EUROGRAPHICS Workshop on ... (200x), pp. 1–8
N.N. and N.N. (Editors)

# Efficient visualization of lit automotive tail lights

S. Kniep[1] and S. Haering[2] and M. Magnor[1]

[1]Technische Universität Braunschweig, Germany
[2]Volkswagen AG, Germany

**Abstract**

*We present a new method for estimating the radiance function of lit automotive tail lamps. The method is based on Jensen's photon mapping algorithm. In order to capture high angular frequencies in the radiance function, we incorporate the angular domain into the density estimation. However, density estimation in position-direction space makes it necessary to find a tradeoff between the spatial and angular accuracy of the estimation. We identify the parameters which are important for this tradeoff and investigate the typical estimation errors. We show how the large data size, which is inherent to the underlying problem, can be handled. The method is applied to different automotive tail lights. It could also be applied to other real-world light sources.*

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.7]: Three-Dimensional Graphics and Realism—

**Keywords:** complex light sources, density estimation, photon mapping, spatial data structures

## 1. Introduction

Nowadays, the lit appearance of automotive tail lamps is an important part of the car's overall design. The development of tail lamps is involved and typically, multiple iterations are necessary to find a design that fulfills the requirements of designers, engineers and executives. To reduce the costs for expensive hardware prototypes, the lit appearance of the tail lamp might be evaluated by virtual prototyping.

Light that is emitted from the filament of a tail lamp is often reflected and scattered multiple times before it leaves the lamp. The surface of the filament is small. This renders the problem unsuitable for backward raytracing, because only a small fraction of rays will hit the filament before reaching the reflection limit. The radiance could be estimated with bidirectional path tracing [LW93], but path tracing is time-consuming for strongly varying radiance distributions. The radiance distribution of the tail lamp has such variations if the filament is visible directly or as a mirror reflection.

There are two important requirements for tail light visualization. The first requirement is physical accuracy. Designers and engineers must be able to base design decisions on the virtual tail lamp. Characteristics of the radiance function, like specular highlights and diffuse glow caused by scattering on the reflector, must be reproduced correctly. The second requirement is computational efficiency. To present a

**Figure 1:** *Visualization of the Volkswagen Touran tail lamp. The image is composed of two separate parts. One part shows the radiance that is emitted by the tail lamp, the other contains a ray-traced visualization of the unlit lamp. The emitted radiance has been estimated with our method.*

new tail light to the management, animations are created that show the tail light from different directions and distances. Since many images of the tail light are needed for an animation, the demand for efficiency is high.

In this paper, a method with these properties is presented. It is based on Jensen's photon mapping algorithm [Jen96]. In a preprocessing step, particles are traced from the filament through the inner parts of the lamp until they reach a surface that encloses the lamp. When a particle passes this surface, its position, direction and flux are saved. A *k*d-tree is build that allows to select particles depending on their position and direction. The *k*d-tree is used to perform a four-dimensional kernel density estimation of the radiance function. With this estimation, it is possible to capture high spatial as well as high angular frequencies of the radiance function.

## 2. Related Work

The method presented in this paper is based on the photon mapping algorithm [Jen96]. Photon mapping is typically used to compute the indirect illumination on non-specular surfaces. The algorithm starts by tracing virtual photons through the scene. Interactions of photons with non-specular surfaces are stored in a data structure called the photon map. For each interaction, the flux, position and incoming direction of the photon are saved. After building the photon map, the contribution of the photons to the reflected radiance on a surface can be estimated from the photons by kernel density estimation. This density estimation is based on the $k$ photons that are closest to the evaluation point $x$ on the surface.

Specular reflections are not saved in the photon map because a photon that is reflected from a specular surface contributes little to the reflected radiance in all but the direction of the mirror reflection. Hence, the $k$ nearest photons might have little influence on the radiance that is being estimated. To estimate directional radiance distributions as they result from specular reflections or transmissions, the direction of the photons has to be taken into account in the density estimation. This has been done by Moon and Marschner to simulate multiple scattering in hair [MM06]. Because scattering in hair often roughly preserves the direction of incoming rays, the radiance function of illuminated hair is highly directional. This is also true for the radiance function of a typical tail light.

## 3. Photon Tracing

Like in the photon mapping algorithm, the first step of our method is to generate virtual photons at the light sources. For most tail lamps, the actual light source is the filament of an incandescent lamp, but LED and fluorescent lamps are also possible. Each photon has three parameters: flux, position and direction. The generation of photons for real-world light sources is described in [Jen01].

The photons are traced through the inner parts of the tail lamp until they pass a surface that encloses the lamp. This surface is defined by the user. It is convenient to use the outer surface of the tail lamp. The tracing process is carried out like in the photon mapping algorithm. At each interaction of a photon with a surface, the photon is either reflected, transmitted or absorbed. Russian roulette is used to choose one of these possibilities. When a particle passes the bounding surface, its position, direction and flux at the time of intersection are saved. We employ a commercial software package for the photon tracing which supports measured light sources.

