PARALLEL NON-LOCAL DENOISING OF DEPTH MAPS

Timo Schairer, Benjamin Huhle, Philipp Jenke, Wolfgang Straßer

University of Tübingen, WSI/GRIS
Sand 14, 72076 Tübingen, Germany
{schairer, huhle, jenke, strasser}@gris.uni-tuebingen.de

ABSTRACT

We give a brief discussion of denoising algorithms for depth data and review our unified approach towards non-local denoising. This algorithm removes outliers from depth data and accordingly achieves an unbiased smoothing result. Taking into consideration the intra-patch similarity and optional color information, strong discontinuities can be handled and fine detail structure in the data can be preserved. To account for the high computational complexity of the algorithm, we examine a parallel formulation, present different data and task decomposition strategies and investigate their applicability towards a GPU implementation. We demonstrate that this approach scales very well and that a significant speed-up can be achieved.

1. INTRODUCTION

In recent years, cameras that measure real world distances based on the time-of-flight (ToF) principle are becoming increasingly popular. A wide range of applications in scene acquisition, virtual reality and entertainment in general rely on data captured with such devices. Cameras that use an active modulated light source and sensors produced in standard CMOS technology (*e.g.*, from PMDTec, Canesta or Mesa Imaging) are prospective low-cost sensors which could be deployed in standard applications, *i.e.*, mass markets. Depth data from different scanning devices may show very different noise characteristics. A common approach is to classify two different origins: Every measurement given by the sensor is affected by a certain level of inaccuracy. In some cases, however, the measurements can completely fail due to the illumination situation or because of problematic properties of the sampled surfaces (e.g., strongly non-Lambertian reflectivity). These outliers need to be excluded from further processing, not only because they degrade the model quality, but since they could also lead to biased reconstructions, *e.g.*, smoothing results. Measurements performed with a ToF sensor are no exception to this observation. Therefore, appropriate postprocessing of the depth data is essential for photo-realistic models. While systematic errors can in part be corrected by means of calibration, which reduces the noise level significantly, the data remains noisy [1, 2].

There are several reasons why it is reasonable to denoise the data in sensor space instead of regarding 3D point clouds: For most sensors we have to deal with (additive) noise in viewing direction which is implicitly modeled if we consider the data as a depth map. Furthermore, the filtering is performed earlier in the pipeline and subsequent steps such as the registration of several frames can rely on the refined data. In contrast, for common surface reconstruction techniques in 3D, *e.g.*, multiple merged frames are necessary – otherwise, occlusions would impede sound solutions. In practice, we believe that a two-stage data enhancement is necessary, comprising an early data filtering in sensor space and a subsequent surface reconstruction computed on a registered model. We review our denoising algorithm that detects outliers in a first stage and yields unbiased and feature-preserving smoothing results [3]. This approach is based on an image restoration technique, namely the recently presented non-local means (NL-Means) filter [4] that makes use of self-similarity in the data. Due to the high complexity of this algorithm and since programmable graphics hardware (GPUs) and multi-core CPUs are widely available, it is useful to look for a scalable and efficient parallel formulation. We present different data and task decomposition strategies and investigate their applicability towards a GPU implementation using the CUDA framework[1].

Firstly, we give a brief overview of different smoothing techniques and review our robust non-local smoothing technique (Section 2). Then, we discuss how this smoothing filter can be used to formulate an iterative outlier detection scheme (Section 3). We look at different parallel formulations of our algorithm (Section 4) and show results using data from a system consisting of a time-of-flight and a color camera (Section 5).

2. SMOOTHING

Denoising in general is a highly ill-posed problem and demands for some means of regularization. The filtering of 2D data such as digital images has been studied intensively for several decades now and a variety of techniques have been developed. Standard linear filters such as the Gaussian are low-pass filters that smooth the data by averaging over neighboring pixels. Thereby, noise is removed but the data is blurred and features such as edges are destroyed. Therefore, low-pass filters are appropriate if a completely smooth reconstruction is desired, but espe-

[1]see the NVidia CUDA website: `http://www.nvidia.com/object/cuda_home.html`

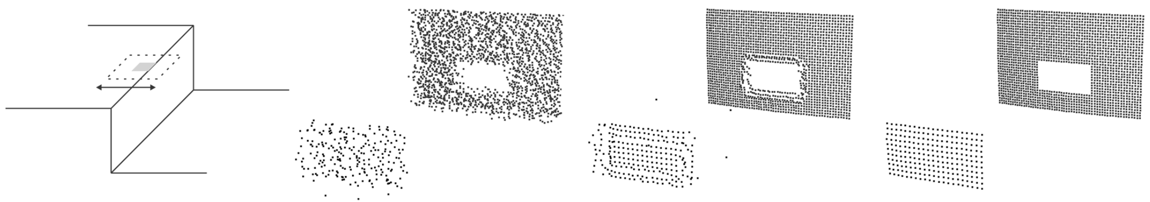


Figure 1. NL-Means Filter at a depth discontinuity: Smoothing in the depicted direction is prevented by a low similarity of the neighbored patches. Results on synthetic data (two distant surfaces at constant depth) from left to right: Input data with Gaussian noise, original NL-Means Filter, and our approach with additional weights ξ.

cially on depth data of general scenes these filters are not applicable.

