

Christian-Albrechts-Universität zu Kiel
Institut für Informatik
Multimedia Information Processing
Prof. Dr.-Ing. Reinhard Koch

Bachelor Thesis

Model-based Detection of Highlights in Dense Light Field Samples

Jan Christian Kässens

March 31, 2010



Supervised by Dipl.-Ing. Daniel Jung

Abstract

In computer graphics, a 3D scene can be reconstructed from 2D images, which contain different views of the same scene and their Z-buffers. Although the storage size complexity is independent from the scene complexity, the reconstructed scene can grow quite large. As the resolution of the viewing angles is directly related to the sampling density, the latter fact can become a real problem. But points in a 3D scene only change if they have *viewing angle dependent* material properties, that is, they contain specular highlights or reflections.

We will introduce data structures which can hold these scenes and allow efficient access and storage but this work focuses on two methods which estimate viewing angle dependencies through specular highlight detection.

Both methods yield parameters that try to fully reconstruct the scene and thus dispose of the original image information, yielding the ability to reduce the storage complexity by orders of magnitude and make the viewing angle resolution arbitrary at the same time.

Then, the actual results will be discussed and proposals are made which can reduce the errors.

Declaration

I hereby declare that:

- this Bachelor Thesis has been completed by myself independently and only the defined sources and study aids were used. Sections that reflect the thoughts or works of others are made known through the definition of sources
- the following parts of the work now submitted have not been submitted or published for a qualification at a university or similar institution before

Kiel, March 31, 2010 _____

Contents

List of Figures	v
List of Tables	vii
1 Introduction	1
2 Basics	3
2.1 The plenoptic function	3
2.2 Data structures	4
2.3 Scene reconstruction	4
3 Detection methods	9
3.1 Requirements and goals	9
3.2 Intuitive approach	9
3.2.1 Background color detection	10
3.2.2 Intensity peak detection	13
3.2.3 Highlight parameter estimation	14
3.3 Phong-based	14
3.3.1 Intensity peak detection	16
3.3.2 Calculation of color and center	18
3.3.3 Phong exponent adjustment	20
4 Results	23
4.1 Efficiency	23
4.1.1 Disjoint 3D point set configuration	24
4.1.2 Same 3D point set configuration	24
4.1.3 Summary	25
4.2 Error	26
4.2.1 Absolute measurement	27
4.2.2 PerceptualDiff	27
5 Discussion and future work	29
Appendices	31

Contents

A Pseudo code	31
A.1 FIND PEAKS (Phong method)	31
A.1.1 MARK PEAK	31
A.1.2 FIND PEAKS	32
Bibliography	33

List of Figures

2.1	Example of an octree	5
2.2	Example of a quadtree	5
2.3	Image with highlight	6
2.4	Depth map with aperture	6
2.5	A quadtree resulting from 64 input images	6
3.1	A simple highlight	10
3.2	Brightness graph of a QuadTree containing a highlight	11
3.3	Enlarged section of an input image	12
3.4	BRDF graph explaining Phong parameters	16
3.5	Graph building	17
3.6	Creating a plateau with Phong's specular term	19
3.7	Jumpy highlights	19
3.8	Noncontinuous highlight	20
4.1	Input image	27
4.2	Interpolated version	27
4.3	PerceptualDiff result	28

List of Tables

4.1	Big-O storage efficiency	26
-----	------------------------------------	----

1 Introduction

A 3D scene in computer graphics generally consists of (at least) geometry information and material information. Scene geometry is given by coordinate systems and voxel compounds, which include boxes, spheres, cones, planes, teapots, etc. On the other side, the surfaces do have *materials*, that have colors, textures, and light-related properties, that include light emittance, translucency, refraction and reflection parameters.

All these factors are used to compute a view from a given camera position, this process is called *rendering*. In the rendering process, all these parameters, which describe the scene, are lost as the rendering result is, in most cases, a 2D digital image which shows how the scene looks like from a viewing position.

Previous work has been done, that can partly reconstruct the scene if we have many different views from the same scene and their Z-buffers. The Z-buffers, also known as *depth maps*, indicate the actual distance from the camera to the voxel which yielded the color in the corresponding image pixel. Reconstructing the scene this way is called *3D warping* [MMB97].

However, 3D warping does not reconstruct the scene parameters, it merely gives us information about how do voxels look like from the viewing direction, the image was taken from. That is, a snapshot of the light that arrives at the camera at a fixed point in time and space. These snapshots are called *light field samples* and will be

explained in the following chapter. If we have many light field samples, we can see, how a certain point in space changes its color as we go through the images.

Unfortunately, saving the reconstructed scenes requires huge amounts of storage space. For every point in space that can be seen, we must save one color for every light field sample to not lose any (more) information. Furthermore, the more light field samples we save, the better the resulting angular resolution will be, and vice versa.

If we compare, for example, the behavior of a sheet of paper and a flower vase, we will see that the voxels of the flower vase will change their colors rather quick while the sheet of paper's voxels never change their color. This behavior is strongly connected to the concept of *gloss*. People argue about what gloss actually is but we will reduce it to *specular highlights* for the sake of simplicity (but without loss of generality) [Sèv93].

V. Grossmann states that the complexity of storage and access can be significantly reduced if the highlights obey to a certain set of rules (a *model*) which allow them to be parametrized [Gro10]. In synthetic scenes, as described before, highlights are induced by the material properties and thus can easily be parametrized.

This work focuses on the detection of highlights and how the results can be used to approximate the original scene from a set of light field samples. Further, we will introduce how the reconstructed scene can be stored and accessed in an efficient way using well-known hierarchical data structures. One side effect of our methods is that the resolution of the viewing angle becomes arbitrary, as we will dispose of all image information in the detection process and are not dependent on the density of the light field samples anymore.

2 Basics

2.1 The plenoptic function

Radiance is a radiometric measure for the amount of light traveling along a light ray. The light that reaches the viewer can be described by the *plenoptic function* [AB91], which takes seven parameters into account:

Let

- V_x, V_y, V_z be the position of the viewer in the scene
- θ, ϕ be the polar angle and azimuth of the incident ray
- λ be the range of wavelengths of the regarded light
- t be the moment of time

$$P = \rho(V_x, V_y, V_z, \theta, \phi, \lambda, t)$$

So, the plenoptic function captures the *light field* of a scene, that is, all light rays of all wavelengths at every point in time [Ger39]. Our scene, however, is static so we can fix the time parameter t , reducing the function to six dimensions.

2.2 Data structures

Expecting that our scene will need six dimensions as well, we need a data structure that provides efficient storage and access. Much research has been done and very efficient ray tracing algorithms have been developed. As the process of reconstructing the scene is similar to ray tracing, we decide to use octrees and quadtrees for data storage. These structures are tree-based, hierarchical partitioning structures that allow access in logarithmic time and simple mechanisms for creation and rendering [Sam89].