The set of all photons represents the radiant flux on the bounding surface. We denote this representation as *flux map*. The flux map is different from the photon map in some ways. It represents emitted flux, whereas the photon map represents incoming flux on non-specular surfaces. In the flux map, directions are saved as 3D vectors of four-byte floating point values. In the photon map, directions are saved as a two-byte discretization of the spherical coordinates. The flux map has a higher directional resolution than the photon map, but also consumes more memory.

## 4. Radiance estimation

In photon mapping, the radiance at $x$ in the direction of $\vec{\omega}$ is estimated as a function of the flux that is carried by the $k$ photons that are closest to $x$. Each flux value is scaled by the value of the BRDF for the corresponding direction. If the BRDF has strong peaks, it is likely that only a few of the photons contribute significantly to the reflected radiance in the direction of $\vec{\omega}$, because the value of the BRDF is small for most directions. This leads to a high variance in the estimate. For this reason, specular reflections are not saved in the photon map, but are handled by ray tracing instead.

Saving specular reflections in the photon map is analogous to representing a directional radiance function with the flux map: The radiance function strongly depends on the direction of the photons. As pointed out by Moon and Marschner [MM06], this situation can be handled by taking the direction of the photons into account in the density estimation. This means that the influence of a single photon on the estimated radiance has to depend on the position as well as the direction of the photon. The contribution of a photon to the estimated radiance has to decrease with increasing spatial and angular distance. To accomplish this, a distance function is needed that represents both – spatial and angular distance. Moon uses a weighted maximum of the spatial and angular distance.

### 4.1. Distance in position-direction space

We choose an euclidean metric to measure the distance in position-direction space. Let

$$d(p,q) = \sqrt{\|x_p - x_q\|^2 + (\lambda\|\vec{\omega}_p - \vec{\omega}_q\|)^2}, \qquad (1)$$

be the distance between two points $p = (x_p, \vec{\omega}_p)$ and $q = (x_q, \vec{\omega}_q)$ in position-direction space, with $\|\vec{a}\|$ denoting the magnitude of the vector $\vec{a}$. The angle between $\vec{\omega}_p$ and $\vec{\omega}_q$ is approximated by the magnitude of the vector difference between $\vec{\omega}_p$ and $\vec{\omega}_q$. The direction vectors are multiplied by a weight factor $\lambda \geq 0$ to control the relative importance of spatial and angular distance. The influence of the angular distance on $d$ grows with $\lambda$. In the following, the distance $d(p,q)$ between two points is called their $d$-distance.

$d(p,q)$ continuously grows with the spatial distance and the angle included between $p$ and $q$. This ensures that the estimated radiance function is smooth if the kernel function is smooth. Photons with a high spatial but low angular distance may have the same $d$-distance as photons with a high angular but low spatial distance.

The error of the approximation $\sphericalangle(\vec{\omega}_p, \vec{\omega}_q) \approx \|\vec{\omega}_p - \vec{\omega}_p\|$ grows with the angle $\sphericalangle(\vec{\omega}_p, \vec{\omega}_q)$. For an angle of $10^\circ$, the relative error of the approximation is less than $0{,}15\%$. In our tests (see section 6), this value was never exceeded. Also, this error does not necessarily lead to an additional error in the density estimation. The approximation can be seen as a slight deformation of the kernel function. The actual shape of the kernel function has little influence on the error of the density estimation [Sil86].

### 4.2. The kernel function

The kernel is a weighting function that is applied to the vector difference between the evaluation point $(x, \vec{\omega})$ and the photons within the bandwidth. The kernel function should be smooth and fast to evaluate. The smoothness property is important because discontinuities in the kernel might lead to visible artifacts (as observed for photon mapping in [Jen96]). The above requirements are met by the epanechnikov kernel (see e.g. [Sil86]). Let $\mathcal{K}$ be the 6-dimensional epanechnikov kernel without its normalization factor:

$$\mathcal{K}(\mathbf{x}) = \begin{cases} 1 - \mathbf{x}^T\mathbf{x} & \text{if } \mathbf{x}^T\mathbf{x} < 1 \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

This kernel can be applied directly to the points in position-direction space. To center the kernel at the photon $p = (x_p, \vec{\omega}_p)$, it has to be applied to the vector difference between $p$ and the point of evaluation (see below). To control the size of the kernel, the vector difference is multiplied with the bandwidth matrix

$$H = \text{diag}(h, h, h, h/\lambda, h/\lambda, h/\lambda). \tag{3}$$

For each evaluation point $(x, \vec{\omega})$, $h$ is set to the distance between $(x, \vec{\omega})$ and its $k$th nearest neighbor. $\lambda$ is the weighting factor from equation 1. The centered and scaled kernel can be written as

$$\mathcal{K}_p(x, \vec{\omega}) = \mathcal{K}\left(H^{-1}\begin{pmatrix} x - x_p \\ \vec{\omega} - \vec{\omega}_p \end{pmatrix}\right). \tag{4}$$

In $\mathcal{K}_p(x, \vec{\omega})$, the distance term $\mathbf{x}^T\mathbf{x}$ from equation 2 becomes

$$\left(H^{-1}\begin{pmatrix} x - x_p \\ \vec{\omega} - \vec{\omega}_p \end{pmatrix}\right)^T H^{-1}\begin{pmatrix} x - x_p \\ \vec{\omega} - \vec{\omega}_p \end{pmatrix}, \tag{5}$$

which is the squared $d$-distance between $(x, \vec{\omega})$ and $(x_p, \vec{\omega}_p)$. By scaling the vector difference with $H$, we have adjusted the kernel to the distance function $d$.

