Faculty of Mathematics, Computer Sciences and Natural Sciences
Chair of Computer Science VIII (Computer Graphics and Multimedia)
Prof. Dr. Leif Kobbelt

Bachelor's Thesis

Reconstruction of Depth Information from Images for Visualisation of Architectural Models

David Geier
Matriculation Number: 273862

August 2009

First referee: **Prof. Dr. Leif Kobbelt**
Second referee: **Prof. Dr. Bastian Leibe**

I hereby affirm, that I composed this work independently and used no other than the specified sources and tools and that I marked all quotes as such.

Aachen, the August 20, 2009

(David Geier)

Contents

Chapter 1

Introduction

1.1 Motivation

The Chair of Computer Science 8 (computer graphics and multimedia) of the RWTH-Aachen is working on a project, which aims at the creation of a photo-realistic virtual walk-through of the city of Aachen. This project is called the *Virtual Aachen Project*. A major job is to create renderings, which are as detailed and realistic as possible. For a realistic visualisation detailed reconstructions of the facades of the buildings are needed. Facades contain lots of small indentations and elevations (so called *meso-structure*), which have to be modelled to achieve a realistic and satisfying rendering. When introducing more and more details it is important, that the rendering speed does not suffer. One way to achieve an increase in richness of detail is to use more polygons for the geometry of the architectural models. This intuitive approach can be realised easily (e. g. by using a laser scanner to fully reconstruct the model), but the problem is that a higher polygon count directly results in lower rendering speed. Next to the problem of how to render this meso-structure as efficiently as possible, another issue is the acquisition of meso-structure information from real world, which has to be as easy and convenient as possible.

Today there are already some methods for rendering highly detailed mesh surfaces, that do not need extra triangles to model these details (e. g. *parallax mapping* [KTI+01] or the newer *relief mapping* [PNC05]). All these techniques are based on a depth-map of the surface, which is used to perform a per-pixel adjustment of the texture coordinates or normals. So the problem of visualising meso-structure is already solved. The issue we still have is how to obtain depth-maps of a building's surface as conveniently as possible. So far, the depth-maps in the Virtual Aachen Project, which are basically nothing more than greyscale 2D images, were drawn by hand. This approach is very time-consuming and the final result might not survive the scrutiny of those expecting real-world simulations. In late 2008 a new technique has been developed by researchers at the Manchester's School of Computer Sciences and Dolby

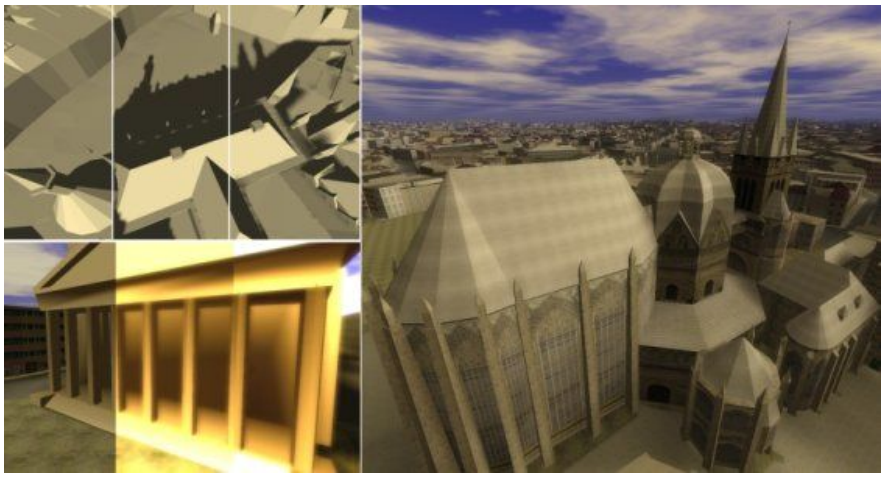


Figure 1.1: Screenshot of the Virtual Aachen Project city viewer

Canada [GWM+08]. They presented a technique called *depth hallucination*, which is supposed to make capturing depth information from 3D surfaces as easy as taking two pictures with a digital camera.

Hence the goal of this thesis is to analyse this new technique in regard to the applicability in the Virtual Aachen Project. Additionally, a tool has to be implemented which can be integrated into the content generation pipeline to easily create depth-maps based on photographs. These depth-maps will be later used for rendering a highly detailed visualisation of the city.

It turned out, that the depth hallucination algorithm has problems in some regions of certain images, which are important to be processable for the Virtual Aachen Project. Thus additionally an *image segmentation* algorithm is implemented to provide a user-friendly way for segmenting the input image into different regions. Then the user can decide in which regions depth estimation should be used, and which regions should get a constant depth, if the algorithm does not work there.

1.2 Shape-from-Shading

Shape recovery is a classical problem in computer vision. It basically means reconstructing information about the 3D structure of a scene from one or more 2D images. The problem is of course the transition from 2D to 3D, based on just two-dimensional information. The extracted 3D shape can be represented in several ways, for example by using depth values $D(x,y)$ (which can be interpreted as relative distances from the camera). There are some more representations, which are not shown here, because they are irrelevant for the remaining thesis. For more information, have a look at [ZTCS99].

Methods for recovering shape from images are called differently, depending on the kind of input images a certain method requires. Examples are shape-from-*motion*, *stereo* or *shading*. The depth hallucination technique can be classified as a *shape-from-shading* method. Shape-from-shading deals with the recovery of shape from a *gradual variation* of shading in the input image(s). Therefore it is essential to study how these images were formed physically, and to use this information to create a model for the shape recovering process. Furthermore, an important aspect is to exploit human limitations in the ability to correctly interpret depth and lighting. These limitations can be used to simplify the algorithms and, at the same time the amount and detail needed for the input images. Shape-from-shading has the advantage, that no special equipment is needed for capturing the images, because just normal photographs are used.

1.3 Previous Methods

In this section previous methods of all algorithms used in this thesis are presented. The main focus is on shape-from-shading as the fundament of depth hallucination. Image segmentation and rendering algorithms are only touched, because they are just used as tools in this thesis.

1.3.1 Shape-from-Shading

In the Virtual Aachen Project various shape reconstruction methods are used. This makes the content generation process a lot more convenient and less time consuming. The landmark paper in this area is from Debevec et al. [DTM96]. They propose a technique for modelling and rendering architectural scenes from a sparse set of still photographs. However surface meso-structure is mostly not taken into account in these kinds of models. Nevertheless this information is essential to achieve a realistic rendering of surfaces.
There are already methods existing to recover meso-structure, but they expect sets of images as input and/or require special equipment for image acquisition. The technique of Yu et al. [YDMH99] needs a geometric model of the scene and a set of high dynamic range photographs, taken with known direct illumination, as input. The technique of Ngan and Durand [ND06] requires a handheld wireless flash as light source. Then a set of images with the flash at different positions have to be captured. Paterson et al. [PCF05] propose a technique, where a set of photographs has to be taken as well, but this time calibration objects have to be placed in the scene for later object registration and to ensure that the light position is known for every captured image. As last example the technique of Li et al. [LFTW05] requires a *gonioreflectometer*[1]. It is obvious, that the acquisition steps of the presented methods are so complicated and

[1] A *gonioreflectometer* is a device for measuring reflectivity of a material. In most cases the **b**idirectional **r**eflectance **d**istribution **f**unction (BRDF) is delivered.

time-consuming, that something easier is needed.

Classical shape-from-shading methods try to extract 3D depth information just from one single input image. Using just one single image makes the problem under-determined, because not enough information is available to uniquely extract depth. Numerous shapes, surface reflectances and lighting conditions can give rise to the same shading pattern and associated ambiguities in shape perception. As already mentioned, exploiting our limitations in correctly interpreting depth and lighting is important. Ostrovsky et al. [OCS05] showed, that humans are quite insensitive to inconsistencies of illumination directions in many real world scenes. The approach by Langer and Bülthoff [LB00], is based on the hypothesis, that shape perception can be explained using a "dark-means-deep" perceptual model. Khan et al. [KRFB05] picked up this approach and successfully showed how to replace one material with another in an image by only using one high dynamic range photograph as input.

The depth hallucination method is to some extent similar to the method of Langer and Zucker [LZ94]. Their model is especially designed for capturing shape-from-shading on a *cloudy day*. They observed that under diffuse lighting conditions, such as the sky on a cloudy day, surface brightness depends primarily on the amount of sky, visible from each surface element. This assumption is the basis of the mathematical model, used in the depth hallucination algorithm. However in comparison to their method the depth hallucination technique is a lot faster, because the used model is simpler and fewer constraints are imposed.

1.3.2 Rendering of Meso-Structure

Improving scene realism by using texture mapping is an fundamental component of modern rendering systems. It adds significant amount of detail to the scenes by simulating the appearance of different materials. In 1976, Blinn and Newell mapped first images over surfaces to create the illusion of detail without adding geometry [BN76]. Over the time a few texture mapping extensions have been developed, which further try to increase the realism of the rendered scenes.

Bump-maps allow flat surfaces to have not only the colour, but the appropriate shading of the perceived detail geometry by using additionally information about the normals [Bli78, Coo84]. *Relief textures*, introduced by Oliviera and Bishop [OBM00], are a technique for pre-distorting textures based on depth information. Although this technique was not practicable enough to get into industry, the idea was picked up later for *parallax mapping* and *relief mapping*. Kaneko et al. [KTI+01] introduced parallax mapping, which for the first time allowed efficient self-occlusion and parallax effects for bump mapped surfaces. Welsh first implemented parallax mapping on a programmable GPU [Wel04]. Today parallax mapping is almost as efficient as plain texture mapping on modern GPUs, but appears a lot more realistic than bump mapping

alone. *Relief texture mapping* on arbitrary polygons, as presented by Policarpo et al. [PNC05], is the latest technique for rendering highly detailed surfaces in real-time, without introducing new geometry. It yields the best visual results, but is the slowest technique at the same time. I chose this technique for visualisation in the tool, because image quality is a lot more important than speed.

1.3.3 Image Segmentation

For separating images into different regions a big number of different algorithms with different approaches exist. They range from quite simple, pixel-based to complex, model-based approaches. Basically the algorithms can be classified into *pixel-*, *region-*, *edge-*, *model-*, *texture-* and *graph-partitioning*-based approaches, which are shortly presented in the following.

Pixel-based Methods Here the classification of segment membership of a pixel is exclusively based on its intensity. The pixel's local neighbourhood is not taken into account. Pixel-based methods are simple to implement and fast, but do not yield *coherent* segments in most cases [N.79].

Region-based Methods In comparison to pixel-based methods, region-based methods take the local neighbourhood of a pixel to classify into account. A set of so called *seed points* is required as hint which pixels belong definitely to a certain segment. Starting from these seed points, the algorithm grows the segments. Thus coherent segments are formed [SS01, WF06].

Edge-based Methods Edges often match the contours of the searched segments. Hence discontinuities between image regions can be used for segmentation. A problem is that most existing techniques do not return closed regions. Thus a post-processing step is nearly always needed, in order to get closed segments [PzG05].

Model-based Methods So far all approaches are only based on local image information. Model-based approaches introduce a model of the searched objects for segmentation (e. g. the geometrical shape of the objects, which are assumed to appear in the input image). So previous knowledge about the content of the images is used and needed for segmentation [KKH+93].

Texture-based Methods Often segments are not characterised by a colour, but by a consistent structure (*texture*). Texture-based methods try to use these structural information for segmentation [WHMM06].

Graph-partitioning-based Methods *Graphs* from graph theory can be effectively used for image segmentation. The pixels of an image are transformed into the graph's nodes. The edge weights describe the (dis)similarity among the neighbourhood pixels. Then graph cutting techniques are used to partition the

nodes of the graph into two disjoint sets, which represents the image's segments [GPS89, FH04].

I decided to use a graph-partitioning-based method for segmentation. The reasons for my choice are given in chapter 3.

Chapter 2

Depth Hallucination

In this chapter the algorithm "A Perceptually Validated Model for Surface Depth Hallucination" [GWM+08], presented at SIGGRAPH 2008 is described in detail. First of all an intuitive explanation and a coarse overview of the algorithm is given. This makes it easier to understand the inner workings. After that, all steps of the depth estimation process are explained in detail.

The depth hallucination method expects three photographs as input. The first photograph is captured without flash under diffuse illumination. Using just this single image for depth estimation would not be enough, because the assumption that portions of the surface that are higher appear brighter, whereas portions that are deeper appear darker, does not hold in general. The problem with this assumption is that different materials of a surface also reflect light differently. This fact makes it impossible to decide, if the brightness difference between two portions (e. g. between two pixels) is a function of colour or a function of depth. By capturing a second photograph, this time with flash, the surface reflectance behaviour, also known as *surface albedo*[1] can be calculated. Using this information, the colour of all visible portions of the surface can be captured. The third photograph is a *white balance* image (flash calibration image). It is taken of a white *Lambertian*[2] surface (e. g. a white sheet, stretched over a frame) at a similar distance and aperture like the other two pictures and is used for vignetting falloff compensation.

In the algorithm, the two images captured first, essentially become a reflectance map (*albedo-map*) and a *depth shading image*. Then, for every matching pair of pixels

[1] The *albedo* (derived from Latin albedo = "whiteness", or albus = "white") is a measure for the extent, to which a surface diffusely reflects light from the sun. It is therefore a more specific form of reflectance.

[2] A *Lambertian* surface has a perfectly diffuse (matte) material. The intensity of light, which is reflected in a given direction from any small surface element, is proportional to the cosine of the angle of the normal to the surface. This is known as *Lambert's Cosine Law*. An important consequence is, that a Lambertian surface always has the same brightness, regardless of the viewing direction [McC05].

