Intensity based Three-Dimensional Reconstruction with Nonlinear Optimization

Christian Hopfgartner[1], Ingo Scholz[2], Martin Gugat[1], Günter Leugering[1], Joachim Hornegger[2]
Friedrich-Alexander Universität Erlangen Nürnberg
[1]: Lehrstuhl 2 für Angewandte Mathematik, Martensstr. 3, 91058 Erlangen, Germany
[2]: Lehrstuhl 5 für Informatik, Martensstr. 3, 91058 Erlangen, Germany

***Abstract*—New images of a three-dimensional scene can be generated from known image sequences using lightfields. To get high quality images, it is important to have accurate information about the structure of the scene. In order to optimize this information, we define a residual-function. This function represents the difference between an image, rendered in a known view from neighboured images and the original image at the same position. In order to get optimal results, we minimize the residual-function by defining a nonlinear least-squares problem, which is solved by an appropriate optimization method. We use a nonmonotone variant of the Levenberg-Marquardt method.**

***Index Terms*—Nonlinear Optimization, Imaging, Rendering, Nonmonotone Levenberg-Marquardt.**

Mathematics Subject Classification: 90C30, 68U10, 94A08

I. Introduction

It is an important problem to obtain images of an object or a scene in positions where no images are available. It would be advantageous, if one could generate this desired image from already given neighboured images.

One important point in order to get high quality images is to have accurate information about the geometry of the object or the scene. When we generate new images using a scene geometry which represents the real geometry well, we get good images, too. Otherwise we get images with more blurring, distortions and superpositions. This also implies that the quality of the rendered image suggests the quality of the scene geometry.

But how can we decide wether the quality of an image is good or bad? For this, we render an image from its neighbours in a position, where already an original image of the scene exists. Then we compare both images. If the difference between the two images is small, also the rendered image and the original image look very similar, the quality of the rendered image is good and so the assumed scene geometry, too. Otherwise, if the difference is large, the assumed scene geometry is bad and must be improved.

We compare the images pixel-wise. Between each pixel of the original image and the pixel at the same position in the rendered image we compute the absolute mean difference per color-channel.

The scene geometry itself is represented by a triangle-mesh. Our objective is to reconstruct this triangle-mesh in such a way that there are no differences between the rendered and the original image. This occurs, if the result of the residual-function is a vector of zeros. Because we have an overdetermined system of equations, in general there is no exact unique solution. So we want to find the solution which is as close as possible to zero. This leads to a nonlinear least-squares problem, where we minimize the sum of the squares of the residual-function result in order to optimize the whole scene geometry. We solve this problem with the Levenberg-Marquardt method where we use a nonmonotone extension to get better results.

In Section 2 we define our objective function. In Section 3 we explain the optimization with the Levenberg-Marquardt method and the non-monotone version. Section 4 contains remarks about the implementation. In Section 5 we show some numerical results and images.

II. The objective function

A. Scene geometry and images

The scene geometry is represented by a triangle mesh of V vertices v_i $(1 \leq i \leq V)$. These vertices can be shifted in each direction in space, so the geometry will be determind by $3V$ parameters

$$x = (v_1^x, v_1^y, v_1^z, \ldots, v_V^x, v_V^y, v_V^z). \tag{1}$$

In the experiments we use an 8×8 triangle mesh so we have 192 parameter.

The used color model for the images is the RGB-color model, so we have three channels per pixel. Each image has a resolution of $w \times h$ pixels. In our experiments, we use a resolution of 256×256 and 512×512.

B. Image rendering

To generate a new image from its neighboured images we use an appropiate renderer from the lgf3-library [10]. The data which are needed for rendering are the neighboured images with camera parameters and the triangle mesh which represents the scene geometry certainly.

Here we define the rendering as the mapping

$$\begin{aligned} f : \mathbb{R}^{3V} &\rightarrow \mathbb{R}^{3wh} && (2)\\ x &\mapsto f(x) =: y && (3) \end{aligned}$$

where x is the above defined parameter vector, which determines the scene geometry. The vector y as result of f contains the rendered image in the form

$$y = (y_{1,1}, \ldots, y_{1,w}, \ldots, y_{h,1}, \ldots, y_{h,w})^\top \tag{4}$$

where $y_{i,j}$ denotes the color of the pixel in the i-th row and j-th column and is defined as

$$y_{i,j} = (y^R_{i,j}, y^G_{i,j}, y^B_{i,j})^\top. \tag{5}$$

Each of the three components of $y_{i,j}$ can be integer values between 0 and 255. So each pixel can receive 256^3 different colors.