The kernel has to integrate to one to ensure that the integral over the estimated radiance function equals the radiant flux that is stored in the flux map. This can be achieved by multiplying the kernel with a normalization factor $m$. Note that the normalization factor of the 6D epanechnikov kernel cannot be used as $m$ because it makes the kernel integrate to one in 6D euclidean space, rather than in 4D position-direction space. A lengthy calculation shows that $m$ equals the normalization factor of the epanechnikov kernel for 4D euclidean space. Note that in this calculation, we have assumed that there is no boundary bias (see [Sco92]) at the edge of the hemisphere of directions in $x$. This assumption does not hold in practice, but we ignore the resulting error because it decreases with $h$. By including $m$, the kernel becomes

$$K_p(x, \vec{\omega}) = \frac{6\lambda^2}{\pi^2 h^4}\mathcal{K}\left(H^{-1}\begin{pmatrix} x - x_p \\ \vec{\omega} - \vec{\omega}_p \end{pmatrix}\right). \tag{6}$$

To ensure that the directional bandwidth does not exceed $180^\circ$, we require that $h \leq 2\lambda$.

To estimate the radiance at $(x, \vec{\omega})$ from the flux map, the kernel from equation 6 is applied to the $k$ photons that are nearest to $(x, \vec{\omega})$. The sum of the kernel values multiplied with the flux of the respective photons is an estimate of the radiant flux density at $(x, \vec{\omega})$. Dividing this value by the cosine of the angle between $\vec{\omega}$ and the surface normal $\vec{n}_x$ gives an estimate of the radiance at $(x, \vec{\omega})$

$$\tilde{L}(x, \vec{\omega}) = \frac{1}{(\vec{n}_x \cdot \vec{\omega})}\sum_{p=1}^{k} K_p(x, \vec{\omega})\Phi_p, \tag{7}$$

where $\Phi_p$ is the flux of the photon $p$. Like in photon mapping, a maximum bandwidth $h_{\max}$ is imposed to speed up the search.

### 4.3. The bandwidth ratio

Consider the photon $p = (x_p, \vec{\omega}_p)$ and the evaluation point $(x, \vec{\omega})$. If $\vec{\omega}_p = \vec{\omega}$, $p$ contributes to $\tilde{L}(x, \vec{\omega})$ iff $\|x - x_p\| < h$. This follows from the definition of the kernel. If $x_p = x$, $p$ contributes to $\tilde{L}(x, \vec{\omega})$ iff $\|\vec{\omega} - \vec{\omega}_p\| < h/\lambda$. This means that the maximum possible spatial bandwidth is $h$ and the maximum possible angular bandwidth is $h/\lambda$. For convenience, we refer to these parameters just as the *spatial* and the *angular* bandwidth. $\lambda$ is the ratio between the spatial and the angular bandwidth and controls the relative amount of smoothing in the spatial and in the angular domain.

If $\lambda$ is small, the angular bandwidth is large. The spatial

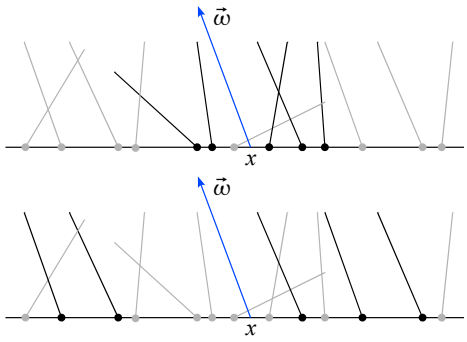


**Figure 2:** *An example of locating the five nearest neighbors of $(x,\vec{\omega})$ with two different bandwidth ratios. Top: $\lambda$ is small, so the spatial bandwidth is also small and the five nearest neighbors (shown black) lie close to $x$. Bottom: $\lambda$ is large, so the angular bandwidth is small and the directions of the nearest neighbors are very similar to $\vec{\omega}$.*