Yaroslavsky [5] introduced a filter that restores a pixel by an average of its neighbors weighted by their similarity. The *bilateral* [6] or *SUSAN* [7] filter uses the same concept. However, instead of using a spatial box function as done in the Yaroslavsky filter, the neighboring pixels are continuously weighted by their spatial Euclidian distance. This results in a weight assigned to each neighboring pixel, that takes into account spatial distance as well as distance in the range domain. The weighting scheme renders the filter discontinuity-preserving, since the influence of the neighboring pixels is limited. A theoretical derivation of the bilateral filter is given in [8] and the relation to extended nonlinear diffusion methods is described in [9]. The bilateral filter is very popular in its original field but was also successfully applied to a wide variety of applications (see [10]). The underlying idea of the bilateral filter is easily extended to multi-modal data resulting in the joint- or cross-bilateral filter (e.g. [11]) where the filter weights are based on the similarity determined on a different data source or from the combination of several data sources, respectively. Kopf et al. [12] use this technique to enhance depth maps exploiting the dependencies of discontinuities in the depth and color domain. Bilateral Filtering was also applied to mesh denoising [13] as well as to 3D point clouds [14].

2.1. Non-Local Means Filter (NL-Means)

The NL-Means filter was proposed recently by Buades et al. [4] for image restoration. It follows a similar idea as the bilateral filtering approach, namely to restore a pixel by a weighted average of similar pixels

$$v'(\mathbf{i}) = \sum_{\mathbf{j}\in\mathbf{W_i}} w(\mathbf{i},\mathbf{j})v(\mathbf{j}), \tag{1}$$

where $\mathbf{W_i}$ is a potentially large search window around $\mathbf{i}$. The similarity weight w, however, is determined in a different way. Instead of comparing the single pixel values at positions $\mathbf{i}$ and $\mathbf{j}$, patches (with index set $\mathbf{N}$) surrounding both pixels are taken into account. The weight is determined as

$$w(\mathbf{i},\mathbf{j}) = \frac{1}{Z_\mathbf{i}} e^{-\frac{1}{h}\sum_{\mathbf{k}\in\mathbf{N}} G_a(\|\mathbf{k}\|_2)(v(\mathbf{i}+\mathbf{k})-v(\mathbf{j}+\mathbf{k}))^2}, \tag{2}$$

with normalization constant

$$Z_\mathbf{i} = \sum_{\mathbf{j}\in\mathbf{I}} e^{-\frac{1}{h}\sum_{\mathbf{k}\in\mathbf{N}} G_a(\|\mathbf{k}\|_2)(v(\mathbf{i}+\mathbf{k})-v(\mathbf{j}+\mathbf{k}))^2}, \tag{3}$$

and the filtering parameter h. The pixelwise distances are weighted according to their offset from the central pixel using a Gaussian kernel G_a with standard deviation a. In this way the self-similarity of the image is taken into account and even fine details that occur repeatedly can be distinguished from noise. A technique based on the same idea has also proven to be successful in the field of texture synthesis [15]. The NL-Means filter was also used for simultaneous denoising and depth reconstruction from noisy image pairs [16]. Schall et al. [17] were the first to apply NL-Means for denoising of 3D point data. They suppose that the data is given on a (not necessarily regular) 2D grid and compute the similarity weight according to the Euclidian distance in 3D. Depending on the type of noise, the point coordinates are corrected in the direction of the line-of-sight or along an estimated normal.

2.2. A variant of NL-Means for Depth Data

In earlier work, we presented a technique to remove noise in depth data based on the NL-Means algorithm [3] which we review briefly in the following.

We experienced that, on depth data, the original NL-Means algorithm as it is used in the field of image restoration produces undesirable artifacts near strong discontinuities. The filter averages pixels along edges, but in a distance less than the neighborhood radius, smoothing will occur in neither direction perpendicular to the edge because of the strong dissimilarities of the patches. We illustrate this effect in Figure 1.