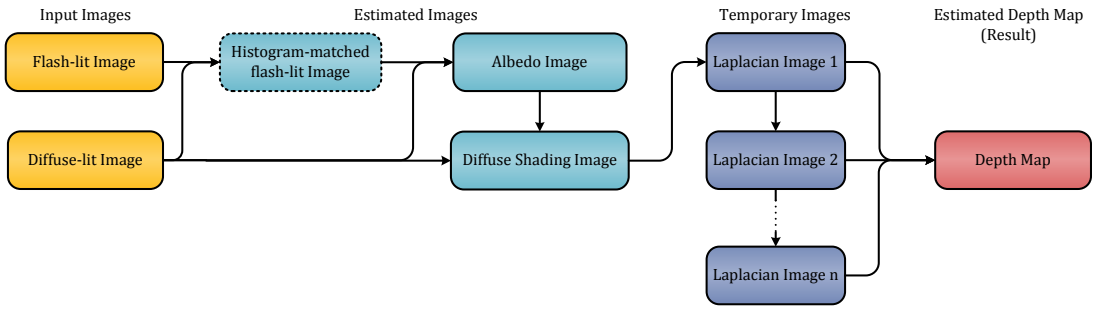


Figure 2.1: Algorithm flow chart and relationship between involved images

from these two images it is calculated how much of the pixel's brightness depends on its depth, and how much is due to its colour.

2.1 Overview

In figure 2.1 an overview of the algorithm together with all used images is shown. The individual steps are: taking photographs, histogram-matching (if required), estimation of albedo-map and diffuse shading image, building a Laplacian pyramid of the diffuse shading image, and finally the depth estimation. All steps are described in detail and illustrated using a sample picture pair in the following sections.

For simplifying the mathematical model used for depth estimation later on the following assumptions are made.

1. The surface to reconstruct is approximately Lambertian and opaque with an average reflectance ρ between 2% and 70%. Further, the surface's materials shouldn't have any specularities. If it contains significant specularities, *cross-polarisation*[3] can be used to minimise highlights [Her].

2. The underlying surface must be plausibly representable as a height field and is without global curvature (like a floor or a wall).

3. A shading change over a small region in general corresponds to a smaller change in depth, than the same shading change over a large region. This is assumed, because in nature very small indentations with a very high depth are rarely found.

4. Under diffuse lighting conditions (such as the sky on a cloudy day), surface brightness depends primarily on the amount of sky visible from each surface element [LZ94].

[3] *Cross-polarisation* can be used to remove reflections from photographs, originated by the flash. This is done by using polarising filters for the camera's meter on the lens and flash heads.

(a) Photograph under diffuse condition (b) Photograph taken with flash

Figure 2.2: Example input photograph pair

2.2 Image Acquisition

Before the hallucination algorithm actually can be executed, the input pictures have to be taken. In the paper a standard digital SLR (**S**ingle-**L**ens **R**eflex) camera is suggested, which I used too, to take all test pictures. The camera is ideally mounted on a tripod (to avoid camera shakes) and perpendicularly orientated towards the surface (to avoid perspective distortions).

First the diffuse-lit image is captured under indirect illumination, e. g. under an overcast sky or in shadow. Next the flash-lit image is captured, with the flash fired at full power and the camera staying at the same position. Ideally the flash is as close as possible to the camera lens to minimise shadows. All images used in this thesis were captured using a standard flash mount. In figure 2.2 a diffuse/flash image pair is shown, which will be used to visualise the algorithm in the next sections. The pictures show a bumpy cobblestone pavement.

2.3 Definition of an Image

In the following sections a lot of operations on two-dimensional greyscale and colour images are performed. Therefore it is essential to give a precise mathematical model of images and their operations first. Images can be interpreted as matrices of pixels. Depending on the number of channels (mostly 1 for greyscale and 3 for colour images), each matrix entry has a different number of values, represented as n-dimensional vectors. The vector components model the different channels of a pixel. The following definitions formalise images:

$G = [0,1] \subset \mathbb{R}$	set of continues greyscale values between 0 and 1
$w,h,c \in \mathbb{N}$	width, height, number of channels of the image
$s = w \cdot h, s \in \mathbb{N}$	total number of pixels (size) of the image
$x \in \{1,\ldots,w\}$	x-coordinate of a pixel
$y \in \{1,\ldots,h\}$	y-coordinate of a pixel
(x,y)	coordinate of a pixel in the image matrix
$I = (i(x,y,c))$	image matrix
$i(x,y) = g = (g_1,\ldots,g_c)^T \in G^c$	intensities of channels of pixel at (x,y)

For example a greyscale image can be defined as $I = (i(x,y,1))$, and a RGB colour image can be defined as $I = (i(x,y,3))$. With exact reference to the mathematical notation of matrices (e. g. $A = (a_{i,j})$) the row and column counter x,y should be written as indices $s_{x,y}$. I deviate from this notation here, because sometimes it is convenient to be able to consider a pixel value at (x,y) as a function value. The mathematical model above is partly adopted from [NFH07].

Next some operators on images are defined, which will be used later on. The *greyscale value* of a pixel $g = i(x,y)$ from a RGB image $I = (i(x,y,3))$ can be calculated as a weighted sum[4] of the red (g_1), green (g_2), and blue (g_3) channel:

$$\text{grey} : G^3 \to G, g \mapsto 0.3g_1 + 0.59g_2 + 0.11g_3 \tag{2.1}$$

In many image processing calculations it can happen, that pixel values overflow or underflow, which means for a pixel's channel: $g_i \notin [0,1]$. To clamp the pixels in the correct range a so called *saturation* function is defined, which makes use of the following *clamping* function.

$$\text{clamp} : \mathbb{R} \to \mathbb{R}, x \mapsto \begin{cases} 0, & x < 0 \\ x, & 0 \leq x \leq 1 \\ 1, & x > 1 \end{cases}$$

$$\text{saturate} : \mathbb{R}^c \to G^c, g \mapsto (\text{clamp}(g_1),\ldots,\text{clamp}(g_c))^T \tag{2.2}$$

2.4 Albedo-Map and Diffuse Shading Image

After the image acquisition first of all the input photographs have to be converted to linear, floating-point pixel values in the range of $[0,1] = G$. Further it is required, that all three images have the same width and height.

[4]The weights are chosen like this, because the eye is most sensitive to green, then to red and lastly to blue. For example, for equal amounts of blue and green light, the green light will nevertheless seem much brighter. By using different weights for the different channels, the produced greyscale image's brightness is perceptually equivalent to the brightness of the original image.

Next, the *albedo-map* $I_a = (i_a(x,y,3))$ is calculated. This is done by a pixel-wise subtraction of the diffuse-lit image $I_d = (i_d(x,y,3))$ from the flash-lit image $I_f = (i_f(x,y,3))$ and a subsequent, pixel-wise division by the white balance calibration image $I_c = (i_c(x,y,3))$. Dividing by the white-balance image is used for *vignetting falloff*[5] correction. This yields approximate reflectance values at each pixel.

$$i_a(x,y) = \frac{i_f(x,y) - i_d(x,y)}{i_c(x,y)} \tag{2.3}$$

Finally, the albedo-map has to be channel-wise normalised to the brightness of the diffuse image. This is required, because in general the albedo-map is quite dark and later on a weighting between the albedo-map and the diffuse image is calculated. This weighting does not work if the colour values differ too much.
For the normalisation first the mean pixel intensity of each channel of the albedo-map and the diffuse image has to be computed. This is done by channel-wise dividing the sum of all pixel intensities by the number of pixels (image size). Given an image $I = i(x,y,c)$ the channel-wise mean colour can be defined as

$$\text{mean} : I \to \mathbb{R}^c, i(x,y) \mapsto \frac{1}{s_i} \left(\sum_{x=1}^{w_i} \sum_{y=1}^{h_i} i(x,y) \right).$$

Next, the normalisation it-self is applied. By first subtracting the mean value $\text{mean}(i_a)$ from the albedo-map the image is made zero-mean. After that adding $\text{mean}(i_d)$ shifts the mean to the mean of the diffuse image.

$$i_a(x,y) = \text{saturate}\left(i_a(x,y) - \text{mean}(i_a) + \text{mean}(i_d)\right) \tag{2.4}$$

It is important to note, that the albedo-map is updated in this step and no new image is created. In figure 2.3 the resulting albedo-map for the cobblestone pavement is shown.

Now, the *diffuse shading image* $I_s = (i_s(x,y,1))$ is calculated. It describes how much of a pixel's brightness is due to its position and how much is due to its colour. This calculation is based on the assumption, that with increasing depth reflectivity decreases. For the diffuse shading image it is enough to only use the luminance channel, because the final depth-map will be greyscale. The diffuse shading image is calculated by pixel-wise dividing the greyscale values of the diffuse-lit image by the greyscale values of the albedo-map.

$$I_s = \frac{\text{grey}(i_d(x,y))}{\text{grey}(i_a(x,y))} \tag{2.5}$$

This can be interpreted as a weighting of the diffuse colour of a pixel by its reflectivity. This way, e. g. a dark pixel with high reflectivity (low albedo) gets a high value and a

[5] In photography and optics, *vignetting* means a gradual darkening of the image towards the corners, compared to the centre [ZLK06].

(a) Estimated albedo-map (b) Estimated diffuse shading image

Figure 2.3: Example albedo-map and diffuse shading image, generated from the image pair shown in figure 2.2

bright pixel with low reflectivity (high albedo) gets a small value. Using the mathematical model presented later on, these depth shading values are transformed into actual depth estimates. The diffuse shading image still has to be normalised to a mean value of 0.5, because the depth estimation method described in the next section assigns a height of 0 to a pixel intensity of 0.5. The normalisation is done in the same way as described above for the albedo-map. The only difference is that the diffuse shading image is normalised to a mean value of 0.5, and just one channel has to be considered, because the image is greyscale. Figure 2.3 shows the estimated and normalised diffuse shading image for the cobblestone pavement sample.

2.5 Depth Estimation

Based on the previously calculated albedo-map and diffuse shading image the depth-map can be estimated. The estimation process is based on a mathematical model, which will be derived in this section. However at first Gaussian Laplacian pyramids are presented, because they play an important role in the depth estimation process later on.

2.5.1 Gaussian Laplacian Pyramids

Sharpness and blurring are characteristics of digital images. They can be used to recognise structures in an image, or to manipulate it. Sharpness and blurring is represented by the different *frequency bands* of an image. With the *Fourier transformation* such a frequency decomposition can be calculated, but this has some drawbacks. First, the computational costs of this method are very high. Second, after the Fourier transformation the different frequency components in frequency space can be identified, but

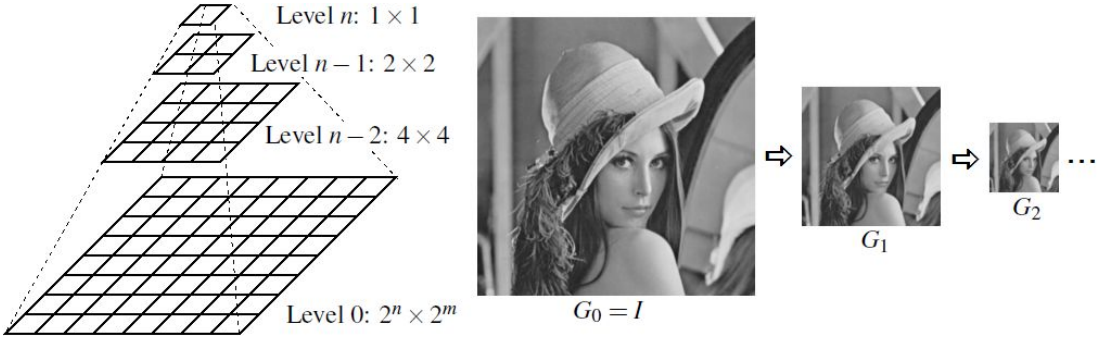


Figure 2.4: Visualisation of a Gaussian pyramid with three example levels

a direct mapping to structures in local area (image space) is missing. A *Gaussian Laplacian pyramid* can be used, to efficiently extract the frequency bands from a digital image. Additionally, it yields a direct mapping to structures in local area. The algorithm was first presented by Burt and Adelson in 1983 [BA83].

2.5.1.1 Gaussian Pyramid

To construct a Gaussian Laplacian pyramid, first a *Gaussian pyramid* has to be constructed. The input image $I = (i(x,y,c))$ has to be quadratic, so that halving the image size does not yield fractions. This means for the width and height: $w = h = 2^r, r \in \mathbb{N}$. The first pyramid level G_0 is set to the input image I. The next pyramid level G_1 is calculated, by low-pass filtering and halving the previous level G_0. This is called the REDUCE operation. For a filter kernel $w(x,y)$ of dimension 5×5 and a reduction factor of four, the following REDUCE operation is often used:

$$\text{REDUCE}(I)(x,y) = \sum_{i=-2}^{2} \sum_{j=-2}^{2} w(2+i,2+j) \cdot I(2x+i,2y+j)$$

These two steps are repeated level by level, until an image size of 1×1 is reached.

$$\begin{aligned} G_0 &= I \\ G_{i+1} &= \text{REDUCE}(G_i), i = 0 \ldots r-1 \end{aligned}$$

This process results in a series of images, where each image represents a certain frequency portion of the input image and each successor image is $\frac{1}{4}$ smaller than its predecessor image [Jäh05, NFH07]. Figure 2.4 shows a Gaussian pyramid[6] with levels G_0, G_1 and G_2.