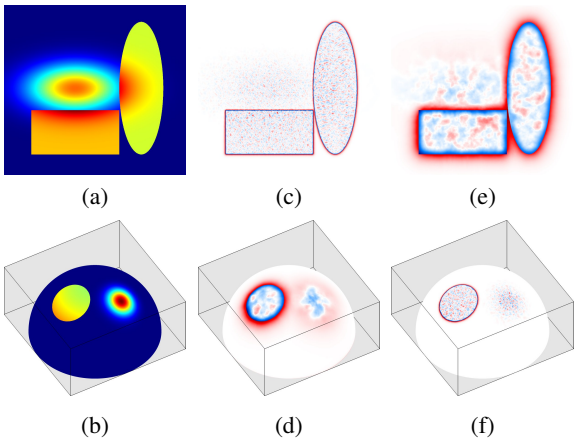


**Figure 3:** *A synthetic radiance function and error plots for two estimations of this function with a fixed number of nearest neighbors but different bandwidth ratios. (a) shows a slice of the synthetic radiance function with a fixed direction. (b) shows a slice with a fixed position, the radiance is plotted on the hemisphere of directions at this position. The plots (c) and (d) show the error of the estimation with $\lambda = 0.5$ for the same slices. Red indicates overestimation, blue indicates underestimation. (e) and (f) show the error of the estimation with $\lambda = 20$. Due to the increase of $\lambda$, the main source of the error moves from spatial noise (c) to spatial bias (e) and from angular bias (d) to angular noise (f).*

bandwidth also depends on $\lambda$. For a small $\lambda$, the set of the $k$ nearest neighbors contains more photons that lie close to $x$ spatially than for a larger $\lambda$ (see figure 2). This means that the spatial bandwidth increases with $\lambda$. Thus, if $\lambda$ is small, the angular bandwidth is large and the spatial bandwidth is small. This may result in *angular bias* and *spatial noise*. We employ the term *angular bias* for a kind of bias that appears in $\tilde{L}(x,\vec{\omega})$ if $x$ is fixed and $\vec{\omega}$ changes. Similarly, *spatial noise* is the noise that appears in $\tilde{L}(x,\vec{\omega})$ if $\vec{\omega}$ is fixed and $x$ changes. If $\lambda$ is large, the angular bandwidth is small and the spatial bandwidth is large. This may result in *spatial bias* and *angular noise*, which are defined analogously. See figure 3 for an illustration of these effects.

Just like $k$ controls the overall tradeoff between bias and noise, $\lambda$ controls the tradeoff between spatial bias and angular noise on one side and spatial noise and angular bias on the other side (see figure 4). To achieve a good tradeoff between spatial and angular errors, spatial and angular frequencies of the radiance function have to be taken into account when selecting $\lambda$. The more high-frequency components are contained in the angular domain, the larger $\lambda$ should be (assuming a fixed spectrum in the spatial domain). Since the frequency spectrum of the radiance function is not known, we need to determine $\lambda$ empirically by trying different values and comparing the results. This needs to be done once for each tail light.

The distance $l$ between the camera and the tail light also should be taken into account when selecting $\lambda$. This is due to the constellation of the six-dimensional evaluation points that result from a certain distance $l$. Assume that the evaluation points are determined by a ray tracer. Each ray that intersects the bounding surface of the tail lamp defines an evaluation point $(x,\vec{\omega})$, with $x$ being the intersection point and $\vec{\omega}$ being the inverted direction of the intersecting ray.

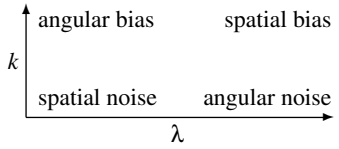


**Figure 4:** *Schematic diagram of how the tradeoff between bias and noise can be controlled by k and $\lambda$.*

The larger $l$, the farer apart from each other are the intersection points. And the larger the distance between two adjacent evaluation points, the larger is the possibility that the photon density differs significantly from one point to the other (which is perceived as spatial noise). This effect can be counteracted by increasing the spatial bandwidth with the spatial distance between the evaluation points. This in turn means that $\lambda$ has to be increased with $l$.

The situation is similar in the angular domain. Imagine that the camera is rotated around the tail light in an animation. The larger the rotation angle $\alpha$ of the camera between two successive images, the larger is the risk of angular noise. Angular noise can be seen as random differences between the two images that show the tail light before and after rotating the camera. To reduce this kind of noise, the angular bandwidth must be increased with $\alpha$. The expected value of

$\alpha$ depends on $l$. The smaller $l$, the larger is the change of $\alpha$ that results from a fixed-length movement of the camera. This means that we can expect a larger change of $\alpha$ if $l$ is small. Consequently, the angular bandwidth should be increased when $l$ becomes smaller. This also amounts to the above finding that $\lambda$ should be increased with $l$.

Note that the parameters which control the image formation in the human eye depend on $l$ in a similar way. With growing $l$, the surface area of the tail lamp that shines on a single rod becomes larger. This corresponds to a large spatial bandwidth. With decreasing $l$, the direction range of light rays that may enter the pupil (and hence may hit a rod) becomes larger. This corresponds to a large angular bandwidth.

We have tried to derive the relationship between $l$ and $\lambda$ from a simple model of the human eye. Our results are that $\lambda$ should be proportional to $l^2$. It turns out that in practice, this relationship results in extreme values for $\lambda$, which lead to biased and noisy estimations. For this reason, we have varied the influence of $l$ and compared the results visually. We have achieved good results if $\lambda$ is proportional to $l^{0.7}$.