In order to smooth over similar regions of depth also in the presence of a near edge, we introduce an additional term ξ_{ij} in the computation (Eq. 2) of the patch similarity

$$w(\mathbf{i},\mathbf{j}) = \frac{1}{Z_\mathbf{i}} e^{-\frac{1}{h}\sum_{\mathbf{k}\in\mathbf{N}} \xi_{\mathbf{ik}} G_a(\|\mathbf{k}\|_2)(v(\mathbf{i}+\mathbf{k})-v(\mathbf{j}+\mathbf{k}))^2}, \tag{4}$$

that results in an influence function similar to the one of the bilateral filter for pixels in other patches of the search window. The new weighting factor

$$\xi_{ik} = e^{-\frac{(v(\mathbf{i})-v(\mathbf{i}+\mathbf{k}))^2}{h}} \tag{5}$$

constrains the similarity comparison to regions of similar depths, using the same parameter h as in the computation of the inter-patch distances. This corresponds to the heuristic of searching self-similarities on the same surface. Note that the differing sampling spacing on surfaces in different depth regions prevents the patch search from detecting similar detail structure. This weighting scheme leads to an effect similar to the adoption of NL-Means to 3D point data. The authors of [17] consider Euclidian distances in 3D instead of a Gaussian weighting in pixel coordinates (Eq. 2) and therefore minimize the influence of pixels that belong to very distant regions in depth. With our approach we explicitly distinguish different depth regions in order to achieve the same smoothing effect also on distant surfaces where the sampling density is lower.

As an additional cue for the estimation of

$$p\left(v(\mathbf{i})|\{v(\mathbf{i}+\mathbf{k})\}_{\mathbf{k}\in\mathbf{N}}\right) \tag{6}$$

we consider the use of color information and compute a weight

$$w^{(u)}(\mathbf{i},\mathbf{j}) = \frac{1}{Z_{\mathbf{i}}^{(u)}} e^{-\frac{1}{h^{(u)}}\sum_{\mathbf{k}\in\mathbf{N}} \xi_{\mathbf{ik}}^{(u)} G_a(\|\mathbf{k}\|_2)\|\mathbf{u}(\mathbf{i}+\mathbf{k})-\mathbf{u}(\mathbf{j}+\mathbf{k})\|_2^2}. \tag{7}$$

corresponding to Equation 2. Here, $\mathbf{u}(\mathbf{i})$ denotes the color (we use RGB) assigned to pixel $\mathbf{i}$ and variables with superscript (u) are analogous to their counterparts in Equation 4. This yields the combined NL-Means formulation

$$v'_{uv}(\mathbf{i}) = \sum_{\mathbf{j}\in\mathbf{W}_i} w(\mathbf{i},\mathbf{j})\, w^{(u)}(\mathbf{i},\mathbf{j})\, v(\mathbf{j}), \tag{8}$$

resulting in an estimate of $v'(\mathbf{i})$ according to

$$p\left(v(\mathbf{i})|\{v(\mathbf{i}+\mathbf{k})\}_{\mathbf{k}\in\mathbf{N}}, \{\mathbf{u}(\mathbf{i}+\mathbf{k})\}_{\mathbf{k}\in\mathbf{N}}\right). \tag{9}$$

3. OUTLIER DETECTION

The NL-Means filter performs as an estimator of a pixel value given its surrounding. We described in [3] how this fact can also be employed in order to detect abnormal values of central pixels in depth patches and classify these as outliers. A short discussion of this outlier detection technique is given in the remainder of this section:

When computing the similarity of patches, we rely on pixel values (in the destination as well as in the source patch) that are potential outliers. We therefore use an iterative algorithm in the manner of expectation maximization (EM, [18]). We consider the probability distribution $p_{inlier}\left(v(\mathbf{i})|v(\mathbf{N}_{\mathbf{i}}^{*})\right)$ of a pixel $v(\mathbf{i})$ given its surrounding $v(\mathbf{N}_{\mathbf{i}}^{*})$, where $\mathbf{N}_{\mathbf{i}}^{*} := \{\mathbf{i}+\mathbf{k}\}_{\mathbf{k}\in\mathbf{N}} \setminus \{\mathbf{i}\}$. The central pixel is not taken into account in order to get unbiased similarity estimates in case pixel $\mathbf{i}$ is an outlier. The NL-Means in its original formulation (Eq. 1) computes the expectation value $E\left(v(\mathbf{i})|v(\mathbf{N}_{\mathbf{i}}^{*})\right) := \mu_{\mathbf{i}}$. Assuming that p_{inlier} follows a normal distribution, we compute the weighted variance

$$\sigma\left(v(\mathbf{i})|v(\mathbf{N}_{\mathbf{i}}^{*})\right) = \frac{1}{C_{\mathbf{i}}} \sum_{\mathbf{j}\in\mathbf{W}_{\mathbf{i}}} w^{*}(\mathbf{i},\mathbf{j})\left(v(\mathbf{j})-\mu_{\mathbf{i}}\right)^2, \tag{10}$$

normalized by

$$\frac{1}{C_{\mathbf{i}}} = \frac{\sum_{\mathbf{j}\in\mathbf{W}_{\mathbf{i}}} w^{*}(\mathbf{i},\mathbf{j})}{\left(\sum_{\mathbf{j}\in\mathbf{W}_{\mathbf{i}}} w^{*}(\mathbf{i},\mathbf{j})\right)^2 - \sum_{\mathbf{j}\in\mathbf{W}_{\mathbf{i}}} w^{*}(\mathbf{i},\mathbf{j})^2}. \tag{11}$$