The octree partitions the 3D space into eight equally sized octants which are octrees themselves. The partitioning stops when octree size has fallen below the voxel size in the 3D scene. These octrees do not have children anymore, we call them *octree elements* (see fig. 2.1). Octree elements contain the actual 3D points and their colors. As one 3D point may have many colors, the point may look different from different viewing angles, another data structure is needed, which can assign colors to viewing angles. If we think of unit spheres around the octree element, we can express the incident rays as spherical coordinates and map them to a 2D plane, the quadtree.

Quadtrees are in fact the same as octrees, but in 2D space. They partition into four quadrants and their smallest children are called, similar to the octree, *quadtree elements*, see fig. 2.2.

2.3 Scene reconstruction

The input data set consists of triples containing the input image (see fig. 2.3), which contains the colors to be put into the quadtrees, the depth map or Z-buffer (see fig. 2.4), which contains depth information for the pixels in the image so we know were

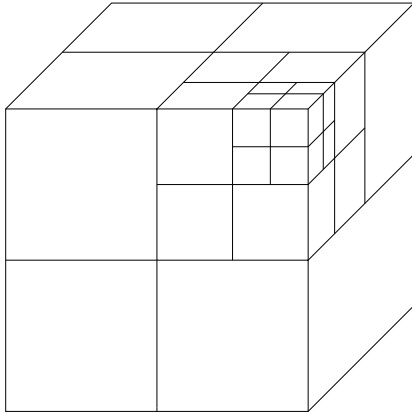


Figure 2.1: Example of an octree

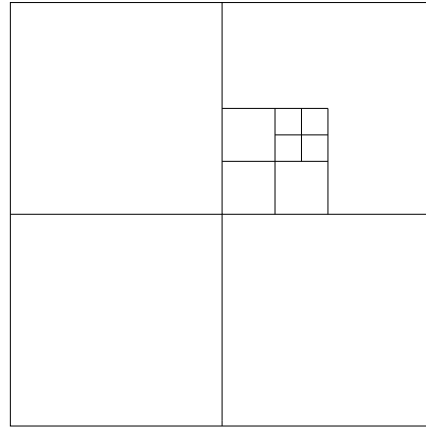


Figure 2.2: Example of a quadtree

to put them in 3D space, the extrinsic camera properties, position, rotation, and the intrinsic camera properties, distortion, field of view (just called “FOV” from now on).

The input images have been rendered from a 3D scene. Now we *unproject* the input images with the known camera properties and obtain a ray into the scene. The depth maps tell us at which depth the original point has been projected from and thus we can determine the absolute 3D position of the point.

At these points, quadtrees are created. Our octrees are aligned to the origin of the scene coordinate system, so their normals’ direction is the *positive Z-axis*. We calculate the azimuth (θ) and polar angles (φ) of the ray relative to the octree normal. The quadtree is addressed via 2D cartesian coordinates so the angles must be mapped to the quadtree coordinate system in order to put the color into the quadtree.

Figure 2.5 shows how a quadtree can look like after 64 images (eight per row) have been shot into the scene. Each point in the figure tells us how that one 3D point, the quadtree belongs to, looks like from the position which is encoded in the quadtree coordinates $x(\theta, \varphi)$ and $y(\theta, \varphi)$, x and y being the functions that map θ

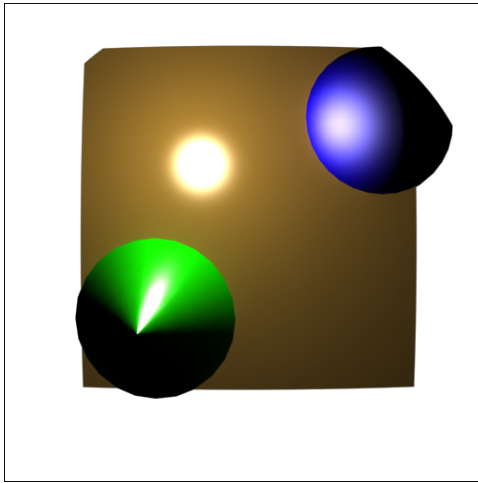


Figure 2.3: Image with highlight

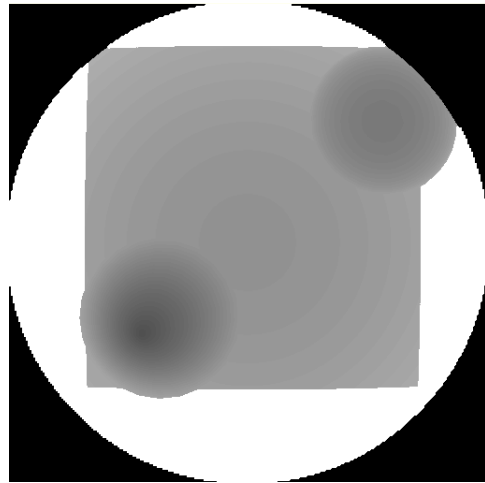


Figure 2.4: Depth map with aperture

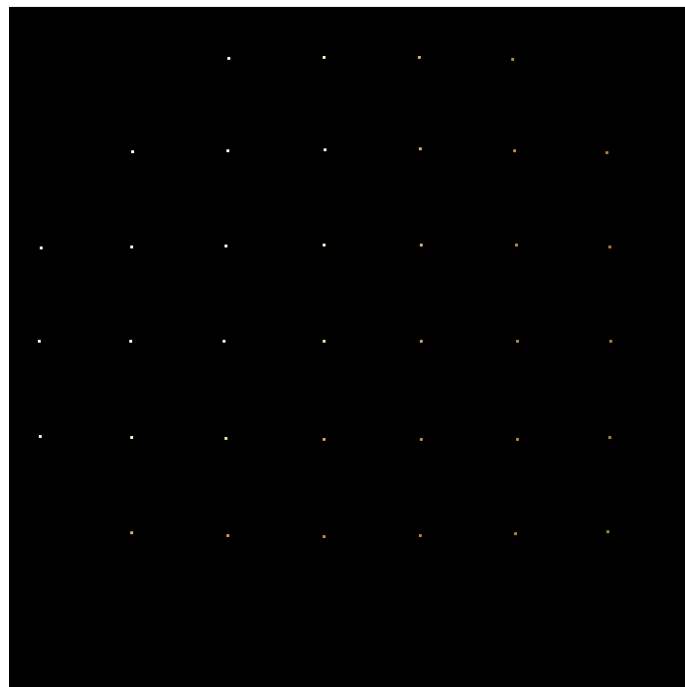


Figure 2.5: A quadtree resulting from 64 input images

and φ to cartesian 2D coordinates, that originate in the lower left corner. So, the azimuth angle determines the distance to the quadtree border from the midpoint and the polar angle the rotation, thus the midpoint of the quadtree is located at $\theta = \varphi = 0$.