[6]The pyramid image was taken from `http://fourier.eng.hmc.edu/e161/lectures/figures/Image_Pyramid.gif`.

2.5.1.2 Laplacian Pyramid

Next, the *Laplacian pyramid* is constructed. It is build by subtracting adjacent Gaussian pyramid levels. This is also known as the *DoG* algorithm (**D**ifference **o**f **G**aussian). A problem is that two adjacent levels of a Gaussian pyramid do not have the same size. So before subtraction the smaller image (the higher pyramid level) has to be upsampled. The missing rows and columns are interpolated. This is called the EXPAND operation, which is the inverse of the REDUCE operation. The EXPAND operation can be applied multiple times, so that every pyramid level can be expanded to the size of the original image. This can be formalised in the following way:

$$\begin{aligned} G_{i,1} &= \text{EXPAND}(G_{i,0}), \text{ with } G_{i,0} = G_i, i = r, r-1, \ldots, 1 \\ G_{i,2} &= \text{EXPAND}(G_{i,1}) \\ &\ldots \\ G_{i,i} &= \text{EXPAND}(G_{i,i-1}) \end{aligned}$$

The images $G_{1,1}, G_{2,2}, \ldots, G_{r,r}$ are expanded to the size of the input image I. In the levels of a Laplacian pyramid, those image information are saved, which got filtered out by the REDUCE operation in a Gaussian pyramid. For a filter kernel $w(x,y)$ of dimension 5×5 and a reduction factor of four, the following EXPAND operation is often used:

$$EXPAND(I)(x,y) = 4 \sum_{i=-2}^{2} \sum_{j=-2}^{2} w(2+i, 2+j) \cdot I\left(\frac{x+i}{2}, \frac{y-j}{2}\right)$$

The Laplacian pyramid levels L_i are then calculated like that:

$$\begin{aligned} L_i &= G_i - \text{EXPAND}(G_{i+1}) = G_{i,0} - G_{i+1,i}, i = 0, 1, \ldots, r-1 \\ L_r &= G_r \end{aligned}$$

The top of the Laplacian pyramid L_r was set to the top of the Gaussian pyramid G_r, because one image is missing for subtraction in the last pyramid level. In figure 2.5, the construction of a Laplacian pyramid, based on a Gaussian pyramid, is shown [Jäh05, NFH07].

2.5.1.3 Simple Implementation

For a practical implementation two requirements of the depth hallucination algorithm have to be considered. First, the input image G_0 can have an arbitrary size, which may not be a power of two, or the image's width and height are different. Second, a Laplacian pyramid is required, which consists of equally sized levels.
To efficiently create such a Laplacian pyramid with n levels, I use the following algorithm. First, create $n+1$ Gaussian blurred copies of the input image, by successively increasing the blurring radius r by powers of three. After that, subtract adjacent levels to get a Laplacian pyramid with n equally sized levels. In listing 2.1, the algorithm is shown.

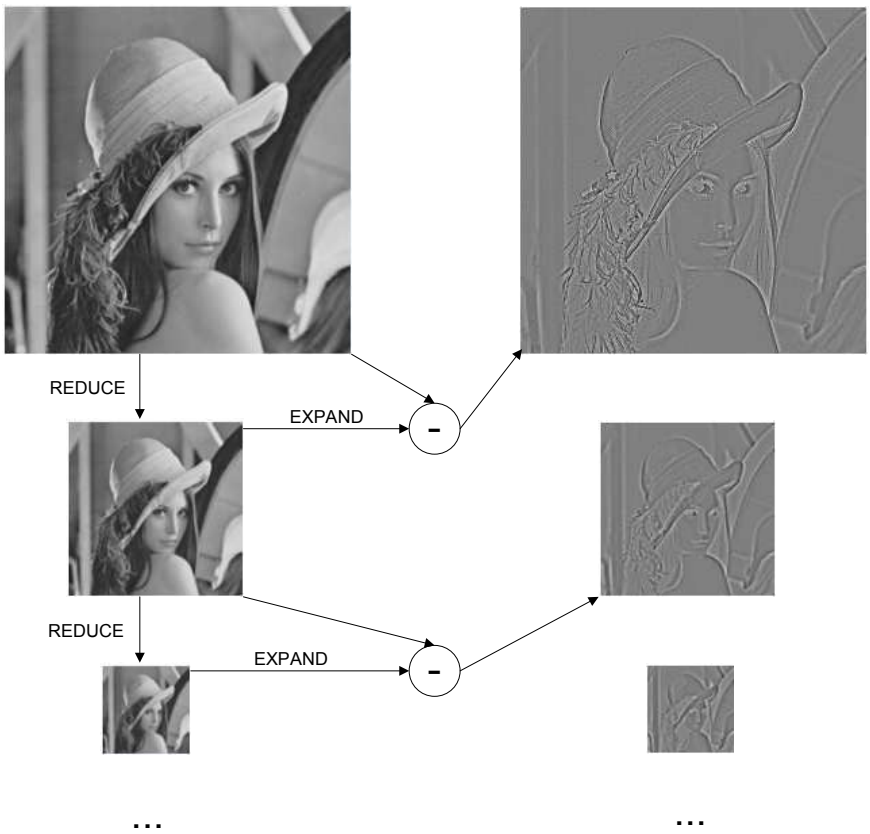


Figure 2.5: Construction of a Laplacian pyramid from a Gaussian pyramid

```
1  image[n] laplacianPyramid(image img, int n)
2  {
3      image gaussPyr[n+1];
4      image laplacePyr[n];
5
6      gaussPyr[0] = img;
7
8      for (int i=3,r=1; i<n; i++) do
9      {
10         gaussPyr[i] = blurRxR(img, r);
11         laplacePyr[i-1] = gaussPyr[i-1]-gaussPyr[i];
12         r = r*3;
13     }
14
15     return laplacePyr;
16 }
```

Listing 2.1: Algorithm for constructing an equally sized Laplacian pyramid of an arbitrarily sized image

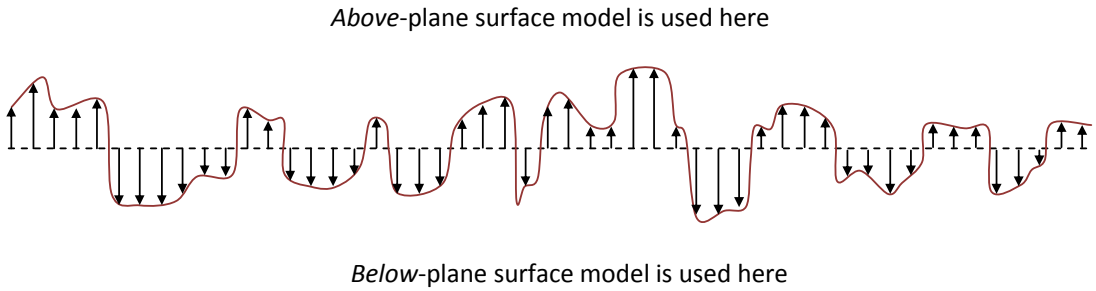


Figure 2.6: Profile of a surface height field and the separation between the above-plane and below-plane model

2.5.2 Mathematical Model

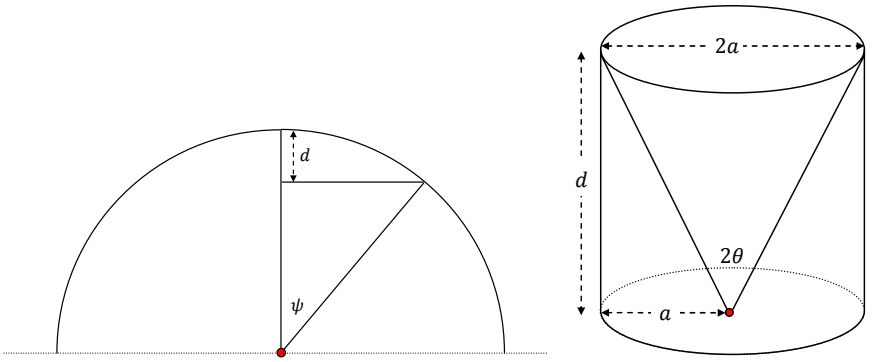
In the following sections a mathematical model is derived, which describes depth depending on the depth shading factors from the diffuse shading image. The depth estimation works pixel-wise and entirely in image space.

Surface meso-structure can be modelled as a terrain with hills and valleys, with both having different amounts of incident light. The orientation to the sky dominates on the hills, while the visible aperture effect dominates in the valleys. In the valleys at least the hillsides are partly in shadow.
Due to the distinction between hills and valleys first two different models are derived to approximate the different types of relationships between depth and shading. The scope of each model is shown on a hypothetical surface in figure 2.6. The transition between the two models is a zero-plane, where depth is assumed to be 0. Above this plane the *above plane model* and below this plane the *below plane model* is used. Later these two models are combined into one. The derivation of both models is based on the assumption, that under diffuse lighting (like on a cloudy day) depth at a surface element mainly depends on the amount of incident light [LZ94].

2.5.2.1 Below-Plane Model

The below plane model is based on approximating pits in the surface as *cylinders* with aperture $2a$ and depth d. See figure 2.7(b) for an illustration. Interreflections are ignored, because they would complicate the model a lot. Further, interreflections just seem to affect the scale, but not the character of the depth approximation.
First, the amount of light $E_c(\theta)$, which is incident to the cylinder, depending on the angle of aperture θ, has to be determined. $E_c(\theta)$ is calculated by integrating the cosine weighting over the solid angle subtended by the visible sky:

$$E_c(\theta) = 2\pi \int_0^{\theta} \cos\theta' \sin\theta' d\theta' = 2\pi \frac{\sin^2\theta}{2} = \pi \sin^2\theta$$



(a) Above-plane model using hemispheres

(b) Below-plane model using cylinders

Figure 2.7: Model for approximating pits and surface protrusions

The integral is solved using *integration by parts*:

$$\begin{aligned} &\int \sin x \cos x = \sin^2 x - \int \cos x \sin x \\ \Leftrightarrow &2\int \sin x \cos x = \sin^2 x \\ \Leftrightarrow &\int \sin x \cos x = \frac{\sin^2 x}{2} \end{aligned}$$

To arrive at the final shading factor S, $E_c(\theta)$ is divided by the illumination factor for the full sky E_h, which is $E_h = E_c(90^\circ) = \pi$. Using $\sin\varphi = \frac{\text{opposite leg of } \varphi}{\text{hypotenuse of } \varphi}$ and Pythagoras' theorem, the shading factor finally becomes:

$$S = \frac{E_c(\theta)}{E_h} = \frac{\pi \sin^2\theta}{\pi} = \sin^2\theta = \left(\frac{a}{h}\right)^2 = \frac{a^2}{a^2 + d^2} \tag{2.6}$$

Pit depth can therefore be estimated by solving equation (2.6) for depth d as (see figure 2.8 for the function graph):

$$d = a\sqrt{\frac{1}{S} - 1} \tag{2.7}$$

2.5.2.2 Above-Plane Model

The above-plane model uses *hemispheres* to model surface protrusions. Shading of these is a function of the visible portion of the hemisphere h_v, subtended by the angle ψ and added to the remaining reflected portion of the hemisphere h_r outside this angle. See figure 2.7(a) for an illustration. This sum is finally divided by the illumination factor for the full sky, which was already shown in the derivation for the below-plane

model to be π.

In the following ρ is the surrounding surface reflectance, which is assumed to be 0 in both models. However, it is included in the derivations for completeness.

$$h_v(\psi) = \frac{\pi}{2}(1+\cos\psi)$$
$$h_r(\psi) = \rho\frac{\pi}{2}(1-\cos\psi)$$

As already said, the shading factor S is calculated as the sum of these two functions, divided by π:

$$S = \frac{h_v(\psi)+\rho h_r(\psi)}{\pi} = \frac{\frac{\pi}{2}(1+\cos\psi)+\rho\frac{\pi}{2}(1-\cos\psi)}{\pi}$$

Next, this equation is simplified and solved for $\cos\psi$, which gives:

$$\begin{aligned} \pi\frac{2}{\pi}S &= 1+\cos\psi+\rho-\rho\cos\psi \\ \Leftrightarrow 2S &= (1-\rho)\cos\psi+1+\rho \\ \Leftrightarrow \frac{2S-1-\rho}{1-\rho} &= \cos\psi \end{aligned} \tag{2.8}$$

Looking at image 2.7(a) and using $\cos\varphi = \frac{\text{adjacent leg of } \varphi}{\text{hypotenuse of } \varphi}$ it can be derived, that

$$\cos\psi = \frac{R-d}{R}, \tag{2.9}$$

where R is the radius of the hemispherical hill. Substituting the term $\cos\psi$ in equation (2.9) with equation (2.8) gives the following linear model (see figure 2.8 for the function graph):

$$\begin{aligned} &\frac{2S-1-\rho}{1-\rho} = \frac{R-d}{R} \Leftrightarrow \frac{-2RS+R+\rho R}{1-\rho}+R = d = \\ &\frac{-2RS+R+\rho R+R-\rho R}{1-\rho} = \frac{R(-2S+2)}{1-\rho} \Leftrightarrow 2R\frac{1-S}{1-\rho} = d \end{aligned} \tag{2.10}$$

2.5.2.3 Combined Model

The previously derived below-plane and above-plane model from equations (2.7) and (2.10) now have to be combined into one single model. First, the transition between both models is calculated as the intersection of both models. By substituting $S = \frac{1}{2}$ (as the zero-plane where height is 0) and solving for a, the result is:

$$2R\frac{1-S}{1-\rho} = a\sqrt{\frac{1}{S}-1} \Leftrightarrow \frac{R}{1-\rho} = a \tag{2.11}$$