Since it is not necessary for the understanding of the context, we do not show here the exact definition of f. For more information about lightfield rendering see [2], [4], [8] for example.

C. Image comparison

We compare the rendered image with the original image. For this purpose we compute the absolute mean difference per color channel for every pixel using the mapping

$$\begin{aligned} g : \mathbb{R}^{3wh} &\rightarrow \mathbb{R}^{wh} && (6)\\ y &\mapsto g(y) && (7)\end{aligned}$$

with

$$g(y) = \begin{pmatrix} g_1(y^R_1, y^G_1, y^B_1) \\ \vdots \\ g_{wh}(y^R_{wh}, y^G_{wh}, y^B_{wh}) \end{pmatrix} \tag{8}$$

and

$$\begin{aligned} g_i(y_i) = \frac{1}{3}\Big(|y^R_i - \hat{y}^R_i| + |y^G_i - \hat{y}^G_i| + |y^B_i - \hat{y}^B_i|\Big), && (9)\\ \forall i = 1, \ldots, wh. && (10)\end{aligned}$$

The original image is stored in the vector $\hat{y}$ and the rendered image in the vector y.

The smaller the difference between the channels of y_i and $\hat{y}_i$ is, the smaller becomes the value of $g_i(y_i)$. If the difference of all channels is equal to zero, $g_i(y_i) = 0$, too.

We note, that the range of g_i consists of 766 possible elements $\{0, \frac{1}{3}, \frac{2}{3}, 1, \ldots, 255\}$.

D. The residual function

We get the residual function as a composition of image rendering and image comparison

$$\begin{aligned} R : \mathbb{R}^{3V} &\rightarrow \mathbb{R}^{wh} && (11)\\ x &\mapsto R(x) := (g \circ f)(x). && (12)\end{aligned}$$

In order to find the optimal scene geometry x^*, we have to solve the overdetermined system

$$R(x) = 0 \tag{13}$$

which has less degrees of freedom than equations ($3V < wh$) and so there need not exist any exact solution.

E. The objective function

We have to find a solution x^* for which the value $\|R(x)\|$ becomes minimal. Here we denote with $\|\cdot\|$ the Euclidean norm. For this, we define our objective function

$$\begin{aligned} F : \mathbb{R}^{3V} &\rightarrow \mathbb{R} && (14)\\ x &\mapsto F(x) := \frac{1}{2}\|R(x)\|^2 && (15)\end{aligned}$$

and want to determine a vector x^*, which minimizes F. This leads to the nonlinear optimization problem without constraints

$$\min_{x \in \mathbb{R}^{3V}} F(x), \quad F(x) = \frac{1}{2}\|R(x)\|^2 \tag{16}$$

This is a nonlinear least-squares problem.

F. Derivatives of the objective function

To solve (16) we need informations about the derivatives of first and second order of F. The first derivative is

$$F'(x)^\top = J(x)^\top R(x) \in \mathbb{R}^{3V} \tag{17}$$

where $J(x)$ is the $wh \times 3V$ Jacobian matrix of $R(x)$. The second derivative is

$$F''(x) = J(x)^\top J(x) + B(x) \in \mathbb{R}^{3V \times 3V} \tag{18}$$

where $B(x)$ contains derivatives of second order of $R(x)$. To compute all of these is very expensive. Since $\|B(x)\| \rightarrow 0$ if $\|R(x)\| \rightarrow 0$ and this happens near an optimal solution x^*, we can omit $B(x)$ in (18).