3 Detection methods

3.1 Requirements and goals

The octree we created in section 2.3 contains quadtrees with an occupation depending on the number of light field samples used. It is not our ambition to fill it up to get maximum angular resolution, but to store *meta data* in the quadtrees that allow the reconstruction of the quadtree. An octree with fully filled quadtrees would occupy too much storage space to be handled efficiently. Our goal is to identify the necessary parameters and to determine them to make the existing quadtree elements obsolete. This could solve two problems: on the one hand, removing the quadtree elements would save a lot of space and on the other hand the angular resolution was virtually unbounded, as we were not dependent on the discrete quadtree size anymore. The latter is more important when the size of the quadtree is very small for the camera FOV used.

3.2 Intuitive approach

The intuitive approach makes some restrictive assumptions about how highlights look like. In this model a highlight is parametrized by four values:

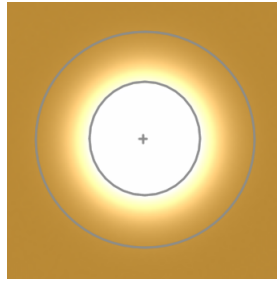


Figure 3.1: A simple highlight

Highlight center is the absolute coordinate of the center in the QuadTree

Highlight color is the color value in the center of the highlight

Plateau radius is the distance between the center and the nearest color value which is not saturated anymore

Highlight radius is the distance between the center and the nearest color value which is equal to the background color

3.2.1 Background color detection

Each quadtree image consists of some *base color* on which one or more highlights could have been drawn. To make detection of these highlights and its parameters easier, we try to separate the base color, which we call the *background color* from the highlights (see fig. 3.2).

We use synthetic scenes so we could have taken the minimum color of the quadtree image as the background color. However, when we created the octree, the input images were anti-aliased which introduced artifacts: These are colors that neither belong to the background of the scene nor to the scene itself (see fig. 3.3). As the background color of our scene was black, we had a few very dark colors or even

Brightness (Intensity)

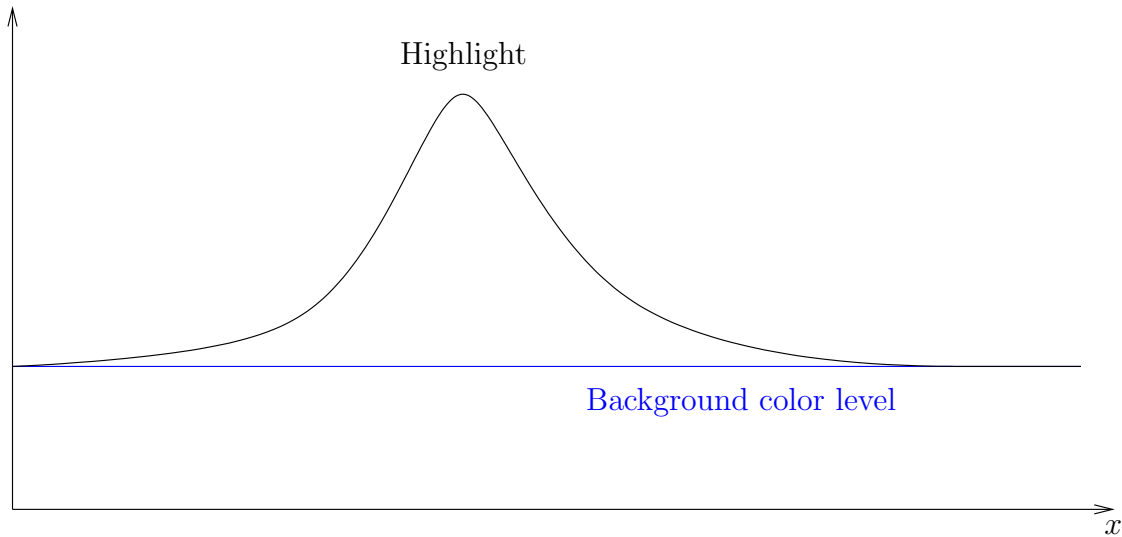


Figure 3.2: Brightness graph of a QuadTree containing a highlight

black which in turn led to a very dark background color of the quadtree image if we took the minimum color.

To remove these artifacts we use standard deviation-based outlier detection and take the average color as background color:

Let

- R_i, G_i, B_i be the red, green and blue color channel values of node i
- $R_{avg}, G_{avg}, B_{avg}$ be the average values,
- $R_{dev}, G_{dev}, B_{dev}$ be the standard deviations
- R_{bg}, G_{bg}, B_{bg} be the resulting background color
- n be the overall number of nodes in the quadtree



Figure 3.3: Enlarged section of an input image

1. Calculate the mean color of all nodes color channel-wise:

$$\begin{aligned} R_{avg} &= \frac{\sum_{i=1}^n R_i}{n-1} \\ G_{avg} &= \frac{\sum_{i=1}^n G_i}{n-1} \\ B_{avg} &= \frac{\sum_{i=1}^n B_i}{n-1} \end{aligned} \tag{3.1}$$

2. Calculate the standard deviation, do this for all nodes in the QuadTree:

$$\begin{aligned} R_{dev} &= \sqrt{\frac{\sum_{i=1}^n |R_{avg} - R_i|^2}{n-1}} \\ G_{dev} &= \sqrt{\frac{\sum_{i=1}^n |G_{avg} - G_i|^2}{n-1}} \\ B_{dev} &= \sqrt{\frac{\sum_{i=1}^n |B_{avg} - B_i|^2}{n-1}} \end{aligned} \tag{3.2}$$

3. Remove all outliers and re-calculate the average color:

$$f(col, avg, dev) = \begin{cases} 0 & \text{if } |avg - col| \leq dev, \\ col & \text{otherwise} \end{cases} \quad (3.3)$$

$$\begin{aligned} R_{bg} &= \frac{\sum_{i=1}^n f(R_i, R_{avg}, R_{dev})}{n-1} \\ G_{bg} &= \frac{\sum_{i=1}^n f(G_i, G_{avg}, G_{dev})}{n-1} \\ B_{bg} &= \frac{\sum_{i=1}^n f(B_i, B_{avg}, B_{dev})}{n-1} \end{aligned} \quad (3.4)$$

Then, this background color has been subtracted from all colors so that we could work with the highlights solely.