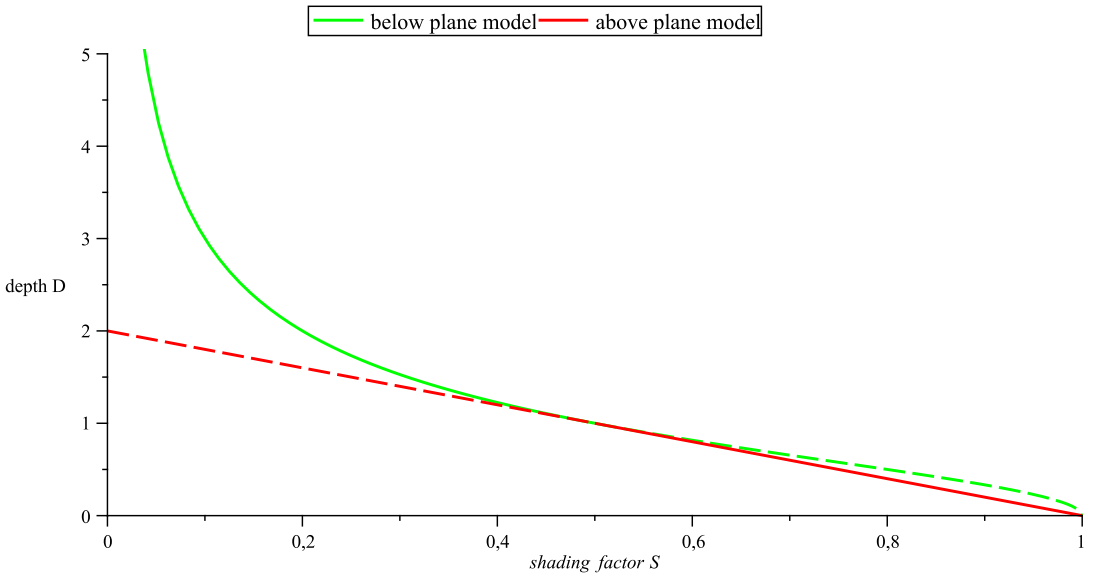


Figure 2.8: The graph of the function $D(S)$, which describes the depth depending on the shading factor S. The dashed lines show the unused parts of the above- and below-plane model.

As already said, the surrounding surface reflectance ρ is assumed to be 0. Therefore, the diffuse shading S, at each scale a (which describes cylinder aperture or hemisphere radius), can be calculated using the following, combined aperture formula:

$$D(S) = \frac{d}{a} = \begin{cases} \sqrt{\frac{1}{S} - 1}, & S \leq \frac{1}{2} \\ 2(1-S), & S > \frac{1}{2} \end{cases} \tag{2.12}$$

The function graph of the final depth estimation function $D(S)$ is shown in figure 2.8.

2.5.2.4 Multiscale Formulation

The combined depth model is not enough to obtain good looking results when estimating depth. The reason is that a shading change which occurs over large region, mostly corresponds to a greater change in depth than the same shading change over a small region.

Since the models derived previously estimate depth from shading relative to a specific feature size a, each scale in the diffuse shading image has to be considered separately. The previously described Laplacian pyramids with the shown implementation are used to separate the diffuse shading image into differently scaled layers. This converts the aperture estimates into depth estimates. The number of pyramid levels N is a user-defined constant, which depends on the image content. It reflects the level of detail

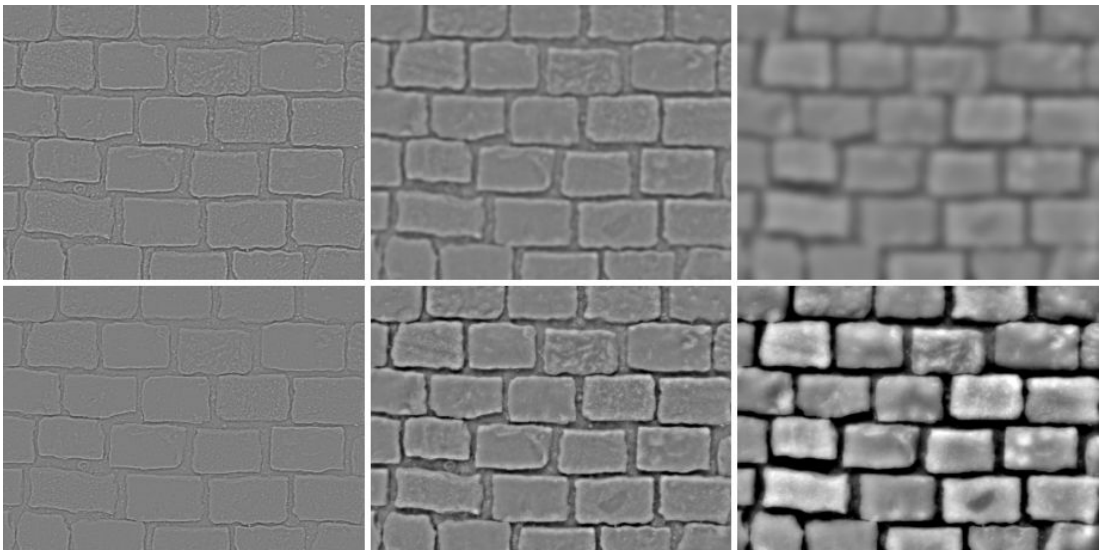


Figure 2.9: In the top row the Laplacian pyramid levels L_2, L_3 and L_4 are shown. In the bottom row some depth-maps using 3, 4 and 5 pyramid levels are shown.

of the final depth-map. The final depth $d(x,y)$ is obtained by using the depth function from equation (2.12) and solving for d. The aperture a is replaced by the blur radius r at each level. The resulting formula is

$$d(x,y) = \sum_{i=1}^{N} r_i \cdot (D(I_p^i(x,y)) - 1), \tag{2.13}$$

where I_p^i is the pyramid image at level i. It should be noted that in this equation depth units correspond to a pixel's width and have to be scaled accordingly. Further 1 is subtracted from the computed depths, because the diffuse shading image is normalised to a mean value of 0.5 and $D(S = 0.5) = 1$ is the depth at this average intensity. Consequently, subtracting 1 makes the average surface displacement to be zero.

Another point concerning the implementation is that the outputted depth-maps are inverted in the sense, that high depth values correspond to bright colours, and low depth values correspond to dark colours. In the implementation there is the possibility to invert the depth-maps. I did this, because on the one hand I think "the brighter the higher" depth-maps are more intuitive, and on the other hand, the later presented *relief mapping* algorithm, used for visualisation, requires such depth-maps in its current implementation. In figure 2.9 different levels of a Laplacian pyramid are shown together with some depth-maps using different numbers of pyramid levels.

2.5.2.5 Advantages and Disadvantages

The combined depth formula is very compact and fast to evaluate. In comparison to previous methods like [LZ94], no computational complex formulas or ray-tracing

(a) Diffuse image (b) Flash image (c) Depth-map

Figure 2.10: Rock wall example

schemes have to be approximated. However, the depth formula is of course a trade-off between simplicity and functionality. A more complex model will result in a more difficult formula, which may be more accurate in some situations, where the modelling of hills as hemispheres and valleys as cylinders may be insufficient. In many surfaces found in nature, valleys appear as crevices and hills as rough peaks. In such cases the presented model would not fail completely, but the results would not be as good as when using a more elaborate or customised model. Further, overhangs in surfaces cannot be reconstructed at all using the presented model. Here not only the model has shortcomings, but the input photographs do not contain enough information. Finally, surface interreflections are ignored, which mainly leads to biased depth-map brightness. To compensate for this a *uniform scaling factor* is applied to each depth-map to achieve an acceptable visual match to the original surface appearance. This means the final depth $d(x,y)$ from equation (2.13) is just multiplied by a scaling factor $a \in \mathbb{R}$. By using a negative a the depth-map can be inverted as previously mentioned.

2.6 Examples

Figure 2.10[7] and 2.11 show two results of the depth hallucination algorithm. It is clearly visible, that photographs of walls and floors seem to work really well. The reason is, that their structure is in general easy (they can be considered perfectly as a height field without global curvature) and their material is without any specular reflections. A detailed evaluation of the depth hallucination algorithm can be found in chapter 5.

2.7 Histogram Matching

It is not always needed and sometimes even not possible to take a diffuse-lit and a flash-lit picture pair of exactly the same surface cut-out. In the absence of a flash-lit image a technique called *histogram matching* [HB95, NFH07] can be used to match against a

[7]The rock wall image was taken from the paper [GWM+08].

(a) Diffuse image (b) Flash image (c) Depth-map

Figure 2.11: Bumpy wall example

visually similar picture for which a model of a captured picture pair have already been recovered. This drastically simplifies the capturing requirements for large surfaces, composed of the same material, but containing significant meso-structure variation (e g. large floors or walls).

2.7.1 Histograms

Mean value and *mean square deviation* are simple measures for characterising the distribution of greyscale values in an image. More information about the distribution of grey scale values can be obtained by looking at the *histogram of relative probabilities* of an image $I = (i(x,y,1))$:

$$\text{histo}_I : G \mapsto \mathbb{R}, g \to \frac{1}{s}|\{(x,y)|i(x,y) = g\}|$$

Such a histogram of an image is a *distribution function*, because the histogram is normalised by the number of pixels of I. Thus it applies, that

$$\sum_{g \in G} \text{histo}_I(g) = 1 \text{ and } 0 \leq \text{histo}_I(g) \leq 1, \forall g \in G.$$

In dark images with low contrast mainly the relative probabilities histo(g) for small g are high, whereas in bright images with low contrast mainly the histo(g) for big g are high. It should be noted, that a histogram does not allow a mapping between greyscale value probabilities and their spatial arrangement in the associated image [NFH07].

2.7.2 Histogram Specification

The goal of *histogram specification* is to modify an image $I_{\text{in}} = (i_{\text{in}}(x,y,1))$, such that its histogram $\text{histo}_{I_{\text{in}}}$ matches an arbitrary reference histogram $\text{histo}_{\text{ref}}$. Thus a mapping function

$$g' = f_m(g), g, g' \in G \tag{2.14}$$

is searched that converts the input image I_{in} into a new image I'_{in}, such that

$$\text{histo}_{I'_{\text{in}}} \approx \text{histo}_{I_{\text{ref}}}$$

Such a mapping function can be found by combining the two histograms (which are, as already mentioned, nothing more than distribution functions). For a given pixel value g from image I_{in}, the new pixel value g' can be obtained using

$$g' = \text{histo}_{\text{ref}}^{-1}(\text{histo}_{I_{\text{in}}}(g)). \tag{2.15}$$

This of course assumes that $\text{histo}_{\text{ref}}$ is invertible, which means that $\text{histo}_{\text{ref}}^{-1}(x), x \in [0,1]$ exists. This mapping can be interpreted as two distribution look-ups. First, the probability for a pixel g in the input image is looked-up. Then the pixel value in the reference histogram for the given probability of g is looked-up, which yields g' [BB07]. Therefore the final mapping function f_m from equation (2.14) can be written using equation (2.15) as

$$f_m(g) = g' = \text{histo}_{I_{\text{ref}}}^{-1}(\text{histo}_{I_{\text{in}}}(g)).$$

2.7.3 Algorithm

To adjust the histogram $\text{histo}_{I_{\text{in}}}$ of an input image to the histogram $\text{histo}_{I_{\text{ref}}}$ of a reference image, the technique above does not work in general. The problem is, that the distribution functions, described by the histograms, are not continuous ($\Rightarrow$ not invertable). For example, if in an image two different pixel values have a probability of 0, the histogram is not *bijective* anymore and thus cannot be inverted.

In the following, a method to match two histograms is described, which works with discrete histograms. The basic idea behind this method is to avoid inversions, but to calculate the mapping function f_m, by "filling in" the reference histogram $\text{histo}_{I_{\text{ref}}}$, by using *cumulative distribution functions* (CDF), which are defined as:

$$\text{CDF}_I(g) = \sum_{p \leq g} \text{histo}_I(p), g, p \in G$$

The advantage of using CDFs instead of distribution functions is that the former are *monotonically increasing*, which makes it easier to approximate their inverse. In general for a discrete CDF F it holds, that [BB07]:

$$\text{CDF}_I^{-1}(y) = \inf_{x \in G}\{\text{CDF}_I(g) \geq y\}. y \in [0,1].$$

The complete algorithm for calculating the mapping function f_m is shown in listing 2.2. It is important to note, that the following algorithm expects image pixels in range $0-255$. This is needed for indexing the look-up table (mapping function).

```
1  float[256]calcCdf(float histo[256], int pixelCount)
2  {
3      float cdf[256];
4
5      cdf[0] = histo[0];
6
7      for (int i=1; i<256; i++)
8      {
9          cdf[i] = cdf[i-1]+histo[i];
10     }
11
12     return cdf;
13 }
14
15 byte[256] matchHistograms(float histoIn[256], float histoRef[256])
16 {
17     byte fm[256];
18     float cdfIn = calcCdf(histoIn);
19     float cdfRef = calcCdf(histoRef);
20
21     for (int i=0; i<256; i++)
22     {
23         int j = 255;
24
25         do
26         {
27             fm[i] = j;
28             j--;
29         }
30         while (j >= 0 && cdfIn[i] <= cdfRef[j]);
31     }
32
33     return fm;
34 }
```

Listing 2.2: Algorithm for calculating the mapping function f_m

This algorithm can be expanded easily to multi-channel images by applying the algorithm separately on every channel.