III. Optimization

For the optimization we use the Levenberg-Marquardt method [7], [9]. This is an iterative method, whose search direction interpolates between the direction of the gradient-method and the Gauss-Newton method. So it combines the positive features of both methods. First, it can ensure the convergence far away from a minimizer point, second it has fast local convergence near a minimizer point. In the Levenberg-Marquardt method, both the search direction and the step length will be computed simultaneously. To compute one step d_k, we solve the system of equations

$$(J_k^T J_k + \lambda I) d_k = -J_k^T R_k \tag{19}$$

based on the second order Taylor approximation of the objective function in x_k where $J_k = J(x_k)$ and $R_k = R(x_k)$. J_k denotes the Jacobian matrix of R_k and I the identity matrix. The step length and search direction of d_k depend on the choice of the Levenberg-Marquardt parameter λ. If $\lambda = 0$, the direction we get is the same as for the Gauss-Newton method. The step length is maximal. If we let $\lambda \rightarrow \infty$, the search direction is the direction of the steepest descent with a very small step length, because $\lim_{\lambda \rightarrow \infty} (J_k^T J_k + \lambda I)^{-1} \rightarrow 0$.

The choice of λ depends on the ratio of the actual reduction to the predicted reduction of the objective function. The actual reduction Δ_{ared} is defined as

$$\Delta_{ared} = F_k - F_{k+1} \tag{20}$$

The predicted reduction Δ_{pred} based on the quadratic approximation of F in the k-th iteration point x_k

$$\tilde{F}(x_k + d_k) = \tilde{F}_k(d_k) = F_k + d_k^\top J_k^\top R_k + \frac{1}{2} d_k^\top (J_k^\top J_k) d_k. \quad (21)$$

is defined as

$$\Delta_{pred} = -\frac{1}{2} d_k^\top J_k^\top R_k + \frac{1}{2}\lambda \|d_k\|^2 \quad (22)$$

The ratio $\Delta_{ared}/\Delta_{pred}$ specifies the quality of the approximation $\tilde{F}_k$. The bigger the ratio is, the better F is approximated by $\tilde{F}$ in a neighbourhood of x_k.

Algorithmus 3.1 (MLM(Monotone Levenberg Marquardt)): To solve (16), start from x_0. Given $\mu > 0$, $\nu > 1$, $\lambda > 0$, $F(x_0)$. Set $k = 0$.

1) Compute R_k, J_k and $F_k' = J_k^T R_k$. If a stop criterion is fulfilled, STOP!
2) Solve $(J_k^T J_k + \lambda I) d_k = -J_k^T R_k$ to obtain d_k.
3) Compute $F_{k+1} = F(x_k + d_k)$.
4) Compute $\Delta_{ared} = F_k - F_{k+1}$,
   $\Delta_{pred} = -(F_k' d_k + \lambda \|d_k\|^2)/2$.
5) If $\Delta_{ared}/\Delta_{pred} < \mu$, set $\lambda = \lambda * \nu$, GOTO 2.
6) If $\mu_k \leq \Delta_{ared}/\Delta_{pred}$, set $\lambda = \lambda/\nu$ and $k = k+1$, GOTO 1.

In section IV-C we give information on our stop criterion in the algorithm.

The algorithm MLM generates a monotonically decreasing sequence of objective function values.

To prevent the algorithm from being trapped early in the iteration in the bottom of a valley, Zhang and Chen [11] relax the monotonicity requirement by the introduction of a nonmonotone linesearch rule. This enhances the possibility of finding a global optimum and leads to the algorithms NMLM1 and NMLM2 defined below.

In Section V, we give an example (Example 1: Erlanger) where only the nonmonotone algorithms NMLM1 and NMLM2 obtain an acceptable solution, whereas the monotone Levenberg-Marquardt method MLM fails. Only for relatively simple scene geometries (see the Examples Perri and Claus), also MLM produces acceptable results.

Define the actual reduction as

$$\Delta_{ared} = F_{\max} - F_{k+1} \quad (23)$$

where $F_{\max}$ is defined by

$$F_{\max} = \max\{F_k, F_{k-1}, \cdots, F_{k-M}\} \quad (24)$$

and F_i does not exists for negative i. Let $M \geq 0$ be given.

Algorithmus 3.2 (NMLM1(Nonmonotone LM1)): To solve (16), start from x_0. Given $\mu > 0$, $\nu > 1$, $\lambda > 0$, $\eta > 0$, $M \geq 0$, $F(x_0)$. Set $k = 0$.