To handle outliers, the weights $w^{*}(\mathbf{i},\mathbf{j})$ denote the patch similarity (Eq. 2) multiplied by the product of the validity estimation of pixel $\mathbf{i}$ and $\mathbf{j}$. These probabilities are initialized with 1.0 and updated in successive iterations according to the mixture model

$$\begin{aligned} p_{mix}\left(v(\mathbf{i})|v(\mathbf{N}_{\mathbf{i}}^{*})\right) &= \alpha\, p_{inlier}\left(v(\mathbf{i})|v(\mathbf{N}_{\mathbf{i}}^{*})\right) + \\ & p_{outlier}\left(v(\mathbf{i})|v(\mathbf{N}_{\mathbf{i}}^{*})\right), \end{aligned} \tag{12}$$

where $p_{outlier}$ follows a uniform distribution, *i.e.*, is constant and α is computed as

$$\alpha = \frac{1}{|\mathbf{W}_{\mathbf{i}}|} \sum_{\mathbf{i}\in\mathbf{W}_{\mathbf{i}}} p_{inlier}\left(v(\mathbf{i})|v(\mathbf{N}_{\mathbf{i}}^{*})\right). \tag{13}$$

Summarizing, in each iteration we compute the mean and variance of the inlier distribution

$$p_{inlier}\left(v(\mathbf{i})|v(\mathbf{N}_{\mathbf{i}}^{*})\right) = \mathcal{N}(\mu_{\mathbf{i}}, \sigma_{\mathbf{i}}) \tag{14}$$

for each pixel $\mathbf{i}$, *i.e.*, one NL-Means step, as well as α and the probability of pixel $\mathbf{i}$ being an inlier according to

$$P(\mathbf{i} = \text{ inlier }) = \theta\, \alpha\, p_{inlier}\left(v(\mathbf{i})|v(\mathbf{N}_{\mathbf{i}}^{*})\right). \tag{15}$$

A user-specified parameter θ is used to determine the sensitivity of the detector. Outliers are finally removed based on the binary decision

$$P(\mathbf{i} = \text{ inlier }) < 0.5. \tag{16}$$

Once outliers in the depth map have been detected, the smoothing algorithm described in Section 2.2 can be applied. We exclude outlier pixels in the sum of Equation 1, *i.e.*, we sum over $\mathbf{j} \in \mathbf{W}_{\mathbf{i}} \setminus \{\mathbf{k}\}$ where $v(\{\mathbf{k}\})$ have been classified as outlier values. Furthermore, we exclude outlier pixels when computing the patch similarity according to Equation 2.

4. PARALLELIZATION

In this section we discuss the need for a fast implementation of both the original NL-Means algorithm as well as our robust non-local denoising algorithm. This is motivated by the complexity of the original algorithm and is of even higher interest in the context of our algorithm, which incorporates an iterative application of the core algorithm for the outlier detection step.

Since programmable graphics hardware (GPUs) and multi-core CPUs are widely available, we explore different methods to efficiently parallelize our denoising scheme. This is in contrast to the work done by Buades [19], who focuses on faster versions of the original serial algorithm.

Kharlamov et. al [20] presented a GPU implementation of the original NL-Means filter for denoising of color images, as well as an optimized version of the algorithm, computing only one distance per patch. Especially the latter algorithm runs very fast and while the approximations result in artifacts that are noticeable but tolerable on color data, these effects are very disturbing on depth data.

We present different implementation independent data and task decomposition strategies in Subsection 4.1 and investigate their applicability and the resulting speed-ups that can be achieved by a GPU implementation in Subsection 4.2.

4.1. General Considerations

Note that when discussing parallel programming principles, we keep to the nomenclature described by Grama et al. [21]. As described earlier in Sections 2 and 3, the algorithm can be divided into two separate parts, namely the iterative outlier detection followed by the final smoothing step. To simplify the discussion of the different parallel formulations of the algorithm, each step of the outlier detection can be thought of as a single smoothing step. The only difference is, that instead of computing a smoothed depth value per pixel, the outlier probability per pixel is calculated.