2.7.4 Example

The following images show an example of using the histogram matching algorithm in the sense of depth hallucination. In figure 2.12, the sample input image pair is shown. Both photographs are of the same material and surface type, but were taken at different locations. In order to recover a depth-map for the diffuse-lit image, a flash-lit image has to be created. This is done by histogram matching the diffuse-lit image against the flash-lit image. In figure 2.13, the resulting flash-lit image in comparison to the reference image is shown. At first glance, the result looks very good. The image became a lot brighter and looks very similar to the reference image. On closer inspection it strikes out, that the contrast of the matched image is a lot higher than in the reference image. This is because the contrast in the diffuse image is higher too, and histogram matching does not change the contrast of an image. More problems

(a) Diffuse-lit input image

(b) Flash-lit reference image

Figure 2.12: Histogram of left image is to be matched to histogram of right image

(a) Result of histogram matching

(b) Reference taken with flash

(c) Difference image

Figure 2.13: Comparison of result of histogram matching and the reference image

may arise when in the image to match appear colours, which are not contained in the reference image. In this case the mapping function may fail. Situations like this are tested more detailed in the evaluation in chapter 5.

Chapter 3

Image Segmentation

3.1 Motivation

The depth hallucination algorithm has limitations on certain images. Especially, it has problems, when there are translucent or reflective regions in an image (e. g. a window or a small puddle, reflecting the environment), because the correct surface appearance cannot be captured correctly. In this case it would be desirable to exclude these regions from the depth hallucination process and let the user specify a constant depth value. Hence a method is needed to select regions. To make selecting regions easy and convenient, it would be desirable to automate this step as much as possible. In the following it is shown how *image segmentation* methods can be used for semi-automatically selecting regions, which are then excluded from the remaining depth hallucination process.

3.2 Definition of Image Segmentation

Image segmentation means the process of partitioning a digital image into multiple sets of pixels, so called *segments*, whose union is the entire image. The goal of image segmentation is to separate an image into regions, which gives the image more meaning or makes it easier to analyse it. More formally, given an image $I = (i(x,y,c))$ with pixel positions

$$P = \{(x,y)|x \in \{1,\ldots,w\}, y \in \{1,\ldots,h\}\},$$

a segmentation of I can be defined as a set of $n \geq 2, n \in \mathbb{N}$ disjoint, non-empty segments $S_i \subset P, i = 1,\ldots,n$, which fulfil the following requirements [Zha06]:

1. $\bigcup_{i=1}^{n} S_i = P$,
2. S_i is a connected segment, for $i = 1,\ldots,n$
3. $S_i \cap S_j = \varnothing, i \neq j,\ i,j = 1,\ldots,n$

Figure 3.1: Seed strokes for advising the algorithm, which pixels definitely belongs to which segment

3.3 Graph-Cut

In this section the principle way of using graphs for segmenting an image in n regions is described. First, a brief overview of the *graph-cut* algorithm for just two regions (*foreground* and *background*) is given. Next it is shown how the graph is constructed based on the image's pixels and seed strokes. After that, it is presented, how the edge weights in the graph are calculated. In the last section the basic graph-cut algorithm is expanded to an arbitrary number of segments.

3.3.1 Overview

The graph-cut approach for segmenting images was first applied in computer vision by Greig et al. [GPS89] in 1989. Graphs from graph theory are used to compute a global segmentation of an image with two regions. The most important advantage of graph-cut based methods to other approaches is its reduced noise sensitivity and a less probable leakage between two segment boundaries, due to the calculation of a global solution. As input the image to segment is needed. Additionally a set of so-called *seed strokes* is required.

Seed Strokes Full automatic segmentations rarely yield expected results. Real-life images mostly do not contain very distinct regions which can be easily detected as such. Therefore it is vital to roughly advise the algorithm, which parts of an image belong to the foreground, and which to the background. This corresponds to a limitation of the search space, which improves the quality of the results and reduces execution times as well. Advises for the algorithm are given by marking certain regions in the image with differently coloured lines. These lines are called *seed strokes*. An image together with some seed strokes is shown in figure 3.1.

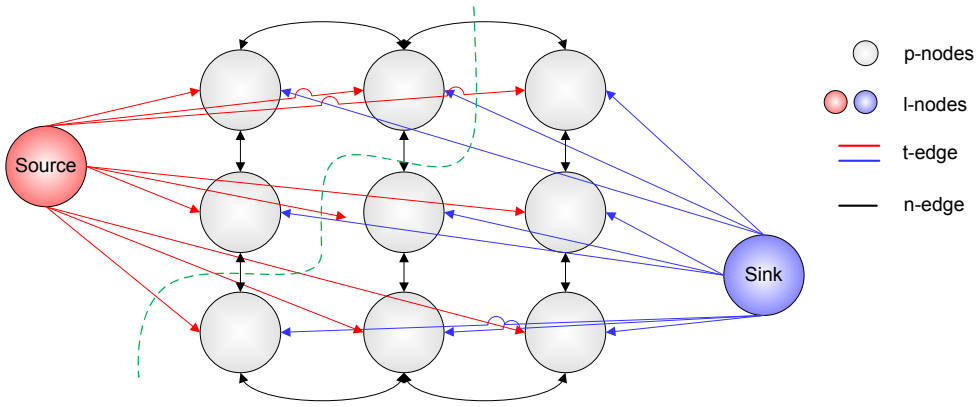


Figure 3.2: Result of transforming an image into a graph for segmentation

3.3.2 Graph Construction

As a start a graph G has to be constructed on which the graph-cut algorithm can be applied later on. This graph contains two different kinds of nodes. The so called p-nodes represent the pixels of the input image $I = (i(x,y,c))$. Additionally two *terminal nodes* s and t, so called l-nodes, are added to the graph. s is the source and t the sink node, which represent the object foreground and background respectively. Bidirectional edges connect the p-nodes and every p-node is connected via a directed edge with both terminal nodes. The p-nodes could be connected in an arbitrary configuration. However, in practice the usage of complex graphs (for example by using diagonal edges) does not yield significant better segmentation results, but higher execution times [BK01]. Hence in the following only grid configurations are taken into account. The edges between p-nodes are called n-links, the edges to and from terminal nodes are called t-links.
The computation of the edge weights are based on *Pott's Energy Interaction Model* [Pot52]. After setting reasonable edge weights, a *minimum cut* is calculated which separates the node set V into two disjoint sets. These sets represent the segmented foreground and background of the image. Such an image-to-graph transformation is shown in figure 3.2 for an image of 3×3 pixels.

3.3.3 Flow of a Graph

Let V be a set of vertices and E be a set of edges. A *s-t-graph* (also called *flow network*) $G = (V,E)$ is defined as a connected, directed, finite graph with positive capacities $c(u,v) \geq 0, (u,v) \in E$. If $(u,v) \notin E$, $c(u,v) = 0$ is assumed. Furthermore, two special nodes are distinguished in G: a source node $s \in V$ and a sink node $t \in V$, where s may only have outgoing, t only incoming edges. The *flow* of such a s-t-graph is a real-valued function $f : V \times V \rightarrow \mathbb{R}$, with the following properties:

Capacity Constraint

$$f(u,v) \leq c(u,v), \forall u,v \in V \tag{3.1}$$

The flow along an edge cannot exceed its capacity.

Skew Symmetry

$$f(u,v) = -f(v,u), \forall u,v \in V \tag{3.2}$$

The flow from node u to node v must be the opposite of the flow from node v to node u.

Flow Conservation

$$\sum_{v \in V} f(u,v) = 0, \forall u \in V \backslash \{s,t\} \tag{3.3}$$

For all nodes, except source and sink, the sum of the incoming flow has to be the same as the outgoing flow.

3.3.4 Maximum Flow and Minimum Cut

The problem of *maximum flow* is to find the maximum amount of flow from the source to the sink in a *s-t*-graph. A so called *s-t-cut* of a graph G is defined as the partitioning of the nodes V into two disjoint sets S and $T = V \backslash S$, so that $s \in S$ and $t \in T$. For a given flow f the *net flow* $F(S,T)$ of a cut (S,T) is defined as the sum of all flow from S to T:

$$F(S,T) = \sum_{u \in S} \sum_{v \in T} f(u,v)$$

The *capacity* of a cut (S,T) is defined in a similar way as the sum of the capacities of all edges from S to T:

$$C(S,T) = \sum_{x \in S} \sum_{y \in T} c(u,v).$$

A cut is a *minimum cut*, if the capacity of this cut is not larger than the capacity of any other cut.

Definition 1 (Cut-Flow-Theorem)**.** Let f be the flow of a *s-t*-graph $G = (V,E)$ with source $s \in V$ and sink $t \in V$. Then, the maximum flow of G is the same as the minimum cut of G.

To determine the minimum cut of a *s-t*-graph, it is therefore sufficient in order to calculate its maximum flow. The proof of this theorem is omitted, because it would rise above the scope of this thesis. The proof can be found in [FF62].

3.3.5 Flow Algorithms

There are a lot of algorithms for calculating the maximum flow of a flow network. Most of them can be classified as *push-relabel* algorithms according to Goldberg-Tarjan, or as *augmenting path* algorithms according to Ford-Fulkerson.

Augmenting Paths The concept of the original algorithm of Ford and Fulkerson is to start with a flow of $f(u,v) = 0$ for all edges in the flow network. Then successively *augmenting paths*[1] from source to sink are searched in the *residual graph*[2], which increase the flow. The worst-case execution time of this algorithm is $O(f_{\max} \cdot |E|)$, where $|E|$ is the number of edges and $f_{\max}$ is the maximum flow in the flow network [FF62].

Push-Relabel The basic idea of the push-relabel algorithm by Goldberg and Tarjan is to not use augmenting paths, but push as much flow as possible from source to neighbouring nodes. Then the flow is *pushed* further step by step towards the sink. This yields incorrect flows in the intermediate steps, because the flow conservation property (3.3) is violated. Hence so called *preflows* are used instead of flows. The worst-case execution time of this algorithm is $O(|V|^2 \cdot |E|)$, where $|V|$ is the number of nodes and $|E|$ the number of edges in the flow network [GT88].

Algorithm of Boykov and Kolmogorov In the area of computer vision it has been shown, that augmenting path based algorithms are more efficient than push-relabel based algorithms. Although the runtime complexity of push-relabel techniques is superior, a little modification on the augmenting path algorithm makes it empirically (in practice) faster. The algorithm of Boykov and Kolmogorov [BK01] uses, similar to other augmenting path algorithms, search trees for finding augmenting paths from source to sink. The major difference to the standard algorithm by Ford and Fulkerson is that two search trees are used instead of just one. One search tree contains the source node as root and the other the sink. When calculating the maximum flow both trees grow towards each other, effectively reducing execution time. The worst-case complexity of this algorithm is $O(f_{\max} \cdot |V|^2 \cdot |E|)$, where $|V|$ is the number of nodes, $|E|$ the number of edges, and $f_{\max}$ the maximum flow in the flow network. Even though the worst-case complexity is worse than the worst-case complexity of the two standard algorithms it turned out that in practise the algorithm is much faster. In table 3.1[3] the execution times of different maximum flow algorithms are shown in comparison to the algorithm of Boykov and Kolmogorov.

3.3.6 Energy Functions

Image segmentation algorithms, and of course graph-cut as well, basically try to find the optimal solution to a binary problem: assigning each pixel in an image a *label*, which represents its associated segment. Of course, constraints have to be defined,

[1] An *augmenting path* is a path from source to sink in the residual graph, where each edge has a positive capacity. Such a path can be used to further increase the flow.

[2] A *residual graph* gives the amount of remaining capacity for each edge. Its topology is identical to the flow network, but its edges are set to the remaining capacities, with respect to the current flow.

[3] The table was taken from the paper [BK01].

Algorithm	Bell photo	Lung CT	Liver MR
DINIC	2.73 s	2.91 s	6.33 s
H_PRF	1.27 s	1.0 s	1.94 s
Q_PRF	1.34 s	1.17 s	1.72 s
Boykov/Kolmogorov	0.09 s	0.22 s	0.20 s

Table 3.1: Comparison of execution times of different maximum flow algorithms for segmenting images

which describe in which fashion the labels are assigned. An important prerequisite is that the labels are homogeneous in regions of low discontinuities, but strong discontinuities have to be kept. These constraints can be formulated by defining the pixel labelling problem as the task of minimising an *energy function*. This minimisation can be computed using a graph-cut.

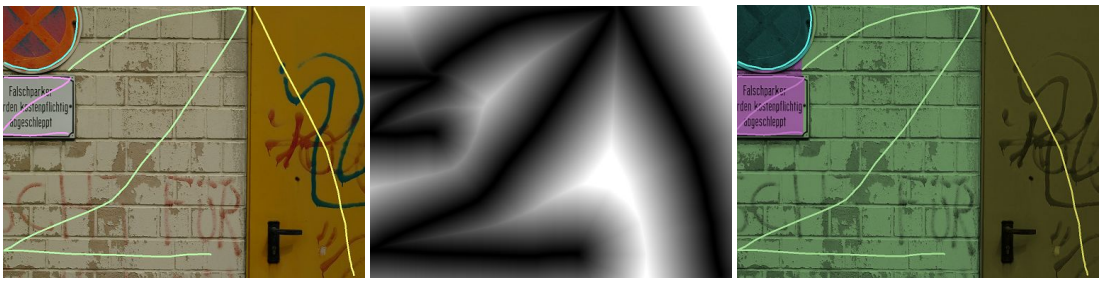
A popular energy model is *Pott's Interaction Energy Model* [Pot52], which is defined as

$$E(f) = \underbrace{\sum_{p \in P} D_p(f_p)}_{\text{data term}} - \underbrace{\sum_{p,q \in N} V_{p,q}(f_p, f_q)}_{\text{interaction term}},$$

where P is the set of image pixels, $f = \{f_p | p \in P\}$ the unknown labelling and $N \subset P \times P$ a neighbourhood of pixels. D_p is called the *data term* and $V_{p,q}$ the *interaction term*. The data term describes the cost of labelling a pixel p with label f_p. The interaction term penalises two neighbouring pixels p and q with labels f_p or f_q, if they are too different. So both terms have to be defined in a way that the energy function's minimum should correspond to a good segmentation. Depending on the problem, the interaction term can be used to prefer continuities or discontinuities between pixels [LVJ07].