1) Compute R_k, J_k and $F_k' = J_k^T R_k$. If a stop criterion is fulfilled, STOP!
2) Solve $(J_k^T J_k + \lambda I) d_k = -J_k^T R_k$ to obtain d_k.
3) Compute $F_{k+1} = F(x_k + d_k)$.
4) Compute $\Delta_{ared} = F_{\max} - F_{k+1}$,
   $\Delta_{pred} = -(F_k' d_k + \lambda \|d_k\|^2)/2$.
5) Compute $\tilde{\mu}_k = \begin{cases} \mu, \text{if } M = 0 \\ \min\{\mu, \eta \frac{\|F_k'\|^2 \, \|d_k\|^2}{\Delta_{pred}}\}, \text{if } M > 0 \end{cases}$
6) If $\Delta_{ared}/\Delta_{pred} < \tilde{\mu}_k$, set $\lambda = \lambda * \nu$, GOTO 2.
7) If $\tilde{\mu}_k \leq \Delta_{ared}/\Delta_{pred}$, set $\lambda = \lambda/\nu$ and $k = k+1$, GOTO 1.

For a further relaxation, Δ_{pred} is set to

$$\Delta_{pred} = -\frac{1}{2} d_k^\top J_k^\top R_k \quad (25)$$

which is based upon the quadratic Levenberg-Marquardt model of F in x_k

$$\hat{F}(x_k + d_k) = F_k + d_k^\top J_k^\top R_k + \frac{1}{2} d_k^\top (J_k^\top J_k + \lambda I) d_k. \quad (26)$$

Algorithmus 3.3 (NMLM2(Nonmonotone LM2)): To solve (16), start from x_0. Given $\mu > 0$, $\nu > 1$, $\lambda > 0$, $\lambda_{\min} > 0$, $\eta > 0$, $M \geq 0$, $F(x_0)$. Set $k = 0$.

1) Compute R_k, J_k and $F_k' = J_k^T R_k$. If a stop criterion is fulfilled, STOP!
2) Solve $(J_k^T J_k + \lambda I) d_k = -J_k^T R_k$ to obtain d_k.
3) Compute $F_{k+1} = F(x_k + d_k)$.
4) Compute $\Delta_{ared} = F_{\max} - F_{k+1}$, $\Delta_{pred} = -F_k' d_k/2$.
5) Compute $\hat{\mu}_k$
   $\hat{\mu}_k = \begin{cases} \mu, \text{if } M = 0 \\ \min\{\mu, \eta \frac{\|F_k'\|^2 \, \|d_k\|^2}{\Delta_{pred}} \|J_k^T J_k + \lambda I\|_\infty\}, \text{if } M > 0 \end{cases}$
6) If $\Delta_{ared}/\Delta_{pred} < \hat{\mu}_k$, set $\lambda = \lambda * \nu$, GOTO 2.
7) If $\hat{\mu}_k \leq \Delta_{ared}/\Delta_{pred}$, set $\lambda = \max(\lambda/\nu, \lambda_{\min})$ and $k = k+1$, GOTO 1.

IV. Implementation of the optimization

A. Computation of the Jacobian matrix

We compute the Jacobian matrix J of the residual function numerically because we do not have any information about the analytical derivatives. For this we favour the use of central differences over the use of forward differences. Indeed computing central differences is twice as expensive. But we get a smaller truncation error and so the iteration points will be computed more accurately.

The adaption of the step length is important in the computation of the Jacobian matrix. Sufficiently small changes in the vector x do not result in changes in the rendered image, because an image consists of finite colors. Expressed in more mathematical terms, f maps to discrete values.

So, if we compute all columns J_j $(1 \leq j \leq 3V)$ of J with a small change δ, given by the machine precision, such that $f(x + \delta e_j) = f(x) = f(x - \delta e_j)$, where e_j denotes the j-th unity vector, we have

$$R(x + \delta e_j) = R(x - \delta e_j) \quad (27)$$

and therefore J contains only zeros.

The optimal step length $\delta = 0.01$ used in our experiments was estimated experimentally.

