cache generation it may happen they are not hit at all – no cache will be placed on light polygons. To prevent this situation we add some samples on lights first (using rejection sampling according to the metric 2.19) and then proceed with random adding caches to the whole scene by raytracing.

4.2 Rendering

Figure 4.1: Visualization of cache positions along with surface normals in the hit point.

As the number of lights (caches) is about 1000, same as [LZT+08], we use deferred shading to display the result. The technique involves rendering to textures all the components needed to compute the lighting reaching the camera. We render surface normals, surface position in 3D view space and surface material properties (diffuse and specular coefficients describing the reflectivity) prior to computing the lighting reaching the camera, thus deferring the computation and rendering the scene only once for all thousands of light caches stored. New graphics hardware architectures support rendering to many render targets at once.

Technically, normals are stored in a 32-bit 2D surface of integers (10 bits for each of three dimensions and two-bit padding) and position is stored on 64-bit surface of integers (16 bits for each of three dimensions and fourth dimension not used). The values must be scaled to $[0, 1]$ interval before writing to an integer texture in fragment shader. For storing depth a standard 32-bit depth buffer is used. These surfaces are loaded as either render targets or textures spanning the whole screen, with one-to-one texel-pixel correspondence.

To compute the lighting we render each cache one by one. Each cache is rendered as a rectangle (quad) centered in the cache position, but caches outside the view frustum are excluded by frustum culling. We use geometry shaders to make quads from points directly on the graphics hardware. Each point of the quad is assigned precomputed lighting stored within the cache and proper weight decreasing radially further from the cache and quad center. This light is combined with normal and other stored material properties (taken from textures) to compute the final lighting. Lighting from each subsequent quad is blended additively with the previously computed and stored on a high dynamic range (HDR) canvas. The HDR canvas is 32-bit texture storing low-precision floating point values – 11 bits are used for red and green and the remaining 10 for blue channel.

```
render deferred lighting components onto
    normal canvas
    position canvas
    diffuse color + transparency canvas
    specular color + power canvas
    depth buffer
splat light caches and light sources on
    HDR canvas
    weight canvas
render brush stroke on
    HDR canvas
    depth buffer
display
    normalization
    tone mapping
    slight diffuse shading
helpers
    display cache positions
    display normals
    display interface
    display texture contents
```

Algorithm 1: Rendering algorithm

Cache weights are also added onto yet another texture, for the next step performs the normalization of lighting (dividing by weight), tone mapping and other necessary post-processing. Sample post-processing effect would be to slightly shade the objects with directional light from the camera and omnipresent (ambient) light, so that the scene is visible even in the initial phase of the lighting design, when there is no defined lighting and the scene would otherwise be black.

The brush strokes are drawn directly onto the HDR canvas due to the stroke power in radiometric units (before tone mapping). Moreover thick strokes are drawn in three dimensions, using 3D distance around the clicked point. To prevent stroke intensity from being weighted during the later display phase, it is multiplied by current weight accumulated under the HDR canvas.

Instead of displaying source lights or lighting in caches, a solution error may be displayed. The error is computed in radiometric units and displayed for the same tone mapping as the solution would be, multiplied by a factor given by the user.

Optionally we can display some rendering artifacts to the user. The intermediate textures (such as fullscreen normal buffer or position buffer) are displayed in a separate user interface pass.

4.2.1 Tone mapping

The first approach was to simply multiply by exposure, which was inverse 80 percentile of the maximum radiance in caches. The radiance was taken from visible caches, because frustum culling is performed in software. Assuming most of the materials are diffuse, we also divide normalizing term by π (integrated cosine-weighted hemisphere) to compensate for computing average on stronger cache intensities, rather than output radiance.

Second approach was to use global only Reinhard's operator from [GWWH05].

$$L_d(x,y) = \frac{L(x,y)}{1+L(x,y)} \qquad L(x,y) = \frac{a\pi}{\bar{L_w}} L_w(x,y)$$

Where a is a photographic key value (0.18 used), L_w is the pixel radiance, L_d is the pixel value after tone mapping and $\bar{L_w}$ is the log average of the visible cache radiance[1].

[1] Computation of the log average (geometric mean) may be unstable for sample values below one, we therefore compute $\bar{L_w} = exp(\sum_i ln(1 + sL_w^i)/n)$ where $s = 1000 max_i(L_w^i)$, thus avoiding the $[0, 1]$ arguments for log.