3.2.2 Intensity peak detection

To get the position and dimensions of the highlight we try to find the intensity peak. This implies that we must *compare* the colors by their *brightness*. Brightness, however, has at least two well-established meanings:

In the HSV/sRGB color space the brightness is computed as [GW08]

$$\gamma_{sRGB} = \frac{R + G + B}{3} \quad (3.5)$$

In human vision brightness is the perceived luminance of a color [Ade93] and is defined as [ITU90]:

$$\gamma_{709} = 0.2126 \cdot R + 0.7152 \cdot G + 0.0722 \cdot B \quad (3.6)$$

Although the latter formula respects the sensitivity of the human perception system, it significantly reduces the resolution in the blue channel and amplifies errors which could arrive from the green channel, we stick to the classical sRGB brightness calculation.

After the peak detection, all nodes whose colors match the color with the maximum brightness are selected to be the *peak group*.

3.2.3 Highlight parameter estimation

Given that there is only one highlight visible, we can assume that all peak nodes belong together, representing the *highlight plateau*.

The calculation of the plateau radius is done by setting the maximum horizontal and vertical diameter to the maximum difference of the independent coordinates $x(\theta_i, \varphi_i)$ and $y(\theta_i, \varphi_i)$ of each node i :

$$\begin{aligned} \Delta x &= \max(|x(\theta_i, \varphi_i) - x(\theta_j, \varphi_j)|) && \text{and} \\ \Delta y &= \max(|y(\theta_i, \varphi_i) - y(\theta_j, \varphi_j)|) && \text{for every node } i, j \end{aligned} \quad (3.7)$$

The radius is computed as the arithmetic mean of both radii:

$$r = \frac{\frac{\Delta x}{2} + \frac{\Delta y}{2}}{2} = \frac{\Delta x + \Delta y}{4}. \quad (3.8)$$

3.3 Phong-based

In computer graphics the Phong Reflection Model is a widely used, empirically motivated local illumination model which can be used to create specular highlights [Pho75]. A highlight is parametrized by the following set of parameters:

Let

- I_{out} be the reflected¹ light

¹The light is not necessarily *reflected* in the narrowest sense since the Phong model is not energy conserving and the total reflected light's energy may especially be more than the incident light's energy

- I_{in} be the incident light
- I_a be the amount of ambient light present in the scene
- $k_{ambient}, k_{diffuse}, k_{specular}$ be a *empirical material constant*, the ratio of reflection from the ambient, diffuse or specular part of the incident light
- ϕ be the angle between the surface normal and the incident light ray
- θ_p be the angle between viewing direction and the optimal reflection direction, that is, the direction with the most intensive specular reflection part
- n be the shininess value or *Phong exponent*

$$I_{out} = \underbrace{I_a \cdot k_{ambient}}_{\text{ambient part}} + \underbrace{I_{in} \cdot k_{diffuse} \cdot \cos \phi}_{\text{diffuse part}} + \underbrace{I_{in} \cdot k_{specular} \cdot \cos^n \theta_p}_{\text{specular part}} \quad (3.9)$$

In our Phong-based model we try to estimate the parameters which were used to create the existing highlights, or at least parameters which could re-create them. Figure 3.4 shows a simplified *bidirectional reflection distribution function* (BRDF) which should clarify the meaning of the angles used in Phong's formula. The above mentioned parameters can roughly be put in two categories, those which can be directly derived or approximated from given images and/or their geometry and those which cannot be derived.

Objects in a 3D scene which uses the Phong reflection model do not have *background colors* as such, they merely provide material properties which show up as $I_a \cdot k_{ambient}$ in the above equation, which *generate* the background color. So, knowing the background color eliminates the need for the ambient term $I_a \cdot k_{ambient}$. The *diffuse* part involves surface normals and incident light rays but not viewing directions. Therefore the diffuse term is constant with regards to the pixel position in the quadtree and is covered by the background color. We can use the same method as in the intuitive method (see section 3.2.1).

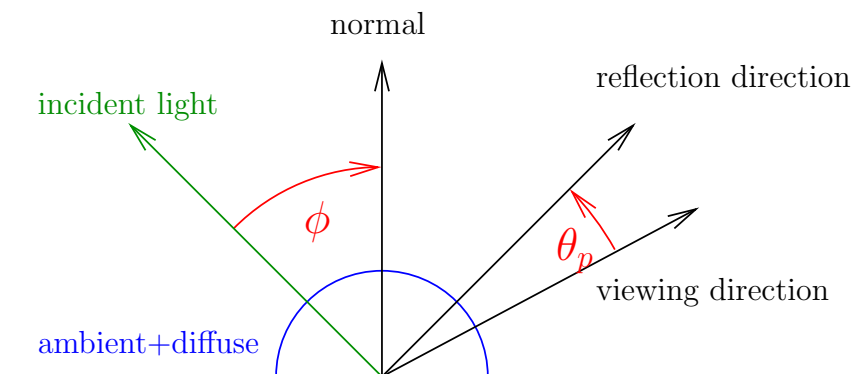


Figure 3.4: BRDF graph explaining Phong parameters

Calculating the angle θ_p can easily be done as the quadtree's coordinates are sphere coordinates. Thus a euclidean distance between two points p_1 and p_2 in the quadtree can be seen as the angle between those rays which hit the quadtree at the given points.

The last thing which must be taken care of is the *Phong exponent*. This value cannot directly be derived from the image data in an easy way. Everything we know is, that it is positive, because zero values would be caught by the background color detection and negative values would generate unrealistic highlights which have a concave intensity graph and would be darker than the background color at their “peaks”.

3.3.1 Intensity peak detection

As we want to be able to find more than one highlight, we need to take a different approach than in the intuitive method. The main idea is to start at an arbitrary quadtree element, determine the difference between the *brightnesses* of each of the neighbor elements and follow them until we reach an element whose gradients are non-positive with at least one negative. The gradient between to elements

is computed as the difference between the *brightness* of both. This element is considered as a peak node, see appendix A.1 for reference.

However, selecting the *neighbor* is not a trivial task. The quadtree elements in fig. 2.5, for example, are not aligned on straight lines, but are a few pixels off because of discretization errors which could have occurred during the space partitioning (see chapter 2.2). In general, the sampling points may or may not be distributed in a grid-like fashion. To cope with neighborhood relationships between the quadtree elements, we build up an undirected, unweighted graph. Each quadtree element represents one vertex and each vertex has at most four edges to the neighbors, *north*, *east*, *south*, *west*. Of course, the problems to know the right neighbors are still to be solved, but using a graph simplifies quadtree traversal greatly. To create the graph, we put a virtual cross onto the quadtree to separate the four regions. Then the vertices with the lowest euclidean distance to the center is selected per region. Figure 3.5 shows this process. This is done to every node and results in a graph which is not necessarily *connected*, i.e. vertices may exist which do not have any edges (neighbors), especially those which arose from quadtree elements in the border regions.