Let $\lambda_0$ be the bandwidth ratio that has been adjusted to the spatial and angular frequencies of a certain radiance function at the distance $l_0$. Define

$$c_0 = \lambda_0 l_0^{-0.7} \tag{8}$$

to be the *bandwidth factor* for this particular radiance function. For any other values of $l$, we employ the equation

$$\Lambda(l) = c_0 l^{0.7} \tag{9}$$

to determine $\lambda$. We have observed that for $\lambda \notin [20, 600]$, the estimate is often noisy and biased. For this reason, we restrict $\lambda$ to the range $[20, 600]$. In a test scene, we have chosen $\lambda = 30$ for $l = 76.4$ cm by visual comparison, which results in $c_0 = 0.288$. In an animation of this scene, $\Lambda(l)$ ranged from 27.6 for $l = 67.7$ cm to 82.6 for $l = 324.4$ cm.

If the above heuristic is used, $\lambda$ has to be tweaked only once for a particular tail light. If more images of the same tail light are computed from different distances, e.g. for creating an animation, reasonable bandwidth ratios can be computed automatically for each $l$.

### 4.4. Locating the nearest neighbors

To evaluate equation 7, the $k$ nearest neighbors of the evaluation point $(x, \vec{\omega})$ have to be located in the flux map. We use a six-dimensional $k$d-tree for this search. If $\lambda$ is fixed, standard algorithms can be used to locate the nearest neighbors in the $k$d-tree. Each photon $p = (x_p, \vec{\omega}_p)$ is represented as the six-dimensional point

$$\mathbf{x}_p = \begin{pmatrix} x_p \\ \lambda \vec{\omega}_p \end{pmatrix}. \tag{10}$$

To avoid the computation of square roots, only squared distances are used in the search routine (as suggested by Sproull in [Spr91]). The squared distance between two points $\mathbf{x}_p$ and $\mathbf{x}_q$ is $(\mathbf{x}_p - \mathbf{x}_q)^T (\mathbf{x}_p - \mathbf{x}_q)$. This is the squared $d$-distance of $p$ and $q$, which can be plugged directly into the kernel formula (equation 4).

To locate the nearest neighbors, we employ the search algorithm by Arya and Mount [AM93] without the approximation that is described in their paper. This algorithm has the important property that it visits only the tree cells that intersect the hypersphere which is centered at the search point. This approach improves the speed of the search in higher dimensions. Earlier search algorithms [Ben75, FBF77] also confine the search to the hypersphere, but require that the boundaries of each tree cell are stored in the $k$d-tree.

To reduce memory consumption, the tree is stored as an array (see Jensen's book [Jen01]). However, this representation requires left-balancing the tree, which restricts the splitting value to a small range. As the splitting dimension, we choose the one with the largest cell extent.

#### 4.4.1. Changing the bandwidth ratio

As noted in section 4.3, it might be necessary to adjust $\lambda$ to the distance between the observer and the tail light. If equation 9 is used to determine $\lambda$, each single change of this distance results in a new $\lambda$. But also if $\lambda$ is chosen manually, different values have to be tried to find a reasonable value. With existing tree search algorithms, the tree has to be build again when $\lambda$ changes. This step is costly because typically, the required number of photons is in the order of $10^8$.

A simple solution for this problem is at hand. Suppose that the $i$-th coordinate of each tree point is multiplied with a fixed value $s_i$. If the splitting values of the tree nodes with splitting dimension $i$ are also multiplied with $s_i$, the inclusion of tree points in tree cells is not affected, i.e. the tree stays valid. Each dimension can be scaled with a distinct value $s_i$.

The scaling of tree points and splitting values can be performed during the search. The points and splitting values are never actually changed, instead they are multiplied with their scaling factor each time they are used in the search algorithm. This is reasonable because typically, only a small number of points and splitting values are used in one query. In the following, this scaling of tree points and spitting values during the search is referred to as *online scaling*. The scaling also has to be applied to the query point.

It is easy to incorporate online scaling into existing $k$d-tree search algorithms. Each occurrence of a coordinate or splitting value in the implementation of the algorithm has to be replaced by the product of the coordinate or splitting value and the corresponding scaling factor. In order to search the nearest photons according to their $d$-distance, the scaling factors for the three angular coordinates are set to $\lambda$. The spatial coordinates need no scaling.

Online scaling affects the speed of the search. The cost of the additional multiplications can be neglected (see section

6). A more critical effect of online scaling is that it changes the aspect ratio (the ratio of the longest to the shortest side) of the tree cells. As pointed out by Duncan et al. [DGK99], the aspect ratio has significant impact on the efficiency of the search. The aspect ratio should not be too large, i.e. it should be bounded. However, bounding the aspect ratio is not possible in a left-balanced *k*d-tree because the splitting value is restricted to a small range.

Still, the aspect ratio can be influenced by the choice of the splitting dimension. In our implementation, the dimension with the largest cell side is chosen for splitting. This keeps the aspect ratio of the unscaled cells as small as possible (in a balanced tree). The aspect ratio of the scaled cells might exceed the aspect ratio of the unscaled cells by a factor of $\lambda$. This may slow down the search significantly (see section 6).