To maintain temporal coherence the system linearly interpolates new $\bar{L}_{w,i+1}$ with that of the previous frame $\bar{L}_{w,i}$.

$$\bar{L}_{w,i+1} := \bar{L}_{w,i} + 0.25(\bar{L}_{w,i+1} - \bar{L}_{w,i})$$

4.3 Optimization

The implemented NNLS algorithm solves least squares problem, by partitioning variables (matrix columns) into positive (P set) and equal to zero (Z set). We start by putting all variables into Z and adding these to P one by one, each time minimizing

$$A_P x_P - f$$

Instead of solving linear least squares every time (using QR decomposition), we build QR decomposition incrementally by augmenting $A_P = QR$ with the new column, or reducing the column. Both procedures involve standard Householder transformations and are also described in [LH74]. To preserve sparseness of the decomposition, we do not compute matrix Q explicitly. Instead we keep only the Householder vectors placed in bottom-left triangle of matrix R. As the vectors greatly resemble the source matrix A, the sparseness is preserved.

Chapter 5

Results and summary

5.1 Measuring the error.

While optimizing for lights across the whole scene and in principle not knowing the renderer or tone mapping operator, we are forced not to rely on perceptual measures. The error which is displayed is in contrary subject to tone mapping, which may impose a judgment that the solution is indeed worse than expected.

The error was measured using objective quantities, like residual norm and (when with regularization) solution norm. Additionally we used the average signal to noise ratio (SNR), which we computed as follows:

$$E_{SNR} = \frac{1}{n} \sum_{i=1}^{n} \frac{|l_i|}{|w_i r_i|}$$

Where the sum is over n caches, which have highest importance (above 0.5) and nonzero desired outgoing radiance l_i. r_i is the residual error of the solution in the cache and w_i is the cache importance (less important caches receive more tolerance). We expect high SNR (order of 100) meaning that error is below 1%, which is commonly used light perception threshold.

For testing purposes we used a scene with cubic room, of which one wall acted as a light source projector, and the opposite was painted uniformly. The initial painting as done with thin stroke-relighting brush and resulted in some fluctuations in the error (most of which remained hidden by the dense cache placement), which gave $E_{SNR} > 30$ (figure 5.1). Painting with simple (not optimizing brush) can sometimes produce smoother error function, but not necessarily higher quality result, which was the case in the described scene.