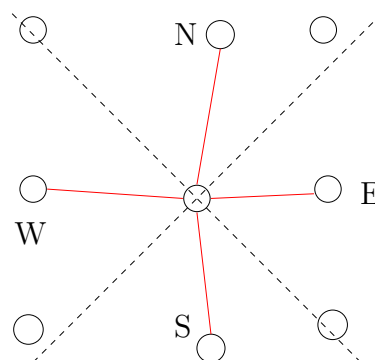


Figure 3.5: Graph building

After we had added every node, we calculated the gradient between each node's brightness and its neighbors' brightness. The brightness used here is, of course, subject to the thoughts on color spaces in section 3.2.2. At this point, we once again traverse the graph to find those vertices whose brightness is higher than the brightness of all of their neighbors. We can *trivially refuse* vertices to be a peak which have at least one neighbor with a positive gradient. This means, that the

neighbor is brighter, so the current node cannot be a peak. We can as well *trivially accept* vertices to be a peak if all gradients to the neighbors are negative (falling). One case is left that cannot be trivially decided on: all gradients are non-positive and not all gradients are negative. In that case, we must examine all adjacent vertices to which the gradient is zero and all *their* neighbors. If at least one of them has a positive gradient to a neighbor, then all connected vertices with the same brightness will *not* be peaks and if these vertices do not have positive gradients to others, we can mark all of them as peaks.

Now that we know which of the vertices are *peak nodes*, we must group them, because a quadtree can have more than one highlight and the peaks may belong to different ones. The groups are made up by the neighborhood criterion of the algorithm proposed in section 3.3.1. As we prepare the estimation of the Phong exponent, we must provide enough information about attenuation of the highlights. Thus, we collect all connected vertices with falling gradients starting from every peak vertex, that each vertex belongs to at most one highlight group. For example, one highlight in the quadtree should result in one huge group of vertices.

3.3.2 Calculation of color and center

Note that the *specular highlights* are modeled using a cosine with an exponent. This term cannot be used to create a plateau like those we use in the intuitive method (see section 3.2). But the RGB color space we are using is discrete and represented by one eight-bit integer number per channel. Consequently, the maximum intensity a color can have, has an upper bound. As highlights could have a plateau on the images, we try to fit a parabola on highlight border points to obtain the *theoretical* intensity the highlight could have at its midpoint, cut off at the intensity's upper bound (see fig. 3.6).

Brightness (Intensity)

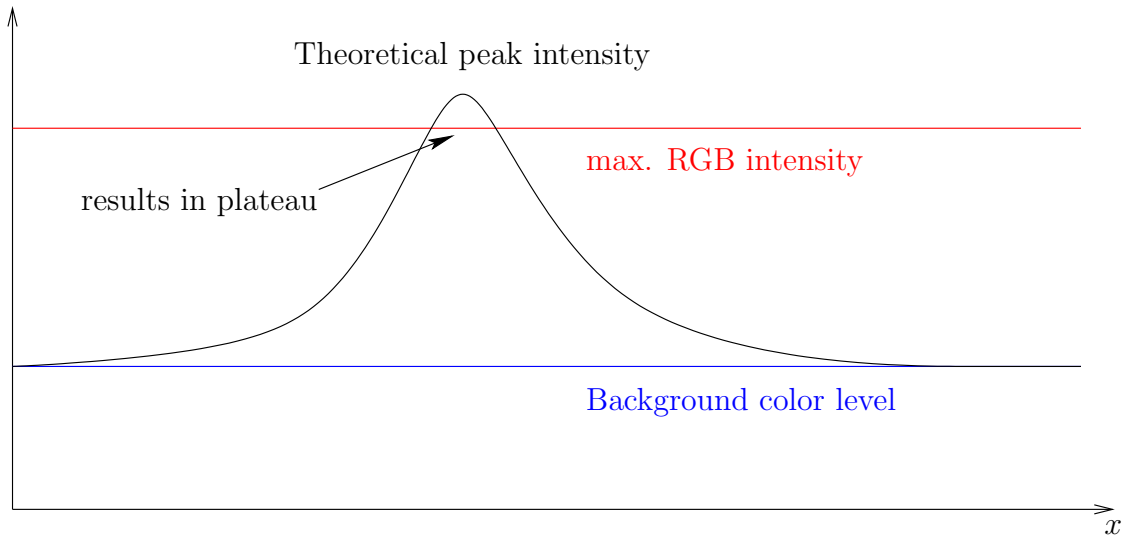


Figure 3.6: Creating a plateau with Phong's specular term

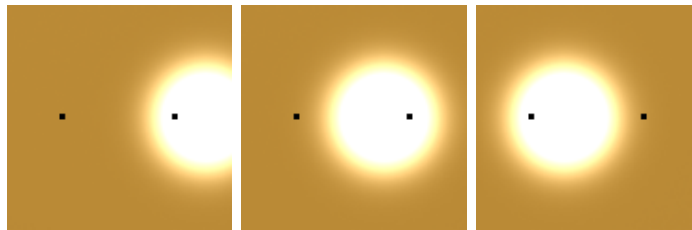


Figure 3.7: Jumpy highlights

This can be done by using the newly-created graph. We have already calculated the gradients between each vertex in the graph. In the gradients, we cannot rely on the brightness of the peak nodes, because they could just be saturated and have the maximum brightness a vertex can virtually have. So we take three vertices (two gradients) which point towards the saturated vertex, use the second order gradient between the vertices to fill in the “missing” gradients over the saturated vertex brightnesses. This way we can calculate the maximum *theoretical* color to make up the highlight plateau with a cosine raised to some power. If the highlight is very sharp (high Phong exponent) and fast-moving, the highlights can appear “jumpy”.

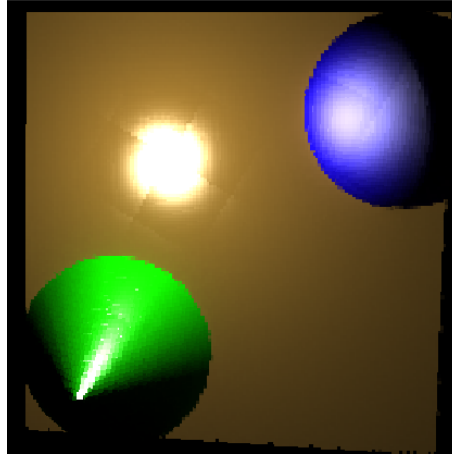


Figure 3.8: Noncontinuous highlight

The black dots in figure 3.7 represent the sampling points of the *real* scene in three different viewing angles. After sampling we only have the color information from the points marked in the figure (dark – bright, dark – bright, bright – dark). In the intuitive center calculation method, this would cause the highlight to be more on the right on the first and second sample and more on the left on the third sample. The positional behavior of the highlight becomes noncontinuous and results in edges, see fig. 3.8 for an example.