B. Efficient storage of the Jacobian matrix

In our experiments the percentage of elements of the Jacobian matrix equal to zero was about 99 percent. So, we save it as a sparse matrix. This has two advantages:

To begin with the acceleration of the matrix-matrix multiplication. In each iteration we multiply the transposed Jacobian matrix of R with the Jacobian matrix itself. In our experiments, the Jacobian matrix has a dimension of $262\,144 \times 192$. Saved as a full matrix, the time to multiply them is about 300 seconds, which is not an acceptable time, seeing that we compute up to 100 iterations. Saved as a column oriented sparse matrix, one multiplication takes about 0.5 seconds.

Moreover, we save a lot of memory. Each entry of the Jacobian matrix is represented by a `double` (8 byte). Saved as a full matrix, it needs about 384MB, saved as a column oriented sparse matrix only 8MB. Note that more refinement of the mesh or a higher resolution of the images yields a Jacobian matrix, whose size exceeds the main memory rapidly, if we would save it as a full matrix.

C. *Choice of the stop criterion*

The most widely used stop criterion $\|F'(x_k)\| < \epsilon$ where $\epsilon > 0$ is a small number cannot be used for our problem. The reason is the finite difference approximation of $\|F'(x)\|$, which can only attain a finite number of values.

We let the algorithm stop if the Levenberg-Marquardt parameter λ exceeds the threshold $\lambda_{\max} = 10^{14}$. Here the computed length of d_k is so small that there are would be no difference between F_{k+1} and F_k because of the discrete state of the residual function.

V. Results

In the following we show three examples of lightfields, where we reconstruct the scene geometry.

We perform the numerical experiments on an Intel Pentium 4 CPU 3GHz with 2GB RAM and a nVidia GeForce MX440SE graphics card.

In the algorithm we choose the following parameter

$$\mu = 0.55,\ \nu = 2,\ \eta = 10^{-3},\ \lambda_{\min} = 1.0,\ \lambda_{\max} = 10^{14}. \quad (28)$$

For the numerical computation of the Jacobian matrix we set $\delta = 0.01$. As initial value we set the Levenberg-Marquardt parameter $\lambda = 1.0$. We represent the scene geometry with a regular 8×8 triangle mesh. As initial data for the optimization we choose a flat mesh parallel to the original image.

A. *Example 1: Erlanger*

In Fig. 1 we show two rendered images (1c) and (1d) as results after the optimization of the scene. All images have a resolution of 512×512 pixels.

We generate a new image in the same position and with the same camera parameters as the original image (1b) using 5 source images. Fig. (1a) shows the rendered image before the optimization of the scene geometry. We see the blurring and superpositions clearly, which arise due to the wrong scene geometry. Fig. (1d) was generated using the optimized scene geometry after applying of a monotone algorithm. For the generation of image (1c) we use the scene geometry which is the result of the optimization with the nonmonotone algorithm NMLM2 with $M = 4$.

We see that after application of the monotone algorithm some superpositions do not yet disappear entirely. Nevertheless it is a good result. We got still better results with the nonmonotone algorithm. In the rendered image, no more blurring is recognizable.

The images (2a) and (2b) in Fig. 2 were generated in the same position as the images in Fig. 1, but this time using 10 source images. Due to the wrong scene geometry in Fig. (2a) this results in still more superpositions and blurring than in (1a). Here the nonmonotone Levenberg-Marquardt algorithm NMLM2 with $M = 8$ yields very good results (see (2b)), too. The images (2c) and (2d) show the differences between the original image (1a) and the rendered images (2a) and (2b) respectively .

Fig. 3 shows the optimized scene geometry from two different vantage points. The third image shows the optimized triangle mesh without texture.

B. *Example 2: Perri*

In Fig. 4 the first two images show the rendered images before and after the optimization, respectively. We used 10 source images to generate them. All images have a resoulution of 512×512 pixels. The last two images of Fig. 4 show the difference between the original image and the rendered images again. In this lightfield there were hardly any differences between the results of the nonmonotone and the monotone algorithms. All of them give good results. The application of the nonmonotone algorithms leads to an objective function's value that is smaller.

Fig. 5 shows the optimized scene geometry from two different vantage points. The third image shows the optimized triangle mesh without texture.