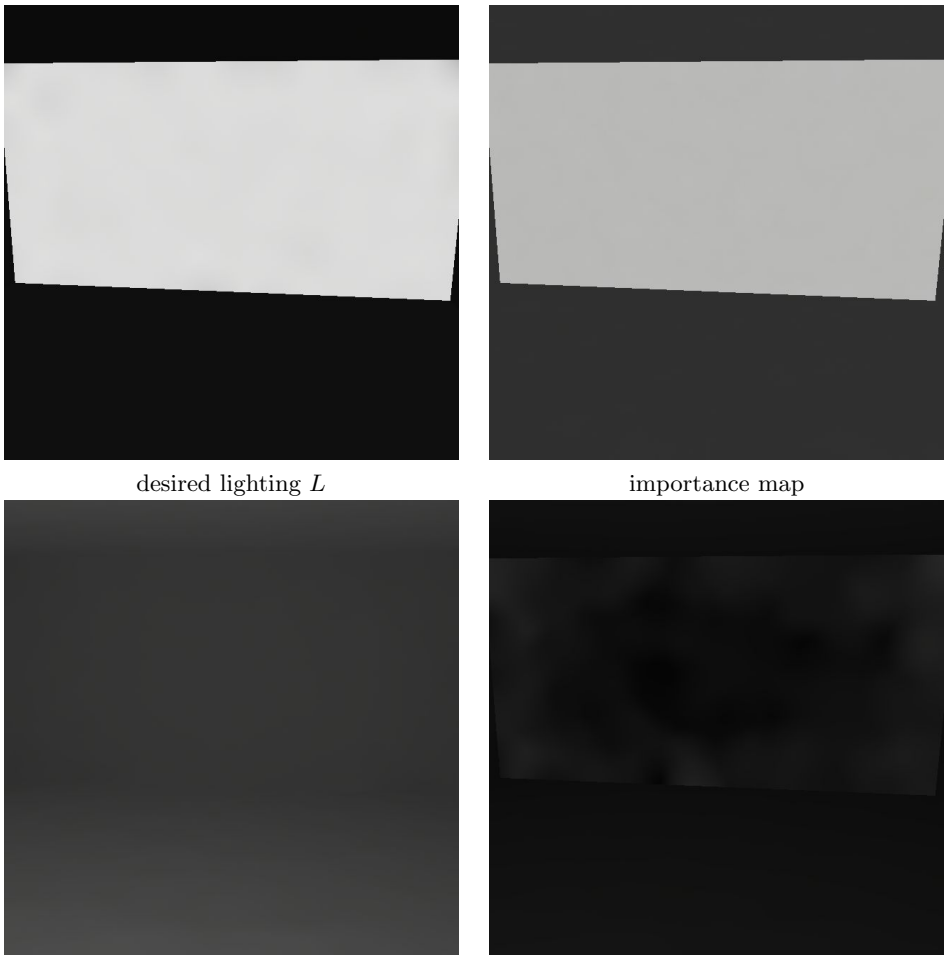
5.2 Results

property	value
rays/receiver cache	1000
maximum order of Neumann expansion	6
not painted importance v	0.05
importance history δ	0.9
NNLS threshold	1e-7
solution precision	double

Table 5.1: The global settings used for all the test scenes.

All results were computed on a Core2 Duo machine using a single core, while the frame rates are given for Nvidia 8600M GT GPU. The table 5.1 summarizes the global settings used.

Five scenes were used for the testing purposes. The table 5.2 groups the per-scene settings, data sizes, performance and quantitative solution quality measures. Figures 5.2, 5.2, 5.2, 5.2 and



desired lighting L | importance map

computed lighting SL_e | weighted error $S_w L_e - L_w$

Figure 5.1: The projector scene for which we achieved $E_{SNR} > 30$. The upper left image represents the desired lighting on one wall (the others were not painted, as seen on the importance map – the top right image). The opposite wall serves as a source light accumulator. The bottom right image shows the solution error with tone mapping of the *resulting* lighting, which is shown as the bottom left image. The ideal error image should be black.

5.2 contain the images of the painted input lighting (top left), the inferred importance map (top right), the resulting lighting SL_e (bottom left) and the residual error (bottom right) with the tone mapping of the resulting lighting. The residual error on the images is always weighted by the importance and given by $S_w L_e - L_w$, where S_w and L_w were defined for the optimization 3.30. The details on light placement are given below each figure. For each scene, after the computation, the user can browse through proposed lights to discover assigned intensity and color. The lights are also seen in the scene and where intensity can be easily retrieved.