Another scenario is that the *real* highlight center lies not completely in the quadtree, see figure 2.5 for an example. Then the center simply cannot be calculated using the mean position of the plateau nodes this way. So without parabolic center estimation, the highlight could never leave the quadtree which leads to noncontinuities in regions of the maximum viewing angle (θ_{max}).

3.3.3 Phong exponent adjustment

Remember that the Phong exponent cannot be guessed directly from the image information. The only clues we have are the supporting points in the quadtree, as well

as the center and the color of the highlight. Phong's specular term ($I_{in} k_{specular} \cos^n \theta_p$) uses I_{in} and $k_{specular}$ to model the absolute highlight color relative to the angle θ . Our parabola fitting has detected the highlight's color so the term that is left is nothing more than $\cos^n \theta_p$, θ_p being known from the quadtree coordinates.

Now we try to find values of n for the above term that fit our supporting points best by applying a least squares optimization. Therefore, we make up sets of supporting points and θ_p s which are called *observation sets*. This enables us to make up a system of equations in an implicit form, with $S = [s_1 \ s_2 \ \dots]^T$ being the column vector of supporting points and $\Theta = [\theta_1 \ \theta_2 \ \dots]^T$ the column vector of corresponding angles to the reflection direction.

$$S = \cos^n(S) \quad \iff \quad 0 = S - \cos^n(\Theta) \quad (3.10)$$

This leads us to the Gauss-Markov-model, a method of steepest descent, which finds the n with the minimum squared error on a given function.

- $l \in \mathbb{R}^m$ be a vector of observations
- $p \in \mathbb{R}^n$ be a vector of parameters to be estimated
- $A \in \mathbb{R}^{m \times n}$ the function f taken as a basis as a (generally not invertible) matrix
- $C_u \in \mathbb{R}^{m \times m}$ be the covariance matrix for l which describes the accuracy of the observations and C_u^{-1} the weighting for the least squares function, respectively

$$A \cdot p \quad = \quad l \quad (3.11)$$

$$\begin{aligned} \iff C_u^{-1} A \cdot p &= C_u^{-1} l \\ \implies A^T C_u^{-1} A \cdot p &= A^T C_u^{-1} l \end{aligned} \quad (3.12)$$

$$\iff p \quad = \quad (A^T C_u^{-1} A)^{-1} A^T C_u^{-1} l \quad (3.13)$$

conclusion 3.12 only holding if and only if matrix A is of full rank.

The Gauss-Markov-model, however, assumes that there is a *linear* relationship between the parameters and the observations (3.11). This is not the case in our Phong equation and thus raises the need for an extension. The Gauss-Helmert-model transforms non-linear implicit functions to linear explicit functions and then uses Gauss-Markov to solve the equation system [MA76]. The Gauss-Helmert-model calculates *increments* Δp on a starting value $p^{(0)}$ with the Jacobian of the parameters and observations:

$$\text{Let} \quad A := \left. \frac{\partial f}{\partial p} \right|_{p_0}, \quad \Delta l := l - f(p^{(0)}) \quad (3.14)$$

Now we can use this in equation 3.11 and get

$$\Delta p^{(0)} = (A^T C_u^{-1} A)^{-1} A^T C_u^{-1} \Delta l \quad (3.15)$$

$$p^{(n+1)} = p^{(n)} + \Delta p^{(n)} \quad (3.16)$$

so we can approximate p in a linearized way until Δp falls below some threshold.

Because the supporting points we chose make up the peak group we have just detected, guessing the center and the color with the parabolic method is critical to the success of the adjustment. Without parabolic detection, the peak group nodes all had the same color, so there is no gradient and thus the observation sets did not have enough distinct information to solve the (then overdetermined) equation system uniquely.

4 Results

4.1 Efficiency

We now compare both, the octree with quadtrees and the octree with interpolation data instead of quadtrees, to each other. Therefore we take the following parameters as a basis:

- n input images and depth maps
- input images of size $l_i \times l_i$ pixels
- octree size with an edge length of l_o and an octree element size of 1
- quadtree size with an edge length of l_q and a quadtree element size of 1
- the FOVs and camera positions are chosen that the samples are equally distributed

We compute the *maximum* size of the octree just after creating it (thus without any detection effort), the whole size of a quadtree being s_q and the size of the octree being s_o , in two (rather extreme) cases, without loss of generality on square numbers as parameters:

4.1.1 Disjoint 3D point set configuration

The input images are natured in a way that one 3D point in the original scene is only visible from *one* camera position. This causes each quadtree to have exactly one quadtree element and a high number of octree elements.

$$s_q = 1 + \sum_{i=1}^{\log_2 l_q} 1 = 1 + \log_2 l_q \quad (4.1)$$

$$s_o = 1 + \underbrace{1 \cdot \sum_{i=1}^{\log_2(l_o)} 1 + s_q}_{\text{first element}} + (nl_i^2 - 1) \cdot \underbrace{\left(\sum_{i=\log_2(l_o)-\log_2(n)}^{\log_2(l_o)} 1 + s_q \right)}_{\text{further elements}} \quad (4.2)$$

$$= 1 + \log_2 l_o + nl_i^2 \log_2 n + nl_i^2 + nl_i^2 s_q - \log_2 n \quad (4.3)$$

In equation 4.1 we describe how many quadtree structures will be created when we put one quadtree element of size 1 into the quadtree, that is, one for the quadtree itself and one for every level we need to descent to get size-one quadtree elements. Essentially, this is similar to equation 4.2 where we do the same for octrees. As we assume that the octree elements are distributed equally, the structures must only be created once up to a certain level. The first element creates the octree itself and anything else to get to the octree element. All further elements can use the (then) already existing structures and just create those which deviate from the first. The result then splits into octree structures and quadtree structures and we achieve a logarithmic storage complexity for the quadtrees regarding the edge length of the quadtree and a quadratic growth $1 + \log_2 l_o + nl_i^2 \log_2 n + nl_i^2 + nl_i^2 s_q - \log_2 n \in \mathcal{O}(l_i^2 \cdot n \log_2 n)$ in the octree complexity.

4.1.2 Same 3D point set configuration

Every input image shows exactly the same 3D points that the other light field samples show. This causes the quadtrees to contain 64 quadtree elements and the

octree to contain fewer octree elements. We assume that the aperture angles are small enough that two incident rays which come from different 3D points in the original scene do not hit the same quadtree element (and overwrite it). This case is actually impossible to achieve, because if all 64 images show the same 3D points. They must be identical and then they would have the same camera positions and resulted in only one quadtree element per quadtree. But if we assume that they did not have the same camera positions, then this case could represent a rather good (but still extreme) case.