C. *Example 3: Claus*

In Fig. 6 the first two images also show the rendered images before and after the optimization, respectively. We used 7 source images to generate them. All images have a smaller resolution of 256×256 pixels. The last two images of Fig. 6 show the difference between the original image and the rendered images, too. Indeed, the differences between the results of the nonmonotone and the monotone algorithms are not as small as for the Perri lightfield. Nevertheless, the results of the algorithms differ hardly. The objective function's value after the optimization with the nonmonotone algorithm is about 10 percent smaller than the value after the optimization with a monotone algorithm.

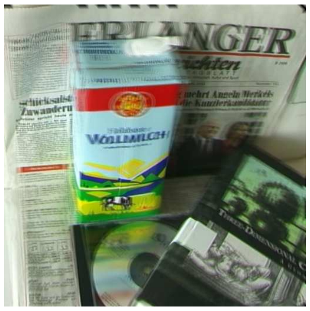
Fig. 7 shows the optimized scene geometry from two different vantage points. The third image shows the optimized triangle mesh without texture.

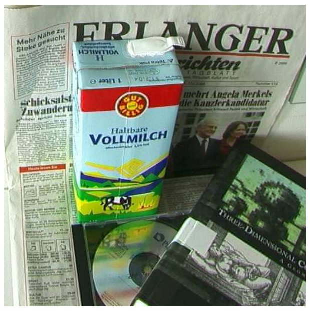
References

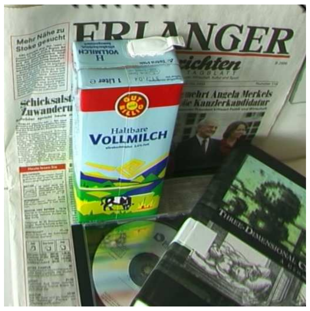
[1] Å. Björck. *Numerical Methods for Least Squares Problems.* SIAM, Philadelphia, 1996.

[2] C. Buehler, M. Bosse, L. McMillan, S. Gortler, and M. Cohen. Unstructured lumigraph rendering. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 425–432, New York, NY, USA, 2001. ACM Press.

[3] Y. H. Dai. On the nonmonotone line search. *J. Optim. Theory Appl.*, 112(2):315–330, 2002.


(a) rendered image before optimization


(b) original image


(c) rendered image after optimization with nonmonotone algorithm

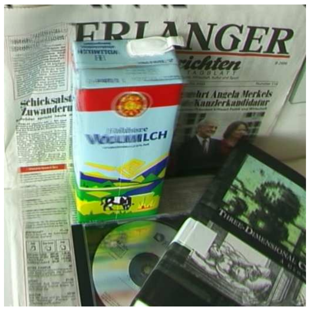
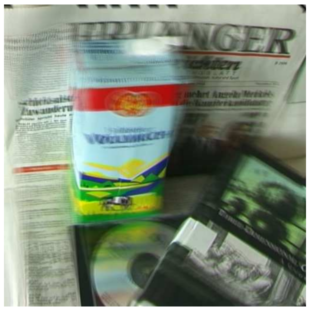
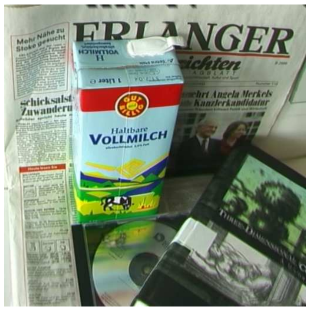
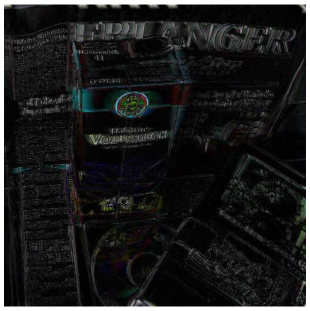

(d) rendered image after optimization with a monotone algorithm

Fig. 1.