Given this simplification, the complexity of our denoising algorithm performing i iterations in the outlier detection step on a depth map of width s_x and height s_y with search window ($\mathbf{W}$) radius $r_\mathbf{W}$ and patch ($\mathbf{N}$) radius $r_\mathbf{N}$ is

$$\left(s_x s_y \times (2r_\mathbf{W}+1)^2 \times (2r_\mathbf{N}+1)^2\right) \times (i+1). \quad (17)$$

As for practical numbers, when using a straightforward serial version of the algorithm running on a normal PC (QuadCore 2.4Ghz) it takes about 14 minutes to denoise a depth map of size 160×120 with parameters $r_\mathbf{W} = 8$, $r_\mathbf{N} = 3$ and $i = 10$. Since it is desirable to expand the size of the search window radius as much as possible while striving for better performance, it is useful to examine possible parallel versions of the algorithm.

It is easy to see, that the parallel computation has to be synchronized after each iteration, since the following calculations rely on the results of the previous iteration. Therefore it is interesting, how to efficiently parallelize one single iteration of the algorithm. The computation of the result of one single pixel is independent of all other results making our algorithm a perfect candidate for a data decomposition technique and it is reasonable and easy to perform partitioning on the output data. Dividing the destination image into non-overlapping regular blocks of pixels induces a straightforward parallel formulation of the algorithm, since no communication across tasks has to be performed: Each task is assigned a block of pixels to compute the corresponding results for each pixel separately. Branching of the control flow only occurs when the computation has to be performed on pixels which are close to the boundary such that their corresponding patch contains pixels that extend beyond the actual image. Since this is not the general case, a minimal synchronization overhead can be expected and load balancing will be of no issue.

Depending on the number of available processors or the type of processor (CPU or GPU), different granularities of the decomposition can be considered. The coarsest one is probably most suitable for a CPU multi-core implementation, where the number of tasks is equal to, or a multiple of, the total number of physical processors. This can alleviate the typically large overheads of thread creation and switching of threads. A more fine-grained formulation can be obtained by blocks consisting of only one pixel, leading to a huge number of tasks suitable for a massively parallel architecture like a GPU, where thread switching is comparably cheap. When looking at the algorithm itself, we compute for every output pixel the intra-patch similarity (see Equation 4) of the current patch and a lot of weighted squared distances between the pixels of both patches (one is fixed for the current pixel, the other ”slides” over all pixels inside the window radius) and their sum for normalization. Note here, that one obvious optimization is to precompute the intra-patch similarity for each pixel of the current patch. An even more fine-grained formulation for potentially large patch sizes can be obtained when the pixel distances between two patches are computed in parallel followed by a parallel reduction step to calculate the final weight.

4.2. CUDA Implementation

For an actual implementation of the parallel version of our non-local denoising algorithm, we employ the CUDA framework that exposes the GPU as a data-parallel computing device capable of executing a very high number of threads in parallel. We implemented the algorithm using the different granularities presented in the previous subsection and illustrate the limitations leading to actually only one feasible version.

A coarse-grained implementation, performing the filtering of multiple pixels per task is limited due to the maximum allowed runtime for a single task (the so called ”kernel” in the CUDA jargon) of approximately 10 seconds, so a more fine-grained data and task decomposition has to be applied in order to lower the runtime of a single task. The other extreme (computing the pixel distances between two patches in parallel) leads to a huge number of kernel invocations. Although a single kernel invocation has very little overhead, this deteriorates performance by an order of magnitude compared to the coarse implementation. By far the most promising variant is to assign each task a single pixel to perform the filtering for. The precalculation of the intra-patch similarity mentioned above is only possible for small patch sizes, since the available fast local memory (”shared memory”) for each multiprocessor is rather limited. This variant of the implementation also benefits from texture cache usage, since the per-task accesses to the source image are spatially close to each other.