To counteract this effect, the direction vectors of the photons are multiplied with a value $\lambda_t$ before the tree is constructed. $\lambda_t$ should be close to the bandwidth ratios that actually will be used and has to be chosen empirically. During the search, the variable value $\lambda_v$ is used for online scaling. The resulting bandwidth ratio is $\lambda = \lambda_t \lambda_v$. If it turns out that the initial ratio $\lambda_t$ differs strongly from the actual $\lambda$, the tree can be reconstructed with $\lambda_t = \lambda$.

## 5. Implementation

To estimate a four-dimensional function like the radiance function on a surface, a large number of data points (photons) is needed. The *k*d-tree might not fit into main memory. For this reason, we have implemented a *k*d-tree that arranges the photons on secondary storage. Only the inner nodes (the splitting dimensions and the splitting values) are kept in main memory permanently. During the search, small subtrees are loaded to main memory on demand. The least recently used subtree is unloaded if a user-defined memory limit is reached. The amount of main memory that is used for the tree construction can also be limited, but this increases the construction time.

To further reduce the consumption of main memory, the bucket size (the number of photons that are stored in a leaf node) can be reduced. By doubling the bucket size, the number of inner nodes is halved. However, the search slows down if the bucket size is too large (see Friedman et al. [FBF77]). Because the tree is stored as an array, the bucket size must be a power of two. For large trees, we have achieved good results with 32 photons per bucket. With this choice, the inner nodes of a *k*d-tree with $10^8$ photons occupy less than 16 MB.

To apply our method in practice, the evaluation points have to be determined and mapped to pixel positions within an image. This can be done via ray tracing. We have written a plug-in for a commercial ray tracer that carries out the radiance estimation. During the tracing step, the photons are stored at the outer surface of the tail lamp. A triangle mesh which approximates this surface is loaded into the ray tracer as the scene geometry. The camera is arranged at the desired relative position to the tail lamp's surface.

When a ray intersects the surface, the ray tracer passes the intersection point $x$ and the direction $\vec{\omega}$ of the intersecting ray (inverted and scaled to length $\lambda_t$) to the plug-in. $(x, \vec{\omega})$ is used as the six-dimensional query point for the *k*d-tree search. Equation 7 is used to estimate the radiance. A simple logarithmic tone-mapping is applied to the radiance values. The tone-mapped value is displayed at the pixel corresponding to the intersecting ray.

## 6. Results

We have applied our method to the Volkswagen Touran and Touareg tail lamps. The reflector of the Touran tail lamp contains many small bumps that result in high frequencies in the radiance function. The reflector of the Touareg tail lamp is smoother, hence its radiance function contains less high frequencies.

Figure 5a and 5b show the upper part of the Touran tail lamp, 5c and 5d show its lower part. The Touareg tail lamp is shown in 5e and 5f. Table 1 lists parameters and performance results for these scenes. 600 MB of main memory were used for building the tree, 500 MB were used for rendering. The images contain $640 \times 480$ pixels, all renderings were carried out on a Pentium M processor with 1.7 GHz. The photon tracing was performed on a 2 GHz Opteron processor with four cores. In all tests, the maximum bandwidth was restricted to 4, the bucket size was set to 32.

Note that although the flux map of the Touareg contains fewer photons than the flux map of the Touran, we have chosen a larger number of nearest neighbors $k$ for the Touareg than for the Touran. The reason for this choice are the high frequencies in the radiance function of the Touran lamp. If a larger $k$ is used for the estimation of the Touran tail lamp, the small bright spots on the reflector (see figure 5) appear blurry due to bias. On the other side, a rather large $k$ is necessary to reproduce the smooth radiance function of the Touareg lamp without too much noise.

To verify the correctness of our estimation scheme, we compare the result of our method for the Volkswagen Scirocco tail lamp to a path-traced image in figure 6. Both images show the same basic features of the lamp's radiance function. The comparison to the bias-free reference image shows that the flux map estimation contains visible bias. Note that areas with low radiance appear smooth in the flux map estimation. In the path-traced image, these areas contain more noise, although it was rendered with 16,000 paths/pixel (almost five billion rays), which took 32 hours on a quadcore Opteron processor. The density estimation took 85.6 seconds on a Pentium M with 1.7 GHz, additional 10.5 hours were needed for photon tracing and for building the tree (which contains 160 million photons).