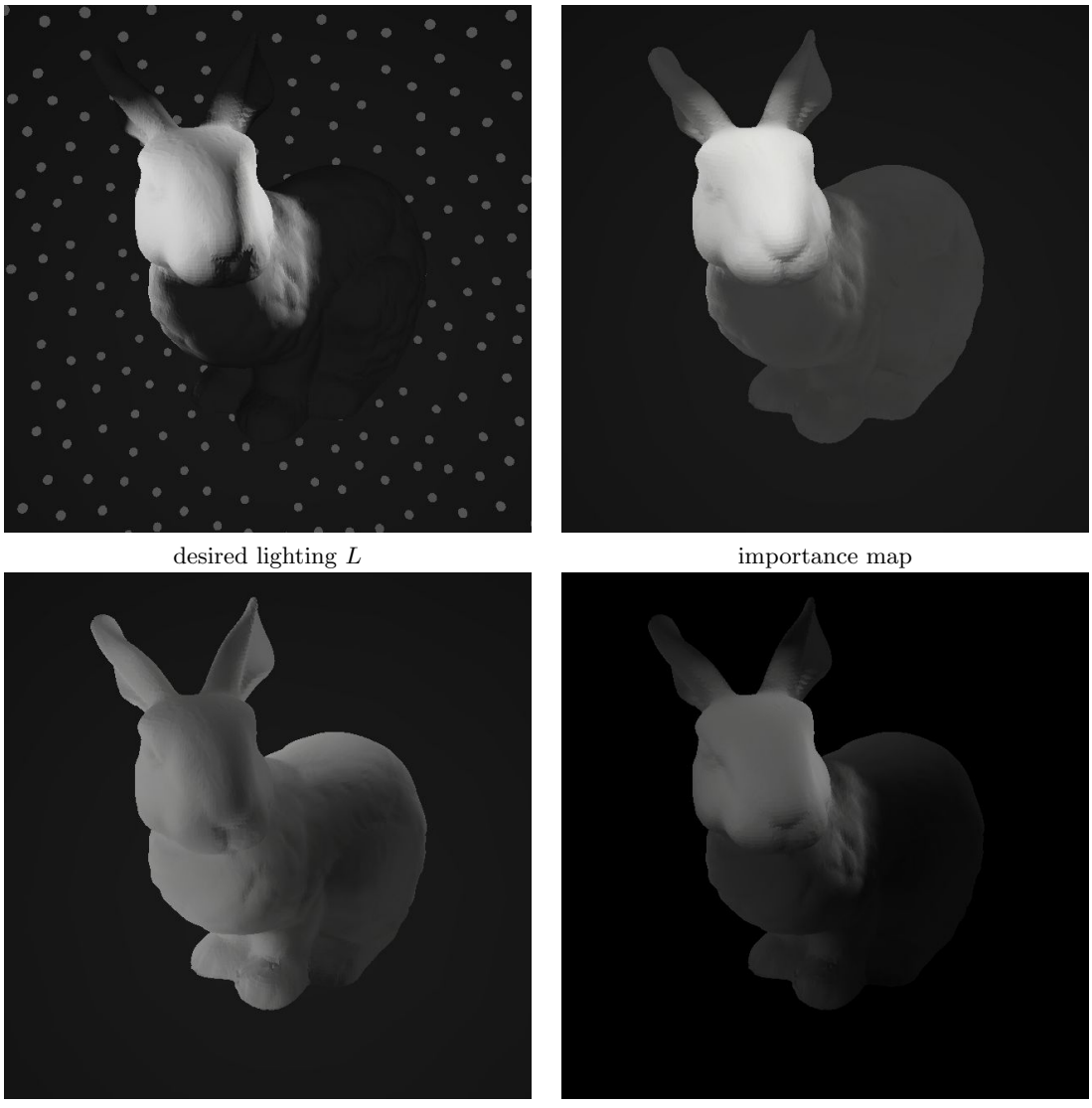
scene	bunny	cabin	maze	office	sibenik
regularization tradeoff μ	0	10^{-8}	10^{-8}	10^{-4}	10^{-8}
cache size	0.5	0.7	2	0.3	6
cache overlap	1.5	1.5	1.5	1.5	1.5
number of caches	2284	3702	7253	3289	2978
number of lights	2067	50	63	54	14
S conditioning	9299	23731	17582	22995	24084
S sparseness	0.59	0.53	0.22	0.92	0.65
S memory	23.45MB	55.08MB	88.77MB	75.63MB	43.96MB
S_{NNLS} sparseness[1]	0.61	0.52	0.21	0.95	0.58
S_{NNLS} memory[1]	21.83MB	0.75MB	0.73MB	1.31MB	0.19MB
S_{QR} sparseness (avg)[2]	0.74	0.92	0.67	1.00	0.79
fps	7.77	3.46	26.48	19.05	2.61
precomputation time t_{pre}	5.8min	12.686min	33.31min	63.70min	10.59min
solution time t_{sol}	102.71s	6.43s	14.17s	8.72s	2.64s
NNLS iterations	13	13	12	10	12
E_{SNR}	13.65	5.47	22.84	8.63	2.65
residual norm weighted $\|\|S_w L_e - L_w\|\|_2$	124.981	6.94	30.34	26.06	14.29
residual norm $\|\|SL_e - L\|\|_2$	1697.29	44.31	362.73	84.90	70.93
solution norm $\|\|L_e\|\|_2$	1372.14	20.16	4509.17	21.59	310.04
lights used	9	15	12	8	9

Table 5.2: Settings, data sizes, performance and solution quality measures for the test scenes.

5.3 Discussion and conclusion

We implemented the system for diffuse lighting design on arbitrary geometry for interior scenes. The optimization is efficient and takes painting order (importance) and preventing overfitting (regularization) into account.

Many improvements are possible. The lighting system is strongly omnidirectional and therefore the current optimizer performs the best when whole scene has smooth lighting. Substituting omnidirectional lights with spotlights is a simplest remedy. Although we discussed extension for specular transfer case, only the diffuse transfer was the one implemented. As the problem is solved per color channel, the solution is sometimes represented by separate red, green and blue lights, or lights only in colors reflected by the scene. The matrices for all channels might been merged and another least squares objective function measuring the differences between all the channels of the same light might be added to prevent decoupling the channels. Other minor extensions include tone mapping for specific region of interest (as in [OMSI07]) and improving brush design so that it acts only on the areas with similar normals (as in [PBMF07]). Full [LZT+08] implementation with hierarchies of caches would increase rendering efficiency and allow us to consider hierarchical solution methods, like multigrid.