Assuming that the camera positions are equally distributed, we get, similar to the previous case:

$$s_q = 1 + \underbrace{1 \cdot \sum_{i=1}^{\log_2(l_q)} 1}_{\text{first element}} + (n-1) \cdot \underbrace{\left(\sum_{i=\log_2(l_q)-\log_2(n)}^{\log_2(l_q)} 1 \right)}_{\text{further elements}} \quad (4.4)$$

$$= 1 + \log_2 l_q + n \log_2 n$$

$$s_o = 1 + \underbrace{1 \cdot \sum_{i=1}^{\log_2(l_o)} 1 + s_q}_{\text{first element}} + (l_i^2 - 1) \cdot \underbrace{\left(\sum_{i=\log_2(l_o)-\log_2(n)}^{\log_2(l_o)} 1 + s_q \right)}_{\text{further elements}} \quad (4.5)$$

$$= 1 + \log_2 l_o + l_i^2 \log_2 n + l_i^2 + l_i^2 s_q - \log_2 n \quad (4.6)$$

We achieve a quadtree complexity of $\mathcal{O}(\log_2 l_q + n \log_2 n)$ and an octree complexity of $\mathcal{O}(l_i^2 \log_2 n)$.

4.1.3 Summary

Real scenes lie somewhere between both extremes. Both detection models allow us to remove the quadtrees as a whole. This results in a very high gain in efficiency. To store the detection data, we introduce a data structure which replaces the quadtrees.

The new data structure contains information about the highlights and background color we have detected and thus, it is of constant complexity on the number of images but is of linear complexity regarding the number of highlights visible on the corresponding quadtree. However, more than one or two highlights on one quadtree are quite rare, most of the time we do not even see *one* highlight on a 3D point. The data structure itself has highlight-independent information (i. e. the background color) and those parameters which describe the highlights itself. For the compression ratio in the table, we assume that one highlight is visible on every 3D point.

Table 4.1 shows how many octrees per scene and how many quadtrees per octree would be created. It shows that we are able to reduce the overall complexity by at least one order of magnitude by cutting the quadtree complexity to $\mathcal{O}(h)$ which, under the assumption of *rare* highlights, can be seen as *constant*.

4.2 Error

Measuring the quality of the highlight detection is not trivial because even if we knew what the parameters of the *real* highlights were, our algorithms detect parameters, that could just re-create the highlights. This does not imply, that the parameters were the same.

So, for reference, we took images from V. Grossmann’s rendering methods [Gro10],

	octrees	quadtrees	detection data sets ¹
Disjoint set	$\mathcal{O}(l_i^2 \cdot n \log_2 n)$	$\mathcal{O}(\log_2 l_q)$	$\mathcal{O}(h)$
Same set	$\mathcal{O}(l_i^2 \log_2 n)$	$\mathcal{O}(\log_2 l_q + n \log_2 n)$	$\mathcal{O}(h)$

Table 4.1: Storage efficiency on octrees

¹ h being the number of highlights

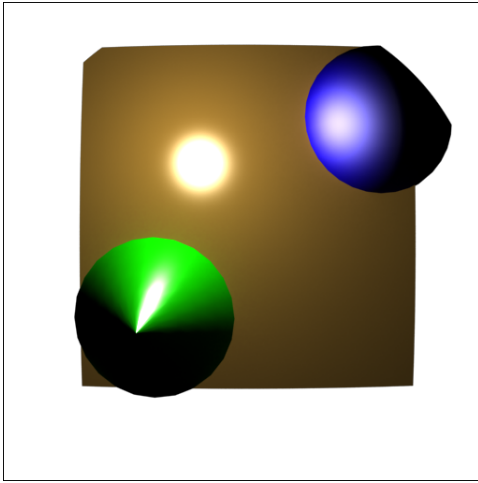


Figure 4.1: Input image

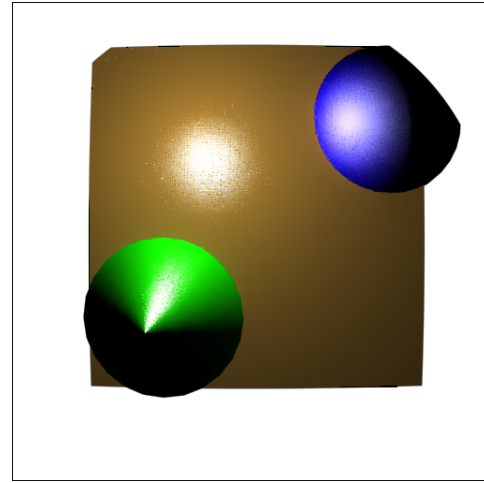


Figure 4.2: Interpolated version

although we must be aware, that his methods could probably introduce new artifacts in the rendering.

4.2.1 Absolute measurement

At first, we did statistical analysis, counted outliers, calculated standard deviation and variance, etc., but as our input images were anti-aliased and the anti-aliased pixels were cut out during the creation of the octree and raise the variances, we found out that this method would not be sufficient, and be even unrepresentative.

4.2.2 PerceptualDiff

Instead, we used an image comparison utility, which is based on a perceptual metric for error measuring [YN04]. Anna Newman and Hector Yee describe a process, which determines the sensitivity of the eye to certain colors using the already introduced L^*u^*v color space together with a *threshold elevation factor* [MR99], which heavily depends on the human eye's ability to perceive spatial frequencies.

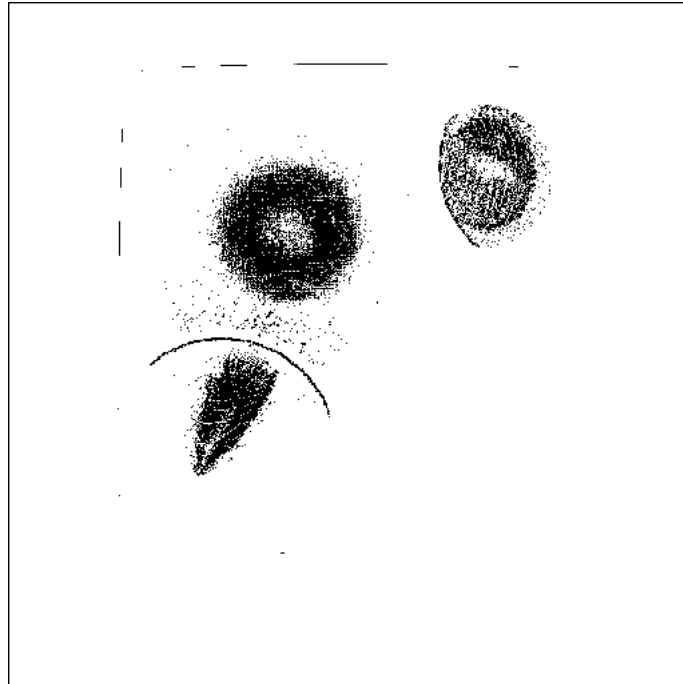


Figure 4.3: PerceptualDiff difference image

Thus, the result of their algorithms could be representative for the error rates in our work, assuming that V. Grossmann's rendering techniques are perfect. fig. 4.3 shows an image generated with Yee's PerceptualDiff utility (with raised contrast for better printing). The black pixels are those pixels which have been *rejected*, that is, which are perceptually different than those in the original image (the input image). One can clearly see the anti-aliasing artifacts, which occur at the upper and left borders, as well as at the borders of the cone. The errors around the highlights at the plane and on the objects result from