(a) rendered image before optimization


(b) rendered image after optimization with nonmonotone algorithm


(c) difference image before optimization

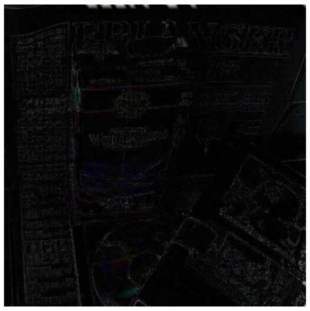
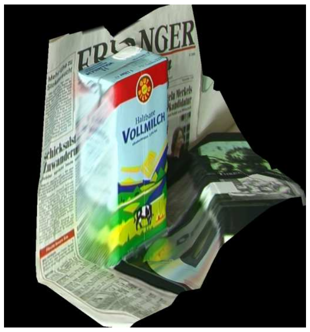
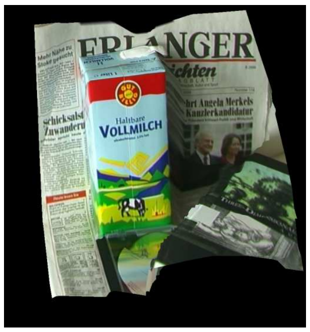

(d) difference image after optimization with nonmonotone algorithm

Fig. 2.


(a) view from online-renderer testULGHW-Renderer


(b) view from online-renderer testULGHW-Renderer

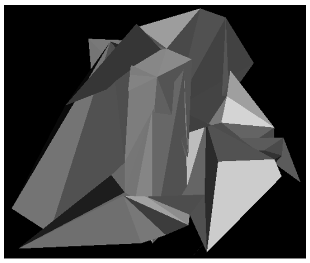
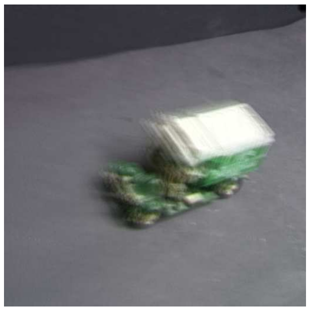
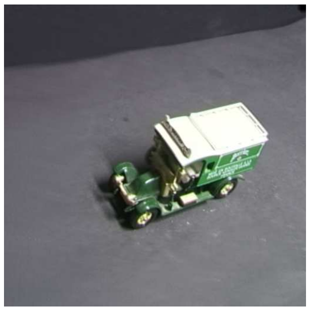
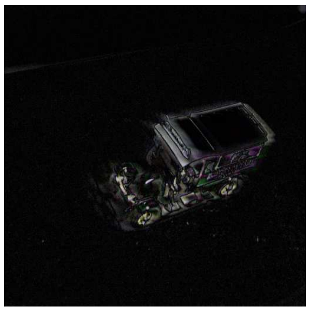

(c) optimized proxy. View from offview

Fig. 3.


(a) rendered image before optimization


(b) rendered image after optimization with a monotone algorithm


(c) difference image before optimization

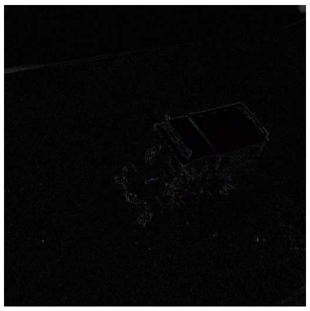
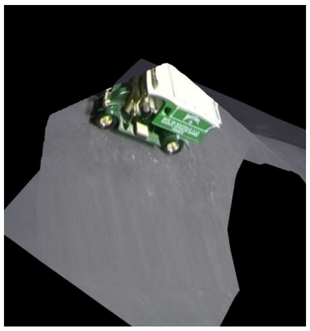
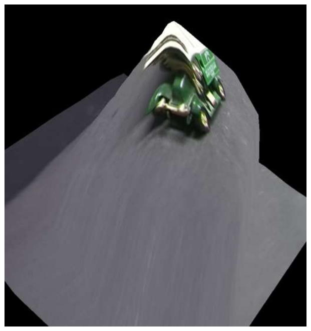