desired lighting L | importance map

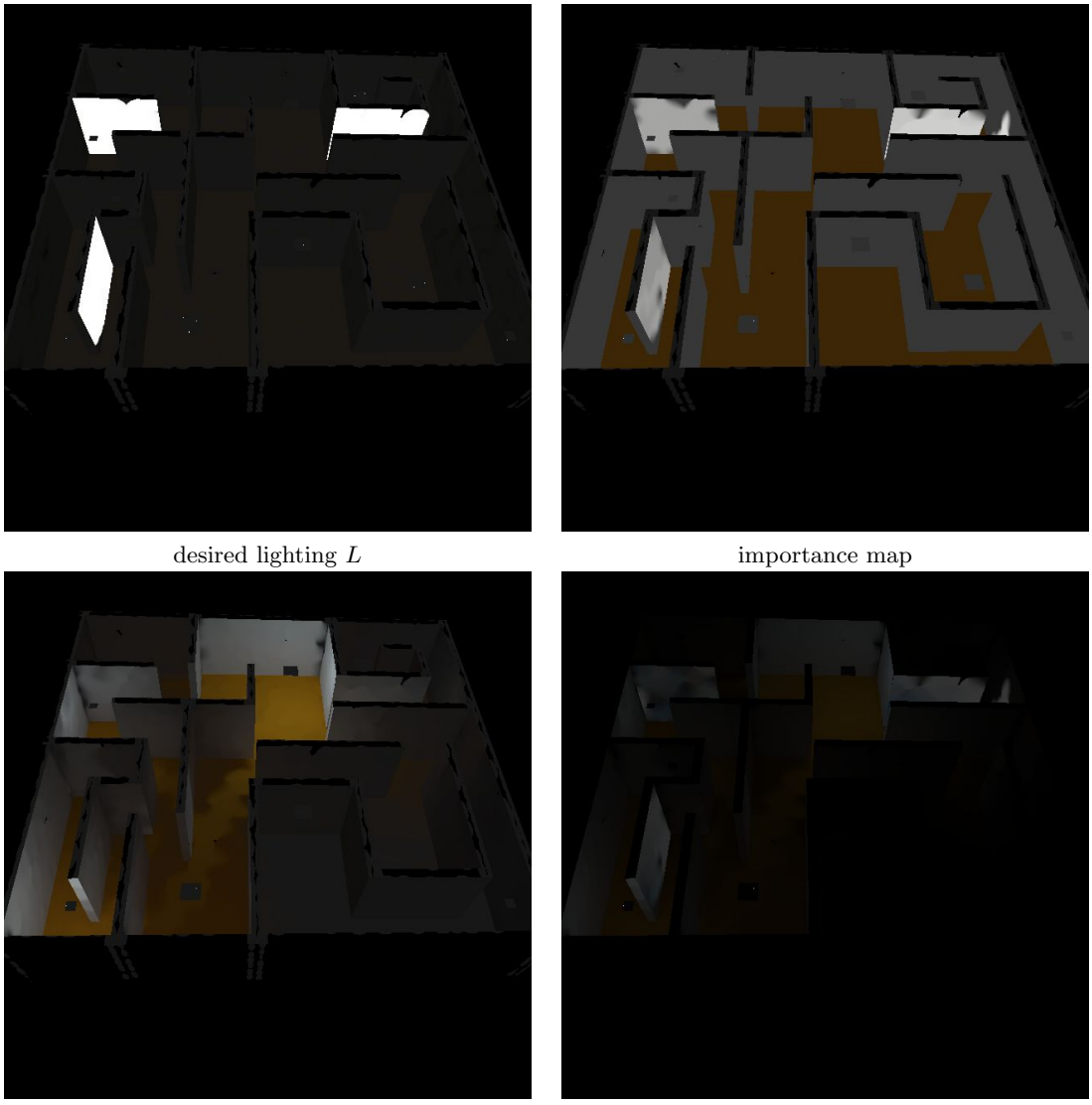
computed lighting SL_e | weighted error $S_w L_e - L_w$

Figure 5.2: The bunny scene. The lights are placed on a sphere surrounding the model, thus simulating environmental lighting. The desired and computed lighting images have similar local effects, but the latter has more background lighting, thus smoother tone mapping. Visual error (with tone mapping of computed lighting) is high.