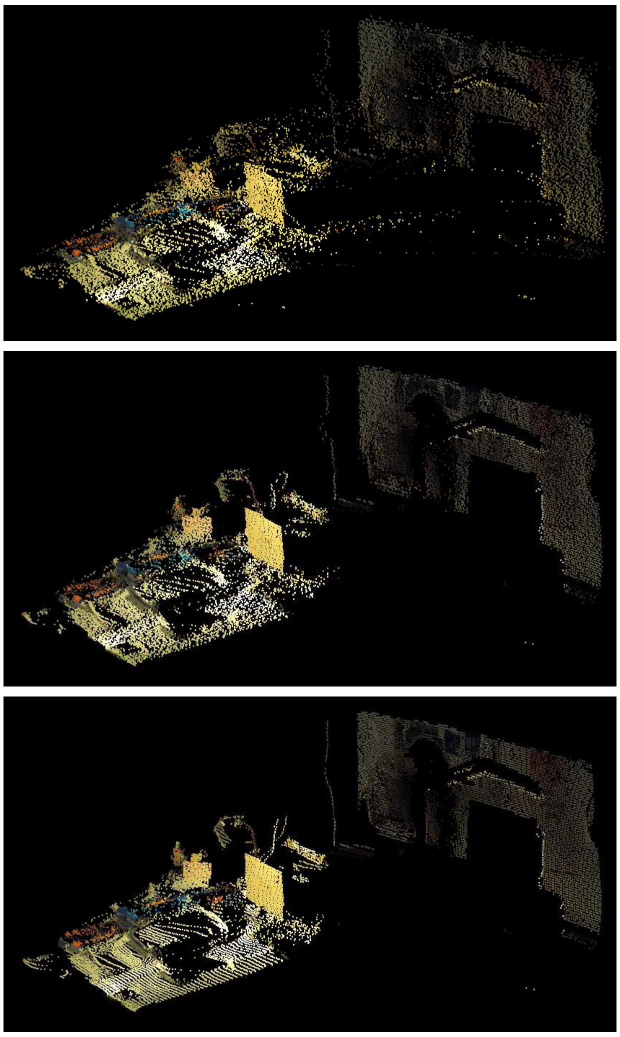


Figure 2. Denoising results. Raw data (top), joint bilateral filter (middle) and robust non-local denoising (bottom). Outliers (except raw) are removed using the iterative non-local approach prior to smoothing.

5. RESULTS

We apply the robust NL-Means Filter to data acquired with a *PMD [vision] 19k* time-of-flight camera. An additional industrial color camera is mounted ontop resulting in a 45mm parallax. Both cameras are calibrated laterally using a standard tool and a nearest-neighbor lookup of color values is performed, mapping the depth values into the color camera frame. No further depth calibration is applied.

Figures 2 and 3 show results using the robust non-local denoising filter on a challenging scene that contains fine structure as well as large differences in depth. For comparison the same outlier set is also pruned before applying the joint bilateral filter where for the reference implementation the approach of [12] is used, yet working on the native resolution of the PMD camera. We observe a better handling of discontinuities and preservation of fine details using the robust non-local approach. Especially, a better reconstruction is achieved for the desktop which is oriented almost parallel to the viewing direction. Results from a second scene are shown in Figure 5.

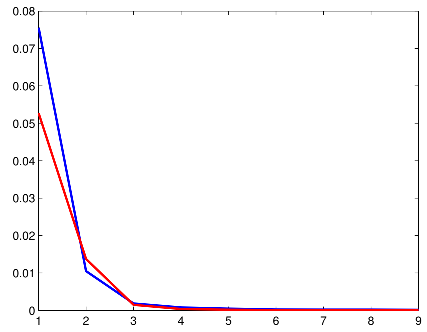


Figure 6. Average squared difference of the outlier probability per iteration for the scenes shown in Figs. 2, 5.

As mentioned in Section 4, the robust non-local denoising algorithm is computationally expensive. Using the parallelization technique presented earlier, we achieve computation times comparable to less demanding postprocessing methods. To evaluate the performance and scalability, we focus on one iteration of our algorithm. Looking at Equation 17 again, it is easy to see that for a single iteration, the runtime is expected to grow quadratically when increasing the search window radius $r_{\mathbf{W}}$ or the patch radius $r_{\mathbf{N}}$. Doubling the resolution of the source image should result in a quartic scaling. We conducted experiments where two of the parameters remained fixed while varying the third one. Figure 4(a) illustrates the effect of varying the patch radius $r_{\mathbf{N}}$ while keeping $r_{\mathbf{W}} = 8$ and $s_x = s_y = 512$ fixed (red curve) and the corresponding results when varying the search window radius $r_{\mathbf{W}}$ with fixed $r_{\mathbf{N}} = 3$ and $s_x = s_y = 512$ (blue curve). The x-axis specifies $r_{\mathbf{N}}$ and $r_{\mathbf{W}}$, while on the y-axis the square root of the runtime in seconds is plotted. We use the square root of the runtime to account for the quadratic increase in complexity, so for a good parallel implementation a linear plot is desirable. It can be seen that this is the case for both the blue and red curves, the latter one revealing some stepping artifacts that we account a suboptimal texture cache usage for. In Figure 4(b) we show the results of fixed patch and search window radii while doubling the image resolution. The y-axis is a function of the 4th root of the runtime, since a quartic scaling is expected. Our parallel implementation exhibits a slower increase in runtime for lower resolutions because a lot of the patches contain pixels outside the actual image. When performing filtering on higher resolutions, the kernel invocation overhead mentioned in Subsection 4.2 is becoming increasingly apparent. For comparable results, we use a patch radius of $r_{\mathbf{N}} = 3$ (implying a Gaussian kernel with bandwidth $a = 1.5$), a search window with radius $r_{\mathbf{W}} = 8$ and the native resolution of the depth image ($s_x = 160, s_y = 120$). On a NVidia GeForce 8800GTX the algorithm takes 1.9 seconds performing 10 iterations