- Discretization during octree creation
- Anti-aliasing effects
- Highlight detection inaccuracy
- Rendering inaccuracy

5 Discussion and future work

The last chapter showed that our methods can achieve much but there are still some areas where the results can be improved:

- In section 3.3.2 we introduced parabolic fitting to calculate the virtual center of a highlight to remove edges in the image. As depth maps belong to our light field samples, we could be able to extract surface normals and calculate global positions for light sources. This would result in consistent highlights on all octree elements and *real* rendering of Phong highlights on *real* surfaces and thus, no edges and discontinuities would be visible. However, errors in the light source estimation could yield to worse results as they would propagate to all octrees.
- In the Gauss-Helmert-estimation in section 3.3.3, we used the non-saturated border-points and one single mid-point. To remove noise from the rendered image, we could have calculated virtual colors for every saturated point on the highlight plateau. On the one hand, less “Matrix is not of full rank” errors and thus less failed estimations would occur, but errors introduced by parabolic fitting would be amplified. Further, if multiple highlights are present, the points used for estimation observations could be weighted by their distance to the center using covariance matrices. This would reduce errors which develop during peak-grouping.

-
- Find a better method for background color detection. In the scene we have used thoroughly in this work, the highlight is rather slow-moving. Most of the supporting quadtree elements in the highlight area never contain the real highlight color, so the *detected* color is always much more intense. If the new camera views keep being in the original range of sample positions, this is no problem as the highlight stays there and actually always occludes the background. But if the camera moves beyond the angle our quadtrees held before the detection took place, the background color should be visible to the viewer although we do not know it. This produces bright patches in the output.
 - As we took the Phong reflection model for its simplicity, one could try other, perhaps more realistic and physically motivated models, such as the Torrance-Sparrow model [TS92] for better results.

A Pseudo code

A.1 FIND PEAKS (Phong method)

A.1.1 MARK PEAK

```
PRECONDITION:  vertex      a vertex from the graph
                peakMap    the peak markers (a vector of booleans)
MARK PEAK(vertex, peakMap)
  IF all gradients to all adjacent vertices are  $\leq 0$  THEN
    peakMap[vertex] := TRUE
    IF vertex "west" exists and gradient is 0 THEN
      MARK PEAK(west, peakMap)
    END IF
    IF vertex "east" exists and gradient is 0 THEN
      MARK PEAK(east, peakMap)
    END IF
    IF vertex "north" exists and gradient is 0 THEN
      MARK PEAK(north, peakMap)
    END IF
    IF vertex "south" exists and gradient is 0 THEN
      MARK PEAK(south, peakMap)
    END IF
  END IF
POSTCONDITION: all connected vertices with zero-gradients
                are marked as peaks
```

A.1.2 FIND PEAKS

```
PRECONDITION: graph      the graph created from all quadtree
                    elements
                peaks      an empty list of vertices
FIND PEAKS(graph, peaks)
    initialize peakMap[] to a boolean vector of FALSE

    FOR EACH vertex v IN graph DO
        IF peakMap[v] IS FALSE THEN
            IF all gradients to all adjacent vertices are  $\leq 0$  THEN
                IF vertex "west" exists and gradient is 0 THEN
                    MARK PEAK(west, peakMap)
                END IF
                IF vertex "east" exists and gradient is 0 THEN
                    MARK PEAK(east, peakMap)
                END IF
                IF vertex "north" exists and gradient is 0 THEN
                    MARK PEAK(north, peakMap)
                END IF
                IF vertex "south" exists and gradient is 0 THEN
                    MARK PEAK(south, peakMap)
                END IF
            END IF
        END IF
    END FOR

    FOR EACH vertex v IN graph DO
        IF peakMap[v] IS TRUE THEN
            add v to peaks
        END IF
    END FOR

POSTCONDITION: peaks      contains all peak-vertices
```

Bibliography

- [AB91] E. H. Adelson and J. R. Bergen. The plenoptic function and the elements of early vision. In M. S. Landy and A. J. Movshon, editors, *Computational Models of Visual Processing*, pages 3–20. MIT Press, Cambridge, MA, 1991.
- [Ade93] E. H. Adelson. Perceptual organization and the judgement of brightness. *Science*, 262:2042–2044, 1993.
- [Ger39] A. Gershun. Svetovoe pole (the light field, in english). *Journal of Mathematics and Physics*, XVIII:51–151, 1939.
- [Gro10] V. Grossmann. *Efficient Interpolation of Realistic Highlights for Lightfield Rendering*. Bachelor Thesis, 2010. Christian-Albrechts-Universität zu Kiel, Institut für Informatik.
- [GW08] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*, pages 407–413. Upper Saddle River, N. J.: Prentice Hall, third edition, 2008.
- [ITU90] ITU-R. *Basic Parameter Values for the HDTV Standard for the Studio and for International Programme Exchange*. International Telecommunications Union, Geneva, Switzerland, 1990. ITU-R Recommendation 709.
- [MA76] E. M. Mikhail and F. Ackermann. *Observations and Least Squares*. University Press of America, 1976.
- [MMB97] W. R. Mark, L. McMillan, and G. Bishop. Post-Rendering 3D Warping. In *I3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 7–14. ACM, New York, NY, USA, 1997. ISBN 0-89791-884-3.
- [MR99] D. P. Greenberg M. Ramasubramanian, S. N. Pattanaik. A perceptually based physical error metric for realistic image synthesis. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 73–82. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1999. ISBN 0-201-48560-5.
- [Pho75] B. T. Phong. Illumination for computer generated pictures. *Commun. ACM*, 18(6):311–317, 1975. ISSN 0001-0782.

Bibliography

- [Sam89] Hanan Samet. Implementing ray tracing with octrees and neighbor finding. *Computers And Graphics*, 13:445–460, 1989.
- [Sèv93] Robert Sève. Problems connected with the concept of gloss. *Color Research & Application*, 18(4):241–252, 1993.
- [TS92] K. E. Torrance and E. M. Sparrow. Theory for off-specular reflection from roughened surfaces. pages 32–41, 1992. ISBN 0-86720-294-7.
- [YN04] Y. H. Yee and A. Newman. A perceptual metric for production testing. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Sketches*, page 121. ACM, New York, NY, USA, 2004. ISBN 1-59593-896-2.