desired lighting L | importance map

computed lighting SL_e | weighted error $S_wL_e - L_w$

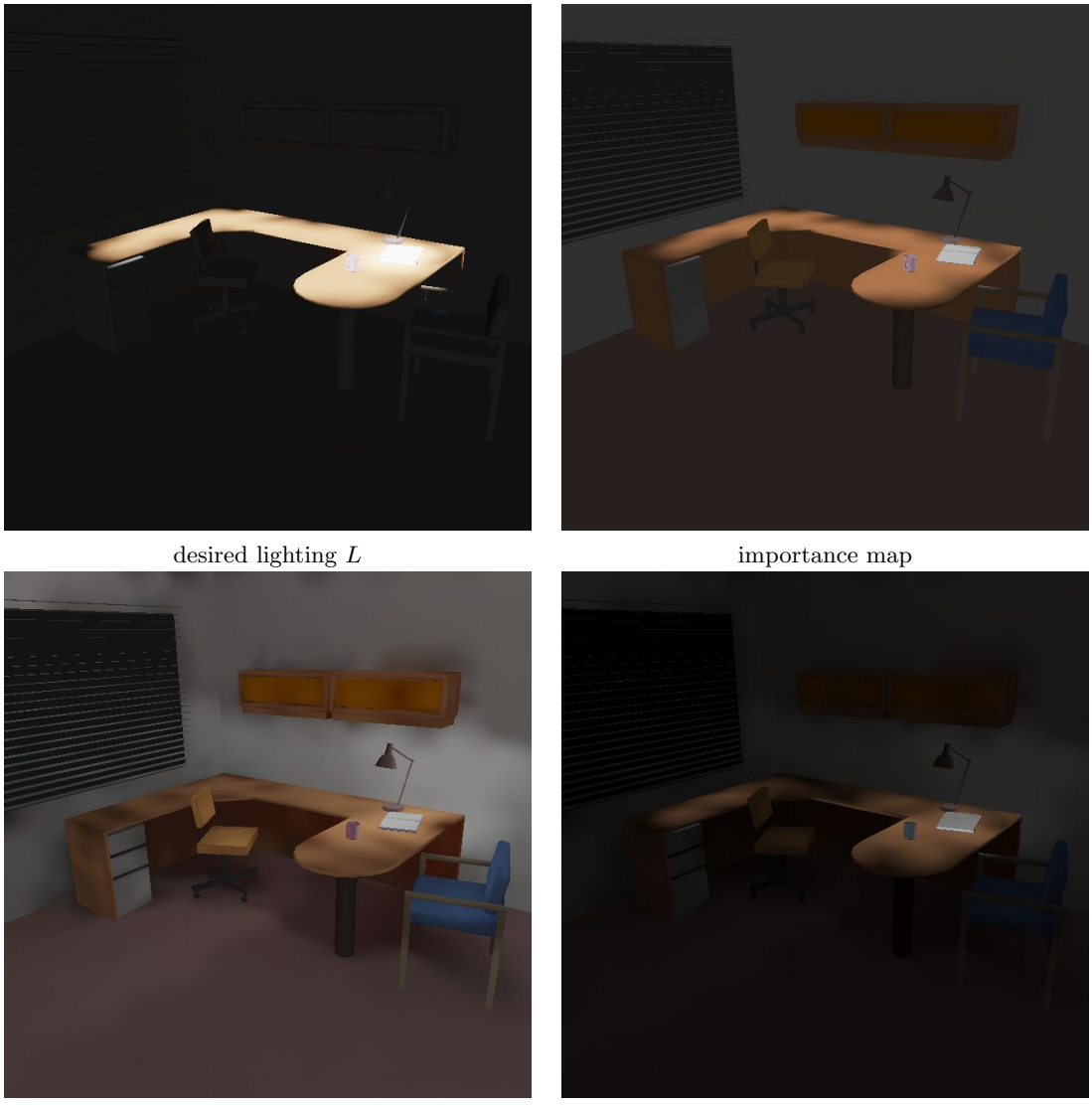
Figure 5.3: The cabin scene. The armchair and the table were painted with constant luminance. The floor was painted black. The lights were allowed on the bowls near the ceiling and additionally one light near the chimney and one behind the table. The upper left image is burned due to global-only tone mapping, but the local effects are similar to the computed lighting. Error is lower on the floor center and some surfaces of the furniture exposed to light. Please see digital results for reference.