3.3.6.1 Data Term

For the data term the Euclidean distance between a pixel and the initial seed strokes for the foreground are used. These distances can be calculated for example using the *8SED* algorithm [Dan80] or *chamfer distance transforms* [RP66]. Applied on an image, these methods calculate a field of vectors, containing the distances to the nearest pixel which is not black. As data term applied on the foreground seed strokes, this can be interpreted as the more the distance of a pixel p to a pixel q of a foreground stroke increases, the less probable it is, that p is classified as a foreground pixel. In figure 3.3(b) the distance transformation for the seed strokes from figure 3.3(a) are shown.



(a) Seed strokes (b) Distance transformation of foreground seed strokes in first iteration (c) Segmentation result

Figure 3.3: Seed strokes, their distance transformation, and the segmentation result

3.3.6.2 Interaction Term

As already mentioned, the goal of the interaction term is to yield high energies in homogeneous regions and low energies in regions with discontinuities. This makes the graph-cut algorithm cutting between nodes (pixels) with strong discontinuities, because the edge capacities represented by the interaction term are low. For the interaction term a lot of different functions exist, which range from easy, intensity based to more elaborate terms. In the following two interaction terms are presented: one pixel based and one edge based.

Exponential Intensity Differences A relatively simple interaction term is to take the intensity difference between two neighbouring pixels p and q as the reciprocal of an exponential function [GWW99, SM97, BFL06]. This interaction term is especially applicable for greyscale images, because pixel intensities (greyscale values) are compared. Segment leakage between differently coloured image regions with the same brightness arise when using the term on multi-channel images.

$$V_{\text{exp}}(p,q) = e^{-\frac{(\text{grey}(p)-\text{grey}(q))^2}{\sigma^2}} \tag{3.4}$$

The term penalises a lot for discontinuities between pixels of similar intensity when $|\text{grey}(p) - \text{grey}(q)| < \sigma^2$. However, if pixels are very different, $|\text{grey}(p) - \text{grey}(q)| > \sigma^2$, then the penalty is small. Intuitively, this function corresponds to the *exponential distribution* of noise among neighboring pixels of an image.

Edge (Gradient) Differences Edges in images are found in regions with sharp colour and/or brightness changes. Thus the *gradient* of an image used for edge detection can be utilised as interaction term. It yields small values for strong discontinuities between the pixels p and q and high values for weak discontinuities.

$$V_{\text{grad}}(p,q) = \frac{1}{||\Delta p|| + ||\Delta q||} \tag{3.5}$$

3.3.7 Expanding to Multiple Segments

The graph-cut algorithm works inherently only for cutting an image into two segments, because the graph only has one source and one sink node, representing foreground and background. To expand the algorithm to n segments, the basic graph-cut algorithm is applied iteratively, whereas the remaining pixels to segment decrease in each iteration. Given a set of $n > 2, n \in \mathbb{N}$ seed strokes the following steps are performed iteratively.

1. Use the first seed stroke to mark the image's foreground and all the remaining seed strokes to mark the background.
2. Calculate a segmentation using graph-cut. The result is two segments: foreground and background.
3. Remove the first stroke (previously used for the foreground) and remove all pixels of the foreground from the image.
4. Go to the first step.

3.3.8 Summary

Graph-cut achieves robust, global segmentations and even difficult images can be segmented by using many seed strokes. Furthermore, the algorithm can be easily expanded to arbitrary dimensions, for example for segmenting three-dimensional data, as it is often required in medical applications.
On the one hand the quality of the segmentation is highly depending on the used energy function and parameters. On the other hand, the same energy function and choice of parameters do not yield the same quality in all fields of application. Thus it is necessary to choose a proper energy function depending on the image data. Furthermore, the graph-cut method is limited by the restricted possibility of user interaction in the segmenting process, which is mainly determined by the capacities and topology of the graph and its edges. Another disadvantage is that the number of segments must be known before segmentation (by the number of seed strokes). Some examples and an evaluation of the graph-cut algorithm are presented in chapter 5.

Chapter 4

Implementation

Computer graphics is a very practical discipline. Therefore it is especially important, to implement a certain technique, to show its practical usefulness in terms of performance and quality of the results. In this chapter important aspects of the implementation of the different previously presented algorithms are shown. In addition to the algorithm implementation I wrote a user friendly tool, which supports creation, previewing and storing of depth-maps. It is supposed to be easily capable of being integrated into an existing content generation pipeline (in particular into the Virtual Aachen Project). The visualisation modes of the tool are also presented in this chapter.

4.1 Used Libraries

For implementing the algorithms and the tool, I used some well known programming libraries. This allowed me to concentrate on the implementation of the algorithms and the tool it-self, instead of reinventing the wheel again by writing elementary and repeating core functionality.
In the following the libraries which were used in the implementation are presented in short. I paid attention, that all libraries are platform independent, that the implementation can be compiled and executed using different operating systems. Windows, Linux and MacOS were tested.

Qt Toolkit The Qt Toolkit[1] is an open-source cross-platform application framework, mainly used for the development of applications with rich user-interfaces. Over the time the Qt library has grown from a GUI-only framework to a full featured application and communication framework. Therefore, it can also be used for non-GUI applications such as console tools and servers.

OpenCV **Open C**omputer **V**ision[2] is an open-source cross-platform library for digital image processing and computer vision. It was originally developed by Intel and

[1] `http://www.qtsoftware.com`
[2] `http://sourceforge.net/projects/opencvlibrary/`

is free for research and commercial use. The library is highly optimised and especially targeted on Intel hardware.

GLEW The Open**GL** **E**xtension **W**rangler Library[3] is an open-source cross-platform library for conveniently loading OpenGL extensions and determining if a certain extension is supported on the target platform.

4.2 Compilation

First all above mentioned libraries have to be installed on the target system. As already said, all libraries are platform independent. Therefore I used two different project types for compilation under Unix operating systems (Linux and MacOS were tested) and Windows (XP and Vista were tested).

Unix For compilation under Unix a `.pro`-file (Qt Project) is available in the directory `depthhallu/unix`. `qmake` is used to generate a makefile on the target platform. To compile the project just execute the following commands:

1. `qmake depthhallu.pro`
2. `make`

Windows For Windows a Microsoft Visual C++ 2008 solution is available in the directory `depthhallu\win32`. Of course `qmake` can also be used to generate a makefile under Windows. If Visual C++ is used instead of `make`, the Microsoft program `nmake` has to be used.

4.3 Algorithms

4.3.1 Depth Hallucination

In the implementation of the depth hallucination algorithm, I used the OpenCV library for a lot of basic work like loading and storing images. However also a non-trivial task could be solved using OpenCV. Calculating the Laplacian pyramid was implemented using the `cvSmooth`, `cvPyrUp` and `cvPyrDown` functions. Programming of the rest of the algorithm was a straight transformation of the described algorithm into C++ code.

4.3.2 Graph-Cut Segmentation

For calculating the maximum flow of a graph I used the reference implementation of the algorithm presented in [BK01] by Yuri Boykov and Vladimir Kolmogorov[4]. The

[3] `http://glew.sourceforge.net`

[4] `http://www.cs.ucl.ac.uk/staff/V.Kolmogorov/software/maxflow-v3.0.src.tar.gz`

Edge	Pixel membership	Capacity
$\{p,q\}$	n-link	$V_{p,q}$
$\{p,s\}$	p is not a seed-point	D_p
$\{p,s\}$	p is object seed-point	∞
$\{p,s\}$	p is background seed-point	0
$\{p,t\}$	p is not a seed-point	D_p
$\{p,t\}$	p is object seed-point	0
$\{p,t\}$	p is background seed-point	∞

Table 4.1: Summary of how to assign the capacities of the different edge types in the graph used for graph-cut image segmentation

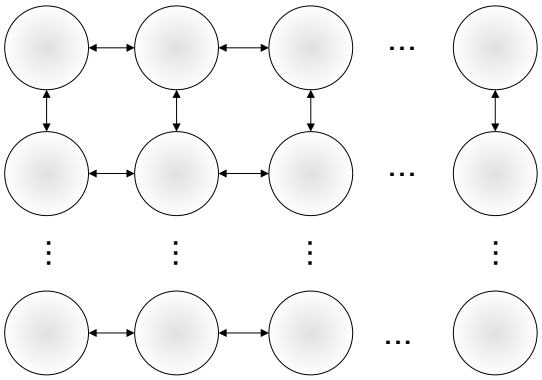


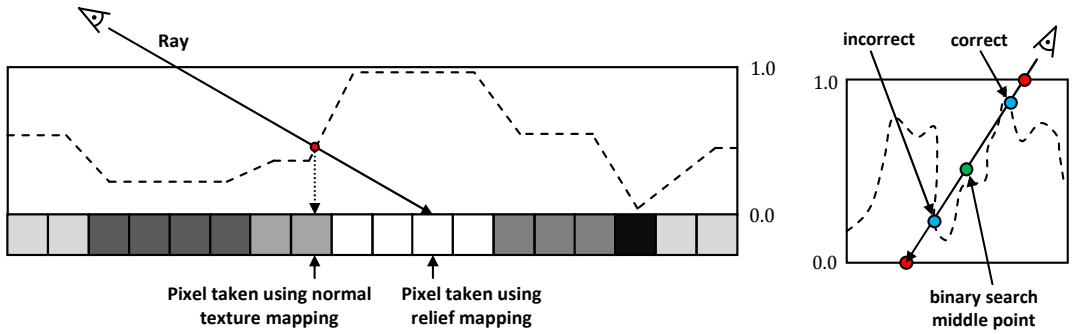
Figure 4.1: A grid-graph as it is used in the graph-cut algorithm

computation of the 8SEG algorithm was solved using the OpenCV library, in the sense of calculating a distance transformation using `cvDistTrans`. The assignment of the edge capacities is summarised in table 4.1. The p-nodes (representing the pixels in the graph) are connected using bidirectional edges, where each p-node is connected with its right and its bottom neighbour. This effectively yields a *grid-graph*, as shown in figure 4.1.

4.4 Visualisation

Visualising the results of the depth hallucination algorithm is very important to get a visual feedback about the quality and correctness of the results. I implemented different visualisation modes in the tool, which are explained in the following.

Relief Mapping This is the most advanced and most realistic visualisation mode, which could be used for visualisation in practise. It uses the generated depth-maps for adjusting the texture coordinates, yielding in much more realistic ren-



(a) Visualisation of ray tracing the depth-map

(b) Problem when searching intersection point

Figure 4.2: Illustration of relief mapping and problem of searching the intersection points

derings. As this technique is a bit more complex, it is described in detail in the following section.

Terrain In this visualisation mode, a terrain (of triangle strips or points) is generated from the depth-map. This terrain mesh can be exported to make comparisons to other terrains. Further in this visualisation hills and valleys can be seen in 3D.

Standard Rendering Here standard diffuse texture mapping with *bilinear filtering* is used for rendering. This mode was implemented to make the difference between standard texture mapping and *relief mapping* directly comparable.

4.4.1 Relief Mapping

Relief mapping is an advanced texture mapping technique, which allows the rendering of highly detailed surfaces without the need of additional geometry. It was first presented by Oliveira et. al in 2000 [OBM00]. An alternative to relief mapping is *parallax mapping* [KTI+01, Wel04], which is less accurate but faster.
Relief mapping basically applies a texture coordinate correction, which is calculated by *short-distance* ray tracing a depth-map in a pixel-shader, as presented in [PNC05]. This correction is desirable, because different texels are hit when viewing a bumpy surface from different angles. This effect is illustrated in figure 4.2(a).

One diffuse texture-map and one depth-map are required as input. The height values in the depth-map are normalised to the range $[0, 1]$. From bottom to top the values increase from 0 to 1. In figure 4.3, a sample input texture pair is shown. In a pixel shader, the intersection point of the viewing ray and the height field represented by the depth-map is searched. Two different searches are required: a linear search first, and a binary search afterwards. The linear search is required, because the first midpoint of

(a) Diffuse-map (b) Depth-map

Figure 4.3: Example input texture pair

(a) Normal texture mapping (b) Relief mapping

Figure 4.4: Differences between default texture mapping and relief mapping

the binary search could lie outside the height-field surface, but the viewing ray could have already hit the surface before. This problem is illustrated in figure 4.2(b).
The result of using relief mapping in comparison to normal texture mapping is shown in figure 4.4. It is clearly visible, that in the visualisation which uses relief mapping the terrain looks a lot more three-dimensional and realistic.