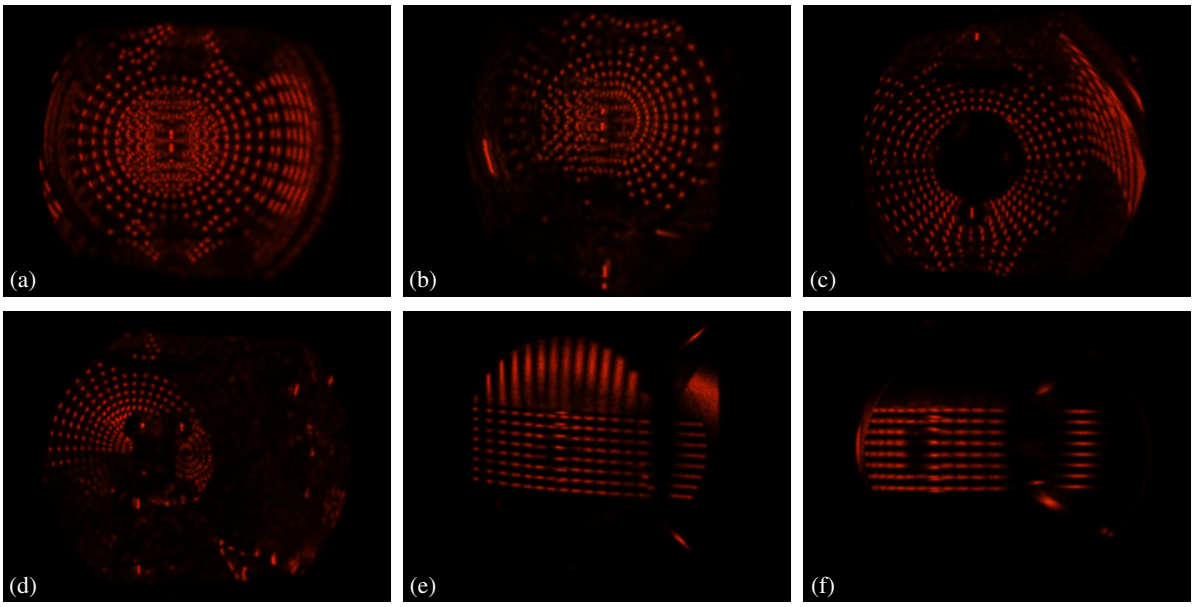


**Figure 5:** *Visualization of different parts of the Touran and Touareg tail lamps from different camera positions.*

| Scene | # Photons | Tree size on disk | Photon tracing time | Tree building time | $k$ | $\lambda$ | Rendering time | # Evaluation points |
|---|---|---|---|---|---|---|---|---|
| a | 129 mio | 3.5 GB | 261 min | 49.9 min | 20 | 80 | 69.6 sec | 251,597 |
| b | 129 mio | 3.5 GB | 261 min | 49.9 min | 20 | 80 | 68.0 sec | 233,148 |
| c | 176 mio | 4.8 GB | 473 min | 80.9 min | 20 | 80 | 75.3 sec | 228,647 |
| d | 176 mio | 4.8 GB | 473 min | 80.9 min | 20 | 80 | 83.7 sec | 244,476 |
| e | 39 mio | 1 GB | 46 min | 9.3 min | 100 | 30 | 73.0 sec | 102,122 |
| f | 39 mio | 1 GB | 46 min | 9.3 min | 100 | 30 | 81.0 sec | 85,534 |

**Table 1:** *Parameters and performance results for the Touran and Touareg scenes. The images contain* $640 \times 480$ *pixels, all renderings were carried out on a Pentium M processor with 1.7 GHz.*

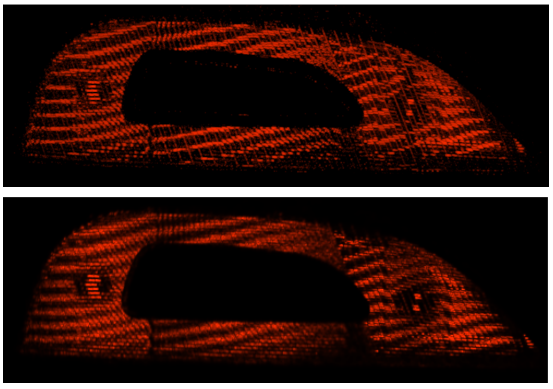

**Figure 6:** *A path-traced image of the Volkswagen Scirocco tail lamp (top) and an image of the same scene that has been generated with our method (bottom).*

To test the effects of online scaling on the search speed, we have rendered the Touran scene from figure 5a with different bandwidth ratios, using the same tree with $\lambda_t = 80$ most of the time. Table 2 shows the results of this test. Note that the rendering times for $\lambda = 79$ and $\lambda = 81$ with online scaling are virtually the same as for $\lambda = 80$ without online scaling. This shows that the additional multiplications have little impact on the search speed. The tree with $\lambda_t = 80$ allows efficient searches for a broad range of bandwidths. A tree with $\lambda_t = 8000$ results in smaller rendering times than a tree with $\lambda_t = 80$ for $\lambda = 8000$ (see section 4.4.1). A tree with $\lambda_t = 0.8$ performs only slightly better than the tree with $\lambda_t = 80$ for $\lambda = 0.8$. The rendering times are higher than in table 1 because we have used a rather large maximum bandwidth in order to prevent any influence of this parameter.