desired lighting L | importance map

computed lighting SL_e | weighted error $S_w L_e - L_w$

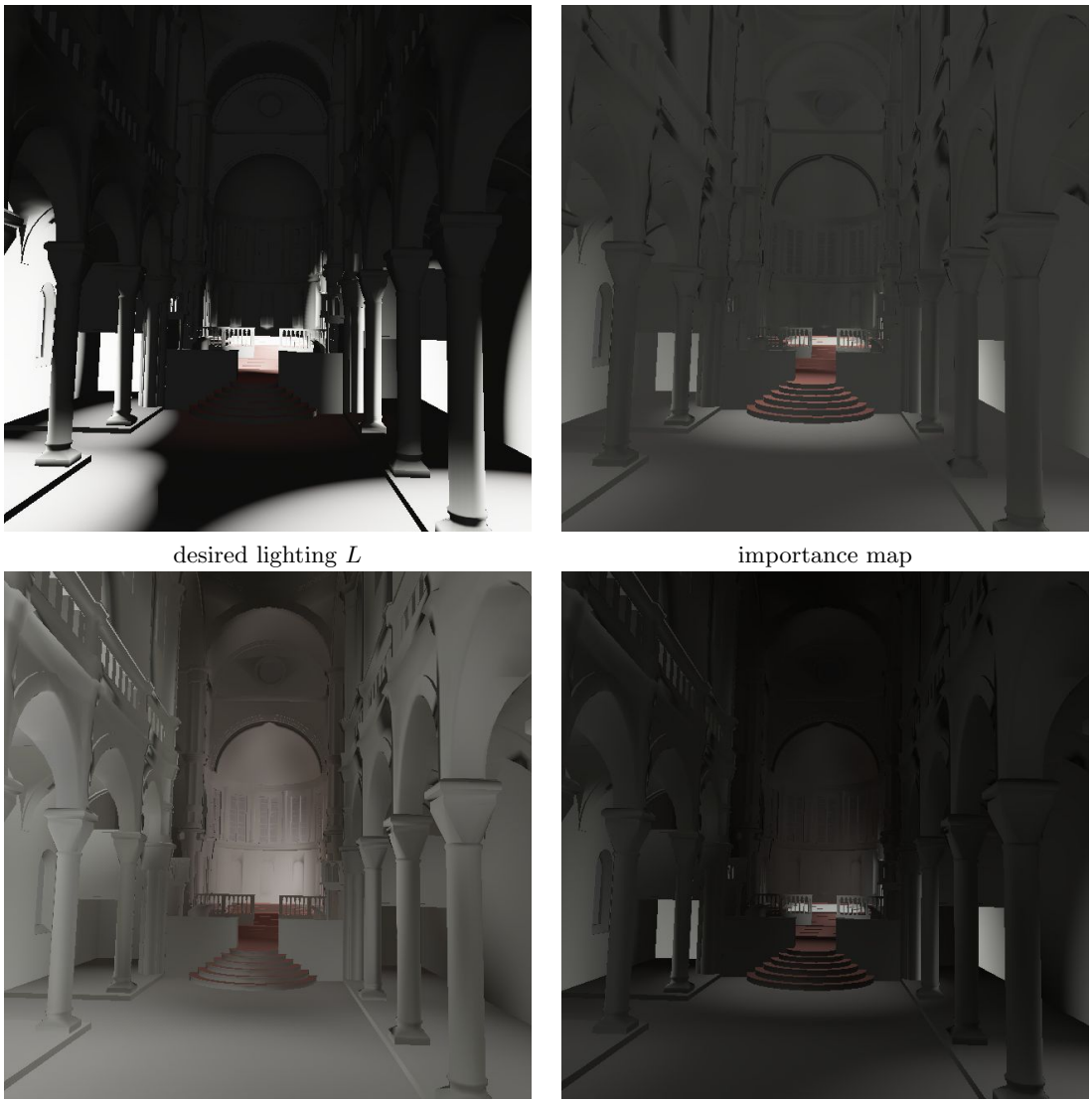
Figure 5.4: The maze scene. The lights are placed on the ceiling level. Three walls were assigned lighting and a floor segment was painted black. The reconstructed intensity in the wall centers match, which is visible as black areas on the error image.



desired lighting L | importance map

computed lighting SL_e | weighted error $S_w L_e - L_w$

Figure 5.5: The office scene. Lights are placed on the center of the ceiling and under the shelf. Only the desk lighting was painted (including the book with twice the intensity), thus the shelf lights are used the most in the solution.



desired lighting L | importance map

computed lighting SL_e | weighted error $S_wL_e - L_w$

Figure 5.6: The sibenik scene. The light was painted onto the floor of both aisles and the chancel, while the nave was painted black. The lights were placed on the aisles' ceilings and the walls around the chancel circumference. The scene is hard to handle due to low cache resolution.