Chapter 5

Evaluation

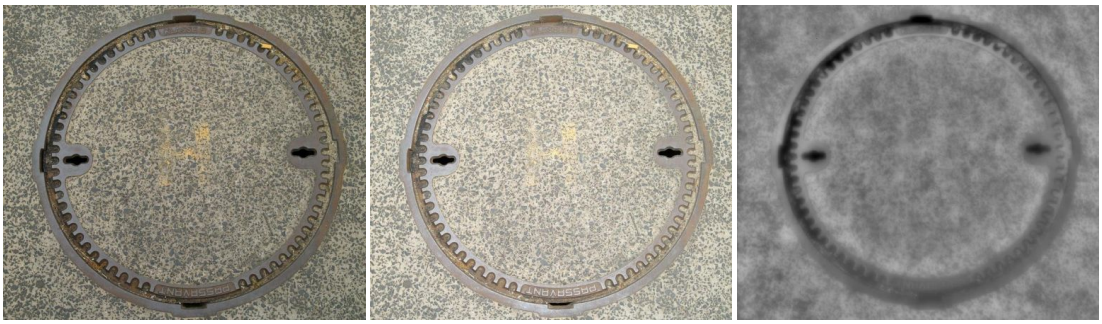
In this chapter the previously presented algorithms (depth hallucination, histogram matching and image segmentation using graph-cuts) are evaluated. The focus is on depth hallucination, because this algorithm is the essential part of this thesis. Histogram matching and graph-cut are only evaluated relating to the usage of depth hallucination. An important aspect is the applicability of depth hallucination for reconstructing depth-maps of facades, because this is very important for the Virtual Aachen Project.

5.1 Depth Hallucination

In this section the depth hallucination algorithm is evaluated. First of all some well working reconstructions are shown together with their input image pairs. Later on image pairs which could not be reconstructed as expected are shown and it is analysed why the reconstruction does not work.

5.1.1 Well Working Examples

Some well working examples were already shown in section 2.6. These images (see figure 5.3 and 5.4) are again listed here, together with some new examples (see figure 5.1 and 5.2).
All input image pairs on which the algorithm is working pretty well are showing floors or walls. The reason is, that on this kind of surface the assumptions that were made are appropriate. Firstly, they can be perfectly represented as height fields. Secondly, they are diffusely illuminated and their materials do not have any specular reflections. A third reason is that the camera can easily be oriented perpendicular to the surface and the distance from the camera to the surface is not too big. This way, the flash has enough power to strongly brighten up the flash-lit image. In summary one can say, that depth-maps of walls and floors can be reconstructed very well using depth hallucination.



(a) Diffuse image (b) Flash image (c) Depth-map

Figure 5.1: Gully cover example

(a) Diffuse image (b) Flash image (c) Depth-map

Figure 5.2: Church wall example

(a) Diffuse image (b) Flash image (c) Depth-map

Figure 5.3: Rock wall example

(a) Diffuse image (b) Flash image (c) Depth-map

Figure 5.4: Bumpy wall example

(a) Diffuse image (b) Flash image (c) Depth-map

Figure 5.5: Ivy wall example: Here the surface cannot be plausibly represented as a height-field

5.1.2 Problems and Limitations

There are cases in which the resulting depth-maps and rendered surfaces do not look as expected and/or desired. These are situations, in which the made assumptions do not hold. In the following some samples, where the algorithm yields unexpected results, are examined and analysed.

The first case is a rock wall with ivy twines, which are physically separated from the stone surface below, as shown in figure 5.5(a). This separation violates the basic assumption that the underlying surface may be plausible represented as a height field. As a result, the mathematical model fails, because hills and valleys cannot be described using hemispheres and cylinders anymore. Hence the resulting depth-map is wrong in regions with ivy twines, as shown in figure 5.5(c).

Images with abrupt depth changes seem to be difficult too, because often a little self-shadowing occurs, as shown in image 5.6(a). In this case, the assumption of a completely diffuse illumination is violated. The result is a depth-map, which is poorly pronounced in regions of shadows, as shown in figure 5.6(c).

(a) Diffuse image (b) Flash image (c) Depth-map

Figure 5.6: Archway example, showing problems due to self-shadowing in regions of heavy changes in depth

(a) Diffuse image (b) Flash image (c) Height field

Figure 5.7: Cobblestone example 2, having problems because of a directional light casting shadows

A last example for an image pair, where the algorithm fails is shown in figure 5.7(a)[1]. Here a directional light casts shadows in one direction. Thus the diffuse illumination assumption is violated, because light is not incident equally from all directions any more. A height field visualisation of the depth estimation shows, that the overall structure of the surface could not be reconstructed, because of the shadows.

5.1.3 Reconstruction of Facades

For the Virtual Aachen Project it would be desirable, to not only be able to reconstruct walls and floors, but also whole facades. It turned out that reconstructing depth-maps of whole facades is quite difficult. During the reconstruction process, different problems arise, which are visible in figure 5.8(c). The three most major problems are described in the following.

1. For reconstructing an entire facade, a large region has to be photographed. This can only be accomplished by positioning the camera and flash far away from the

[1]The image was taken from `http://www.bourgetbros.com/site-admin/images_lotus/20112008_2046351_Multiblend%20Cobbles%20Tumbled%20Installed.JPG`.

(a) Diffuse image (b) Flash image (c) Depth-map

Figure 5.8: Facade example having different problems: low brightness (blue), not allowed materials (marked red), overhangs and elevations with a not perpendicular light direction (marked green)

facade. The major problem now is the distance of the camera and flash to the surface. Due to this distance, the brightness in the flash-lit image is quite low, because the flash has not enough power. The results are poor and sometimes even wrong depth-maps. More precise, regions of exactly the same material, but with drastic differences in brightness (marked blue in the depth-map).

2. Another problem is that some facade elements (e. g. windows or doors) violate the assumption, that the underlying surface material does not contain specular or reflecting regions. It is clearly visible, that the algorithm failed for the windows, which appear black and grey for the same depth in the depth-map (marked red in the depth-map).

3. Facades contain overhangs and elevations. Additionally the direction of incident light is not perpendicular to the facade. Thus the above plane model does not hold anymore. The result are mainly quite dark colours in the regions of the overhangs and elevations (marked green in the depth-map).

At a first glance it seems, that capturing whole facades is impossible, because of the three major problems described previously. At a second glance it turned out, that at least some of these problems can be solved.

Regions of materials which violate the specular reflection assumption can be excluded from the hallucination process, by selecting them in advance. The selection step can be automated using image segmentation as described earlier. The depth for the selected regions can then be specified by the user. The effect of using different depth-maps, for visualising the facade shown in figure 5.8(a) using relief mapping is shown in figure 5.9.
For solving the low brightness issue I tried to capture some flash-lit images of the facade from a close range. Afterwards I merged these images into one (by resize and copy and paste) and used this image as reference image in the histogram matching

(a) Relief mapping using original depth-map, see figure5.8(c)

(b) Relief mapping using adjusted depth-map

(c) Adjusted depth-map

Figure 5.9: Effect of visualising the same surface using the original and the adjusted depth-maps

(a) Combination of near range flash images

(b) Matched flash image

(c) Depth-map

algorithm in order to approximate a flash-lit image. I hoped to be able to calculate a flash-lit image this way. It turned out, that this does not work very well, as shown in figure 5.9. The depth-map does not contain these drastically brightness changes in areas of same depth, but the overall quality is still bad. However, in regions of overhangs and elevations (e. g. above the windows), depth could be estimated a lot better than before.

5.1.4 Performance

At the end of this section a short performance analysis is shown. In table 5.1 the execution times needed for generating the depth-maps of the previous examples are shown. All tests were done on a Intel Core 2 Duo with 2.5 GHz and 4 GB main memory. One can clearly see, that the execution times of my quite unoptimised implementation mainly depends on the size of the images and the level of detail ($\widehat{=}$ number of Laplacian pyramid levels) used. This appears due to the computational costs for calculating a Gaussian Laplacian pyramid and evaluating the combined depth model for each pixel in each pyramid level. All other image-based operations are quite cheap, because they are performed pixel-wise. This means their computational complexity is linear in the

Image	Size	Level of detail	Execution time
Cobblestone	1024×769	3	406 ms
Cobblestone	1024×769	4	582 ms
Cobblestone	1024×769	5	1.005 s
Archway	2048×1538	3	1.708 s
Archway	2048×1538	4	2.490 s
Archway	2048×1538	5	4.504 s
Rock wall	900×603	5	667 ms
Bumpy wall	4288×2848	5	15.870 s

Table 5.1: Execution times of depth hallucination algorithm for different images

number of image pixels: $O(wh)$. In summary, for medium sized images, as they are used mainly in visualisations (e. g. 512×512 or 1024×1024 images), the performance is already acceptable. If larger images have to be processed frequently, the algorithm has to be optimised.

5.2 Histogram Matching

Histogram matching is a great simplification and improvement for image acquisition. Its strength and flexibility can be demonstrated further, by reconstructing depth-maps of photographs, downloaded from the internet. In figure 5.10 and 5.11 the results of estimating depth of two downloaded images[2] are shown. Both images were matched against the flash-lit cobblestone pavement image from figure 2.2(b). The structures of the two images are very different, but the reconstruction works, as the depth-maps look correct. For the histogram matching, only the overall image histogram counts (which follows from the surface material). Thus images of arbitrary surface structure can be used as long as their material is similar enough.

As the histogram matching works channel by channel, another interesting question is what happens, if the greyscale histogram of one or more channel(s) of the image to match against has a heavy brightness shift. In order to investigate on this, I painted some red, green and blue coloured rectangles on the input image. The results are shown in figure 5.12. It is clearly visible, that the overall image colour is shifted towards the inverse of the shifted channel colour. For example for the red shift shown in figure 5.12(a), the overall colour is shifted towards $(1,1,1)^T - (1,0,0)^T = (0,1,1)^T$, which is cyan. Applying depth hallucination on such images, where the flash-lit image

[2]The rock wall image was taken from `http://www.katalog.foto-lizenzfrei.de/hintergrund/mauer-1.jpg`. The cobblestone pavement image was taken from `http://www.themarkeffect.com/JPG%20Photo%20Textures/Cobblestone%20Texture.jpg`.

(d) Diffuse image (e) Matched flash image (f) Depth-map

Figure 5.10: Result of histogram matching the cobblestone image from the Internet against the cobblestone flash-lit image

(a) Diffuse image (b) Matched flash image (c) Depth-map

Figure 5.11: Result of histogram matching the rock wall image from the Internet against the cobblestone flash-lit image

(a) Red channel shifted (b) Blue channel shifted (c) Green channel shifted

Figure 5.12: Matched images with different channels having a heavy brightness shift

(a) Seed strokes (b) Segmentation result

Figure 5.13: Sign wall segmentation

is approximated using histogram matching does not work of course. The problem is, that in the regions of the differently coloured object the surface material differs too much.

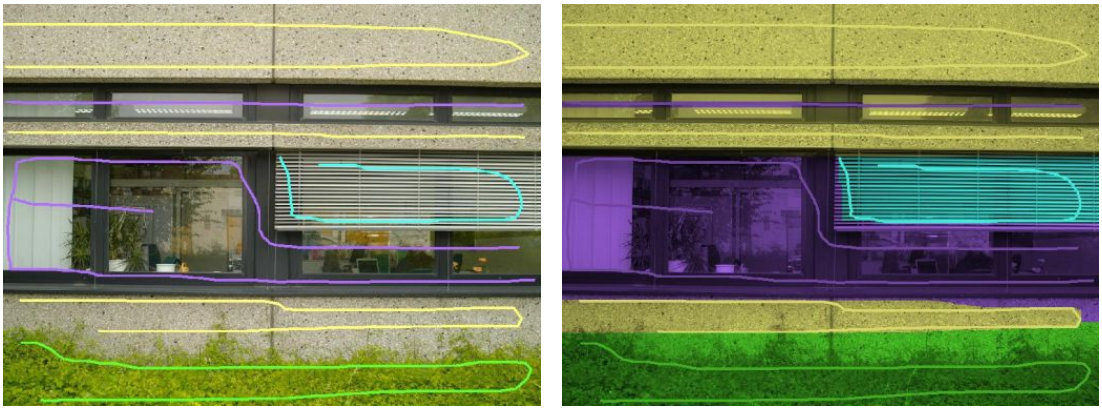
In summary one can say, that histogram matching is a great simplification and advancement for image acquisition. As long as the surface and material parameters of the two images are similar, the algorithm works very well and produces satisfying results. As soon as completely different colours arise, histogram matching gives incorrect results in these regions. However, for depth hallucination this is not a serious problem, because the images to match will in general have quite similar materials.

5.3 Image Segmentation

The graph-cut algorithm is only evaluated in regard to the usage in depth hallucination. This means only images of facades are tested.
In general the segmentation works quite well, because facades mainly consist of equally coloured regions (e. g. windows, doors, walls). The major problem is to get a sharp separation between the different regions. In the following figures some examples of using graph-cut based image segmentation on images of facades are shown. As interaction term the edge differences term shown in equation (3.5) was used and the Euclidean distance was used as data term. The segmentation of the images shown in the figures 5.13 and 5.14 (1024×766 pixels) takes about 3 seconds on a Intel Core 2 Duo with 2.5 GHz and 4 GB main memory. In practise it turned out, that the edge differences interaction term works quite well on facade images. The exponential intensity differences only yield acceptable results in greyscale images, because it does not take colour changes into account as it only regards greyscale intensity changes.

(a) Seed strokes (b) Segmentation result

Figure 5.14: Blue facade segmentation



(a) Seed strokes (b) Segmentation result

Figure 5.15: Reflecting window facade segmentation

In figure 5.15 a facade is shown, which is really difficult to segment. Especially the rolling shutter, the alternating windows and wall, as well as the hedge in front of the building make problems, because these regions are varying very much. They consist primarily of patterns and not of a constant colour. Hence texture-based image segmentation might help here.