To generate an image of a tail lamp with different colors like in figure 1, the radiance of each set of light bulbs with a common color has to be estimated separately. The pixel values of the resulting images are summed up to obtain

| $\lambda$ | $\lambda_t$ | Online Scaling | Rendering time |
|---|---|---|---|
| 0.8 | 0.8 | no | 335.6 sec |
| 0.8 | 80 | yes | 383.1 sec |
| 16 | 80 | yes | 261.2 sec |
| 40 | 80 | yes | 126.7 sec |
| 79 | 80 | yes | 141.0 sec |
| 80 | 80 | no | 141.5 sec |
| 81 | 80 | yes | 141.6 sec |
| 160 | 80 | yes | 170.8 sec |
| 400 | 80 | yes | 221.1 sec |
| 8000 | 80 | yes | 818.6 sec |
| 8000 | 8000 | no | 158.2 sec |

**Table 2:** *Rendering times of the Touran scene from figure 5a with different bandwidth ratios.*

one combined image. The colors have to be chosen manually during the tone-mapping. To show the ensemble of the lit tail lamp and the rear part of the car, the pixel values of a ray-traced image of the unlit tail lamp can be added to the combined image.

## 7. Conclusion and future work

Many recent automotive tail lamps are built of materials with non-diffuse reflection and transmission properties. The radiance functions of such tail lamps are highly directional. To perform a density estimation of directional radiance functions, it is necessary to take the position as well as the direction of the photons into account. Due to the curse of dimensionality (see [Sco92]), the required number of photons is high (in the order of $10^8$). Density estimation in position-direction space necessitates a tradeoff between bias and noise in the spatial and in the angular domain. This tradeoff can be controlled by the number of nearest neighbors $k$ and the ratio $\lambda$ between the spatial and the angular bandwidth. With Online Scaling, both these parameters can be changed quickly, that is without rebuilding the tree. The distance of the observer to the tail light has to be taken into account when selecting $\lambda$. The heuristic $\Lambda$ adopts $\lambda$ to the distance automatically.

The method could be improved by adding the possibility to handle different colors. One possibility to represent color in the flux map is to assign three flux values to each photon. These values represent the flux in the corresponding RGB channel. In the density estimation, the radiance of each channel has to be estimated separately.

Another possible improvement is the use of a boundary kernel (see [Sco92]) for evaluation points with small angles between $\vec{\omega}$ and the bounding surface of the tail lamp. A boundary kernel reduces the boundary bias at the edge of the hemisphere of directions. We have not addressed this problem here because in the flux map of a typical automotive tail light, there are no photons with small angles due to total internal reflection. But when estimating the radiance of a light source which has significant emission for small angles (like the Lambertian emitter), boundary bias becomes a problem.

The flux map could be used for the density estimation in Moon and Marschner's method for visualizing densely packed scatterers [MM06]. The photons would be simulated as described by Moon, but stored in the flux map and estimated with our method. This might lead to a speed-up compared to Moon's method because of the efficient $k$d-tree search. With online scaling, it would be easier to find a reasonable value for the ratio between the spatial and the angular bandwidths (called $w$ by Moon). It would also be interesting to investigate how the smooth kernel affects the density estimation results.

## References

[AM93] ARYA S., MOUNT D. M.: Algorithms for fast vector quantization. In *Proceedings of DCC 93: Data Compression Conference* (1993), Storer J. A., Cohn M., (Eds.), IEEE Press, pp. 381–390.

[Ben75] BENTLEY J. L.: Multidimensional binary search trees used for associative searching. *Communications of the ACM 18*, 9 (1975), 509–517.

[DGK99] DUNCAN C. A., GOODRICH M. T., KOBOUROV S.: Balanced aspect ratio trees: Combining the advantages of $k-d$ trees and octrees. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'99* (1999), ACM SIGACT, SIAM, Association for Computing Machinery, pp. 300–309.

[FBF77] FRIEDMAN J. H., BENTLEY J. L., FINKEL R. A.: An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software 3*, 3 (1977), 209–226.

[GGHS03] GOESELE M., GRANIER X., HEIDRICH W., SEIDEL H.-P.: Accurate light source acquisition and rendering. *ACM Transactions on Graphics 22*, 3 (July 2003), 621–630.

[Jen96] JENSEN H. W.: Global illumination using photon maps. In *Eurographics Rendering Workshop 1996* (New York City, 1996), Pueyo X., Schröder P., (Eds.), Springer Wien, pp. 21–30.

[Jen01] JENSEN H. W.: *Realistic Image Synthesis Using Photon Mapping*. AK Peters, Natick, 2001.

[LW93] LAFORTUNE E. P., WILLEMS Y. D.: Bi-directional Path Tracing. In *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques* (1993), Santo H. P., (Ed.), pp. 145–153.

[MM06] MOON J. T., MARSCHNER S. R.: Simulating multiple scattering in hair using a photon mapping approach. In *Proceedings of SIGGRAPH 2006* (2006), vol. 25, pp. 1067–1074.

[Sco92] SCOTT D. W.: *Multivariate Density Estimation: Theory, Practice and Visualization*. Wiley series in probability and mathematical statistics. John Wiley, New York, 1992.

[Sil86] SILVERMAN B. W.: *Density Estimation for Statistics and Data Analysis*. Monographs on Statistics and Applied Probability. Chapman and Hall, New York, 1986.

[Spr91] SPROULL R. F.: Refinements to nearest-neighbor searching in $k$-dimensional trees. *Algorithmica 6* (1991), 579–589.