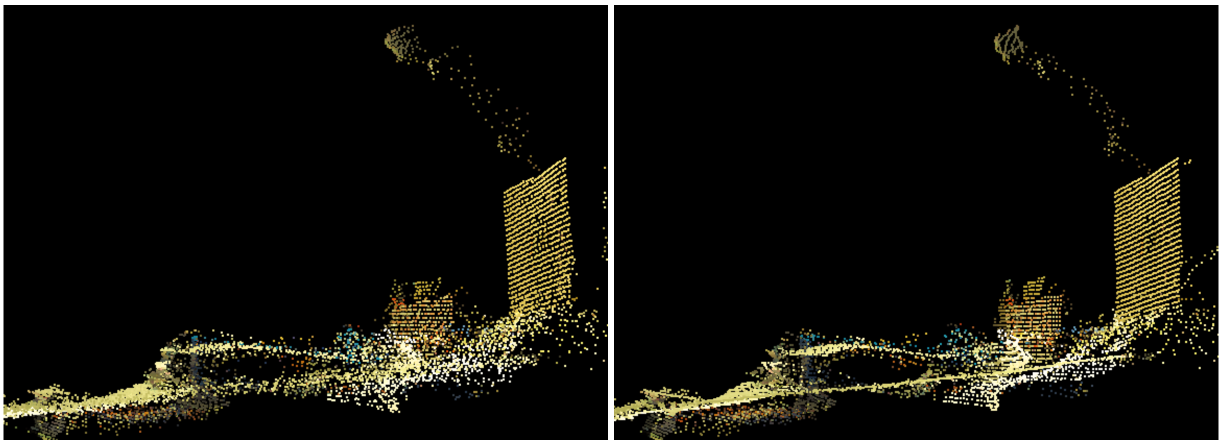
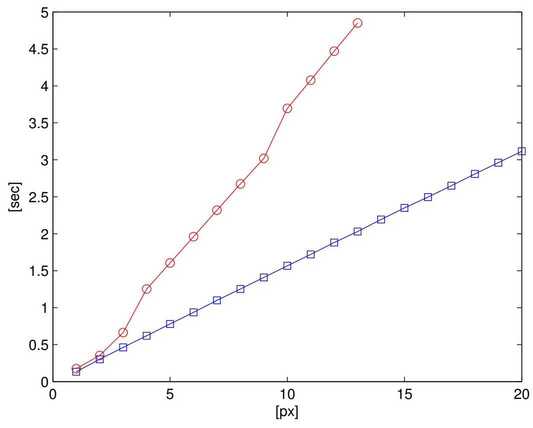
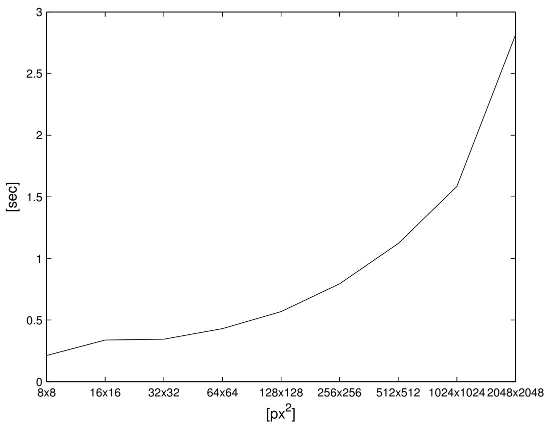


Figure 3. Detail view of the denoising result using joint bilateral filter (left) and our approach (right).



(a) Plots of the algorithm runtime for varying r_N (red, circles) and varying r_W (blue, squares). The x-axis specifies r_N and r_W respectively, while on the y-axis the square root of the runtime in seconds is plotted.



(b) Algorithm runtime with varying resolution of the source image. Image resolution is specified by the x-axis, while the y-axis is a function of the 4th root of the runtime.

Figure 4. Evaluation of one parallelized iteration of of our robust non-local denoising algorithm.

in the outlier detection step, compared to the CPU version that takes approximately 14 minutes to compute. For performance evaluations we used a fixed number of iterations in the outlier detection step which was chosen on the basis of the convergence results illustrated in Figure 6.

6. CONCLUSION

We reviewed our unified non-local denoising technique that explicitly handles outliers in the input data and is applicable to depth data with strong discontinuities. While our approach exhibits a performance in the outlier detection step comparable to the well-established tensor-voting framework and is superior to state-of-the-art smoothing techniques, it is computationally expensive. Since multicore CPUs and programmable graphics hardware (GPUs) are widely available, we examined a parallel formulation, presented different data and task decomposition strategies and investigated their applicability towards a GPU implementation. We demonstrated that our algorithm scales very well and achieves significant speed-ups compared to a CPU version.