(d) difference image after optimization with a monotone algorithm

Fig. 4. Perri


(a) view from online-renderer testULGHW-Renderer


(b) view from online-renderer testULGHW-Renderer

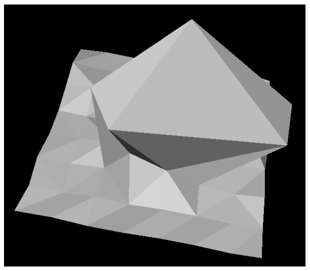
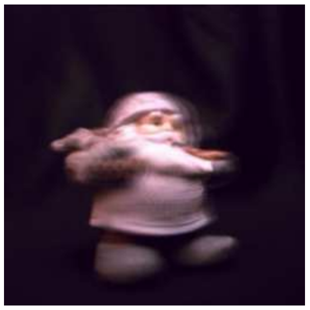
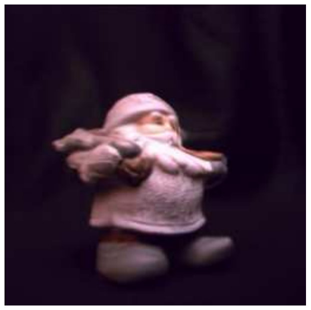
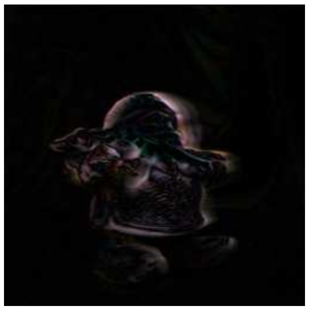

(c) optimized proxy. View from offview

Fig. 5.


(a) rendered image before optimization


(b) rendered image after optimization with a monotone algorithm


(c) difference image before optimization

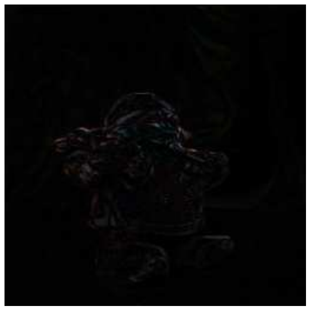
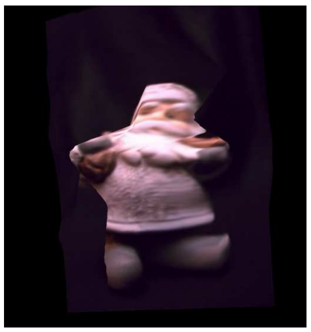
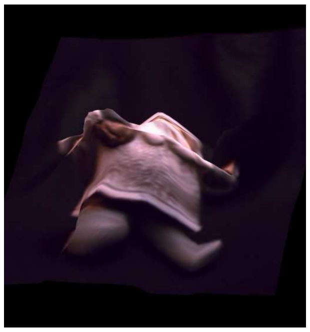

(d) difference image after optimization with a monotone algorithm

Fig. 6. Claus


(a) view from online-renderer testULGHW-Renderer


(b) view from online-renderer testULGHW-Renderer

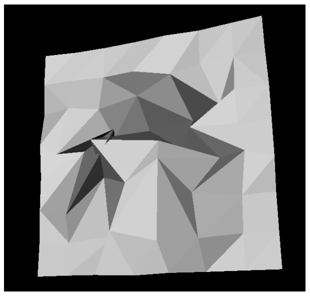

(c) optimized proxy. View from offview

Fig. 7.

[4] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The lumigraph. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 43–54, New York, NY, USA, 1996. ACM Press.

[5] C. Hopfgartner. Tiefenoptimierung in Lichtfeldern mit dem Levenberg-Marquardt Verfahren. Master's thesis, Dept. Applied Mathematics, Friedrich-Alexander-Univ., Erlangen-Nuremberg, 2006.

[6] C. T. Kelley. *Iterative Methods for Optimization*. Frontiers in Applied Mathematics. Siam, Philadelphia, 1999.

[7] K. Levenberg. A method for the solution of certain non-linear problems in least squares. *Quart. Appl. Math.*, 2:164–168, 1944.

[8] M. Levoy and P. Hanrahan. Light field rendering. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 31–42, New York, NY, USA, 1996. ACM Press.

[9] D. W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *SIAM Journal on Applied Mathematics*, 11(2):431–441, 1963.

[10] C. Vogelgsang. *The lgf3 Project: A Versatile Implementation Framework for Image-Based Modeling and Rendering*. PhD thesis, Dept. Computer Sciences, Friedrich-Alexander-Univ., Erlangen-Nuremberg, 2005.

[11] J. Z. Zhang and L. H. Chen. Nonmonotone levenberg-marquardt algorithms and their convergence analysis. *J. Optim. Theory Appl.*, 92(2):393–418, 1997.