Chapter 6

Closing Words

A new method for estimating depth from shading has been presented. For simplifying image acquisition histogram matching was introduced. Due to the problems of depth hallucination with specular and reflecting materials, it was shown how image segmentation algorithms can be used to semi-automatically select these regions. The strengths and weaknesses of depth hallucination and the other algorithms with regard to depth hallucination were evaluated.
All in all it has been shown, that the depth hallucination algorithm can be used perfectly to reconstruct depth-maps of floors and walls. On facades the algorithm has certain problems. Only the problem with reflecting and specular materials could be solved by using image segmentation. The depth in these regions could then be set to a constant value. For the other problems, no really well working solution could be found. Some experiments were made to improve the depth-map quality of facades, but they yield only small improvements. Due to lack of time, a more complex experiment using a reflection pyramid during image acquisition to measure the amount of incident light from each direction, could not be conducted. This could be done in future.

Currently, the histogram matching algorithm works channel-wise. Hence the greyscale histogram of each channel is matched separately. This may result in incorrect colours, because humans are sensitive to chromatic changes. Unfortunately expanding the histogram matching algorithm from one dimensional greyscale images to three dimensional colour images is not straightforward. Expanding from a greyscale histogram to a joint colour histogram (usually red, green and blue), as well as taking a human's way of colour perception into account makes coloured histogram matching not as easy as it might look at a first glance [WSM99].

As the interaction term is primarily responsible for the segmentation quality, the graph-cut algorithm could be improved by implementing better fitting energy functions adapted individually to the given image data. For example the *laplacian zero crossing* filter for edge detection [MH80], proposed by Mortensen and Barret [MB95] could yield better results than the currently used terms. Their term combines three energy terms, taking

different image information into account.

Besides improving the presented algorithms, there are many ways in which the tool can be improved as well. The following ideas came into my mind during its development:

Large or Seamless Textures (Texture Synthesis) When rendering surfaces with reconstructed depth-maps of for example floors or walls, the texture wrapping mode is mostly set to *repeating*. Consequently, it is important to have either large or tileable texture- and depth-maps to obtain a seamless rendering of the surface. Of course, photographs are hardly ever tileable. Thus a feature in the tool for generating seamless or large texture- and depth-maps would be worthwhile. For generating large images based on smaller ones *texture synthesis* techniques like the one presented in [LLX+01] could be used. For creating seamless textures, one could attempt to scale the input image by $\frac{1}{4}$ and mirror it 4 times. Then Gaussian Laplacian pyramids could be used to blend between the images 1 and 2 and between the images 3 and 4. The two resulting images are then blended again, yielding a seamless image [OABB85].

Normal-Map and Cone-Map Generation In the city viewer of the Virtual Aachen Project, there are next to depth-maps also normal-maps and cone-maps[1] required for rendering. Generating all these needed textures would simplify the content generation process by concentrating all work into just one single tool.

Automatic Rectification When capturing photographs of facades, in the majority of cases it is impossible to orientate the camera perpendicular to the facade, because too much of the street and pavement would be visible. Such pictures are perspectively distorted and the degree of distortion depends on the angle the camera orientation differs from being perpendicular to the facade. Hence a feature in the tool for automatic rectification[2] of images would be desirable. An automatic rectifier could be implemented, by first finding lines in an image. Then vanishing points of the lines are computed and a transformation mapping is calculated. For finding lines and vanishing points *hough transformation* [Hou62] could be used as described in [Rot00].

Self-Shadowing/Occlusion In order to further increase the realism of the relief mapping visualisation mode *self-shadowing* and *self-occlusion* could be implemented [PNC05].

[1] Ray-tracing the depth-map in relief mapping leads to artifacts in the rendered image due to the linear search step for finding the first intersection. A cone-map stores *cone ratios*, which advises the linear search how far it can go further on the ray in each step [Dum].

[2] *Rectification* means the elimination of geometric (e. g. perspective) distortions in images.

Bibliography

[BA83] Peter J. Burt and Edward H. Adelson. The Laplacian Pyramid as a Compact Image Code. *IEEE Transactions on Communications*, COM-31, 4:532–540, 1983.

[BB07] Wilhelm Burger and Mark J. Burge. *Digital Image Processing: An Algorithmic Introduction using Java*. Springer, November 2007.

[BFL06] Yuri Boykov and Gareth Funka-Lea. Graph Cuts and Efficient N-D Image Segmentation. *Int. J. Comput. Vision*, 70(2):109–131, November 2006.

[BK01] Yuri Boykov and Vladimir Kolmogorov. An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26:359–374, 2001.

[Bli78] James F. Blinn. Simulation of Wrinkled Surfaces. In *Computer Graphics (Proceedings of SIGGRAPH 78)*, pages 286–292, August 1978.

[BN76] James F. Blinn and Martin E. Newell. Texture and Reflection in Computer Generated Images. *Commun. ACM*, 19(10):542–547, October 1976.

[Coo84] Robert L. Cook. Shade Trees. In Hank Christiansen, editor, *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 223–231, July 1984.

[Dan80] P. E. Danielsson. Euclidean Distance Mapping. *Computer Graphics and Image Processing*, 14:227–248, 1980.

[DTM96] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image-Based Approach. Technical Report UCB/CSD-96-893, EECS Department, University of California, Berkeley, Jan. 1996.

[Dum] Jonathan Dummer. Cone step mapping: An iterative ray-heightfield intersection algorithm. http://www.lonesock.net/files/ConeStepMapping.pdf. [Online, accessed 20. July 2009].

[FF62] L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[FH04] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient Graph-Based Image Segmentation. *Int. J. Comput. Vision*, 59(2):167–181, September 2004.

[GPS89] D. M. Greig, B. T. Porteous, and A. H. Seheult. Exact Maximum A Posteriori Estimation for Binary Images. *JRSS*, 51(2):271–279, 1989.

[GT88] Andrew V. Goldberg and Robert E. Tarjan. A New Approach to the Maximum Flow Problem. *Journal of the ACM*, 35:921–940, 1988.

[GWM+08] Mashhuda Glencross, Gregory J. Ward, Francho Melendez, Caroline Jay, Jun Liu, and Roger Hubbold. A Perceptually Validated Model for Surface Depth Hallucination. In *SIGGRAPH '08: ACM SIGGRAPH 2008 Papers*, pages 1–8, New York, NY, USA, 2008. ACM.

[GWW99] Y. Gdalyahu, D. Weinshall, and M. Werman. Stochastic Image Segmentation by Typical Cuts. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 2, page 601 Vol. 2, 1999.

[HB95] David J. Heeger and James R. Bergen. Pyramid-based texture analysis/synthesis. In *SIGGRAPH '95: Proceedings of the 22nd annual Conference on Computer Graphics and Interactive Techniques*, pages 229–238, New York, NY, USA, 1995. ACM Press.

[Her] Wil Hershberger. Taming those Annoying Highlights: Cross-Polarization Flash Macro Photography. http://www.naturescapes.net/042004/wh0404.htm. [Online, accessed 15. June 2009].

[Hou62] P.V.C. Hough. Method and Means for Recognising Complex Patterns. In *US Patent 3069654*, 1962.

[Jäh05] Bernd Jähne. *Digitale Bildverarbeitung*. Springer, 6. edition, April 2005.

[KKH+93] Charles Kervrann, Charles Kervrann, Fabrice Heitz, Fabrice Heitz, and Projet Temis. A Markov Random Field Model-Based Approach to unsupervised Texture Segmentation Using Local and Global Spatial Statistics, 1993.

[KRFB05] Erum Khan, Erik Reinhard, Roland Fleming, and Heinrich Bülthoff. Image-Based Material editing. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Sketches*, page 148, New York, NY, USA, 2005. ACM.

[KTI+01] Tomomichi Kaneko, Toshiyuki Takahei, Masahiko Inami, Naoki Kawakami, Yasuyuki Yanagida, Taro Maeda, and Susumu Tachi. Detailed Shape Representation with Parallax Mapping. In *In Proceedings of the ICAT 2001*, pages 205–208, 2001.

[LB00] M.S. Langer and H.H. Bülthoff. Depth Discrimination from Shading under Diffuse Lighting. *Perception*, 29:649–660, 2000.

[LFTW05] Hongsong Li, Sing Choong Foo, Kenneth E. Torrance, and Stephen H. Westin. Automated Three-Axis Gonioreflectometer for Computer Graphics Applications. In *Optical Engineering*, page 2006, 2005.

[LLX+01] Lin Liang, Ce Liu, Ying-Qing Xu, Baining Guo, and Heung-Yeung Shum. Real-Time Texture Synthesis by Patch-Based Sampling. *ACM Trans. Graph.*, 20(3):127–150, 2001.

[LVJ07] Y. Liu, O. Veksler, and O. Juan. Simulating Classic Mosaics with Graph Cuts. In *EMMCVPR07*, pages 55–70, 2007.

[LZ94] M. S. Langer and S. W. Zucker. Shape from Shading on a Cloudy Day. *J. Opt. Soc. Am.*, 11(2):467–478, 1994.

[MB95] Eric N. Mortensen and William A. Barrett. Intelligent Scissors for Image Composition. In *SIGGRAPH '95: Proceedings of the 22nd annual Conference on Computer Graphics and Interactive Techniques*, pages 191–198, New York, NY, USA, 1995. ACM.

[McC05] Jeffrey J. McConnell. *Computer Graphics: Theory Into Practice*. Jones and Bartlett Publishers, Inc., USA, 2005.

[MH80] D. Marr and E. Hildreth. Theory of Edge Detection. *Proceedings of the Royal Society of London. Series B, Biological Sciences*, 207(1167):187–217, 1980.

[N.79] Otsu N. A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man and Cybernetics*, 9(1):62–66, January 1979.

[ND06] Addy Ngan and Frédo Durand. Statistical Acquisition of Texture Appearance. In *Rendering Techniques 2006: 17th Eurographics Workshop on Rendering*, pages 31–40, Jun. 2006.

[NFH07] Alfred Nischwitz, Max Fischer, and Peter Haberäcker. *Computergrafik und Bildverarbeitung: Alles für Studium und Praxis*. Vieweg und Teubner, 2. edition, Feb. 2007.

[OABB85] J. M. Ogden, E. H. Adelson, J R. Bergen, and P.J. Burt. Pyramid-Based Computer Graphics, 1985.

[OBM00] Manuel M. Oliveira, Gary Bishop, and David F. McAllister. Relief Texture Mapping. In *SIGGRAPH*, pages 359–368, 2000.

[OCS05] Y. Ostrovsky, P. Cavanagh, and P. Sinha. Perceiving Illumination Inconsistencies in Scenes. *Perception*, 34(11):1301–1314, 2005.

[PCF05] J. A. Paterson, D. Claus, and A. W. Fitzgibbon. BRDF and Geometry Capture from Extended Inhomogeneous Samples using Flash Photography. *Computer Graphics Forum (Special Eurographics Issue)*, 24(3):383–391, Sep. 2005.

[PNC05] Fabio Policarpo, Manuel M. O. Neto, and Joao L. D. Comba. Real-Time Relief Mapping on Arbitrary Polygonal Surfaces. *ACM Transaction on Graphics*, 24-3(0730301):935–935, 2005.

[Pot52] R. B. Potts. Some Generalized Order-Disorder Transformations. *Proc. Camb. Phil. Soc.*, 48:106–109, 1952.

[PzG05] Mahinda P. Pathegama and Özdemir Göl. Edge-End Pixel Extraction for Edge-Based Image Segmentation. *Transactions on Engineering*, 2(2):213–216, 2005.

[Rot00] Carsten Rother. A new Approach for Vanishing Point Detection in Architectural Environments. In *In Proc. 11th British Machine Vision Conference*, pages 382–391, 2000.

[RP66] A. Rosenfeld and J. L. Pfaltz. Sequential Operations in Digital Picture Processing. *Journal of the ACM*, 13(4):471–494, 1966.

[SM97] Jianbo Shi and Jitendra Malik. Normalized Cuts and Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:888–905, 1997.

[SS01] Linda G. Shapiro and George C Stockman. *Computer Vision*. Prentice Hall, January 2001.

[Wel04] Terry Welsh. Parallax Mapping with Offset Limiting: A PerPixel Approximation of Uneven Surfaces. Technical report, Infiscape Corp., January 2004.

[WF06] Slawo Wesolkowski and Paul Fieguth. Hierarchical Region Mean-Based Image Segmentation. *Computer and Robot Vision, Canadian Conference*, 0:30, 2006.

[WHMM06] Lior Wolf, Xiaolei Huang, Ian Martin, and Dimitris Metaxas. Patch-Based Texture Edges and Segmentation. In *In ECCV*, 2006.

[WSM99] Arthur R. Weeks, Lloyd J. Sartor, and Harley R. Myler. Histogram Specification of 24-bit Color Images in the Color Difference (C-Y) Color Space. *Journal of Electronic Imaging*, 8(3):290–300, 1999.

[YDMH99] Yizhou Yu, Paul Debevec, Jitendra Malik, and Tim Hawkins. Inverse Global Illumination: Recovering Reflectance Models of Real Scenes from Photographs. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 215–224. ACM Press/Addison-Wesley Publishing Co., 1999.

[Zha06] Y.J. Zhang. *Advances in Image and Video Segmentation*. IRM Press, May 2006.

[ZLK06] Yuanjie Zheng, Stephen Lin, and Sing Bing Kang. Single-Image Vignetting Correction. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 1:461–468, 2006.

[ZTCS99] Ruo Zhang, Ping-Sing Tsai, James E. Cryer, and Mubarak Shah. Shape from Shading: A Survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, 21(8):690–706, August 1999.