Bibliography

[CW98] Michael F Cohen and John Raymond Wallace. *Radiosity and Realistic Image Synthesis*. Morgan Kaufmann, 3 edition, 1998.

[Gre03] Robin Green. Spherical harmonic lighting: The gritty details. In *Game Developers Conference 2003*, 2003.

[GWWH05] Nolan Goodnight, Rui Wang, Cliff Woolley, and Greg Humphreys. Interactive time-dependent tone mapping using programmable graphics hardware. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*, page 180, New York, NY, USA, 2005. ACM.

[HMS09] Robert Herzog, Karol Myszkowski, and Hans-Peter Seidel. Anisotropic radiance-cache splatting for efficiently computing high-quality global illumination with light-cuts. *Computer Graphics Forum (Proc. EUROGRAPHICS)*, 28(2):259–268, 2009.

[HT00] Nicholas J. Higham and Françoise Tisseur. A block algorithm for matrix 1-norm estimation, with an application to 1-norm pseudospectra. *SIAM J. Matrix Anal. Appl.*, 21(4):1185–1201, 2000.

[HV03] B. Hassibi and H. Vikalo. Maximum-likelihood decoding and integer least-squares: The expected complexity. *Multiantenna Channels: Capacity, Coding and Signal Processing*, 2003.

[KSL05] Jan Kautz, Peter-Pike Sloan, and Jaakko Lehtinen. Precomputed radiance transfer: theory and practice. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*, page 1, New York, NY, USA, 2005. ACM.

[Kŏ5] J. Křivánek. *Radiance Caching for Global Illumination Computation on Glossy Surfaces*. Ph.d. thesis, Université de Rennes 1 and Czech Technical University in Prague, December 2005.

[LH74] C.L. Lawson and R.J. Hanson. *Solving Least Squares Problems*. Prentice-Hall, 1974.

[LZT+08] Jaakko Lehtinen, Matthias Zwicker, Emmanuel Turquin, Janne Kontkanen, Frédo Durand, François X. Sillion, and Timo Aila. A meshless hierarchical representation for light transport. *ACM Trans. Graph.*, 27(3):1–9, 2008.

[Mar98] S.R. Marschner. *Inverse Rendering for Computer Graphics*. Ph.d. thesis, Cornell University, August 1998.

[OKP+08] Juraj Obert, Jaroslav Křivánek, Fabio Pellacini, Daniel Sýkora, and Sumanta Pattanaik. icheat: A representation for artistic control of indirect cinematic lighting. *Computer Graphics Forum*, 27(4):1217–1223, June 2008.

[OMSI07] Makoto Okabe, Yasuyuki Matsushita, Li Shen, and Takeo Igarashi. Illumination brush: Interactive design of all-frequency lighting. *Computer Graphics and Applications, Pacific Conference on*, 0:171–180, 2007.

[PBMF07] Fabio Pellacini, Frank Battaglia, R. Keith Morley, and Adam Finkelstein. Lighting with paint. *ACM Trans. Graph.*, 26(2):9, e 07.

[RH01] Ravi Ramamoorthi and Pat Hanrahan. A signal-processing framework for inverse rendering. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 117–128, New York, NY, USA, 2001. ACM.

[RMB07] Ravi Ramamoorthi, Dhruv Mahajan, and Peter Belhumeur. A first-order analysis of lighting, shading, and shadows. *ACM Trans. Graph.*, 26(1):2, 2007.

[SDS+93] Chris Schoeneman, Julie Dorsey, Brian Smits, James Arvo, and Donald Greenberg. Painting with light. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 143–146, New York, NY, USA, 1993. ACM.

[SKS02] Peter-Pike Sloan, Jan Kautz, and John Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 527–536, New York, NY, USA, 2002. ACM.

[TS06] Yu-Ting Tsai and Zen-Chung Shih. All-frequency precomputed radiance transfer using spherical radial basis functions and clustered tensor approximation. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 967–976, New York, NY, USA, 2006. ACM.

[WRC88] G. J. Ward, F. M. Rubinstein, and R. D. Clear. A ray tracing solution for diffuse interreflection. In *Proceedings of SIGGRAPH*, pages 85–92, 1988.