7. REFERENCES

[1] Marvin Lindner and Andreas Kolb, "Calibration of the intensity-related distance error of the PMD TOF-Camera," in *SPIE: Intelligent Robots and Computer Vision XXV*, 2007, vol. 6764, pp. 6764–35.

[2] Sigurjon A. Gudmundsson, Henrik Aanæs, and Rasmus Larsen, "Environmental Effects on Measurement Uncertainties of Time-of-Flight Cameras," in *Proc. International Symposium on Signals Circuits and Systems (ISSCS)*, 2007.

[3] Benjamin Huhle, Timo Schairer, Philipp Jenke, and Wolfgang Straßer, "Robust Non-Local Denoising of Colored Depth Data," in *IEEE CVPR Workshop on Time of Flight Camera based Computer Vision*, 2008.

[4] Antoni Buades, Bartomeu Coll, and Jean-Michel Morel, "A Non-Local Algorithm for Image Denoising," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.

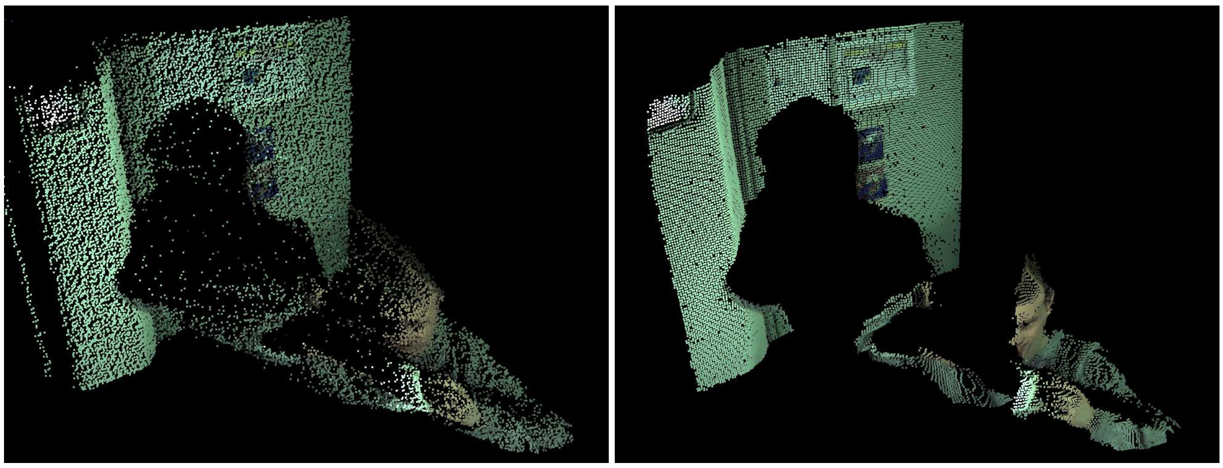


Figure 5. Input (left) and denoising result using our robust non-local denoising technique (right).

[5] Leonid P. Yaroslavsky, *Digital Picture Processing. An Introduction*, Springer Verlag, 1985.

[6] Carlo Tomasi and Roberto Manduchi, “Bilateral Filtering for Gray and Color Images,” in *Proc. IEEE International Conference on Computer Vision (ICCV)*, 1998, pp. 839–846.

[7] S. M. Smith and J. M. Brady, “SUSAN – A new Approach to Low Level Image Processing,” Tech. Rep., Oxford University, 1995.

[8] Michael Elad, “On the Origin of the Bilateral Filter and Ways to Improve It,” *IEEE Transactions on Image Processing*, vol. 11, no. 10, pp. 1141–1151, 2002.

[9] Danny Barash and Dorin Comaniciu, “A Common Framework for Nonlinear Diffusion, Adaptive Smoothing, Bilateral Filtering and Mean Shift,” *Image and Video Computing*, vol. 22, no. 1, pp. 73–81, 2004.

[10] Sylvain Paris, Pierre Kornprobst, Jack Tumblin, and Fredo Durand, “A Gentle Introduction to Bilateral Filtering and its Applications,” in *ACM SIGGRAPH Course*, 2007.

[11] Georg Petschnigg, Maneesh Agrawala, Hugues Hoppe, Richard Szeliski, Michael Cohen, and Kentaro Toyama, “Digital Photography with Flash and No-Flash Image Pairs,” in *Proc. SIGGRAPH*, 2004.

[12] Johannes Kopf, Michael F. Cohen, Dani Lischinski, and Matt Uyttendaele, “Joint Bilateral Upsampling,” in *Proc. ACM SIGGRAPH*, New York, NY, USA, 2007, p. 96.

[13] Shachar Fleishman, Iddo Drori, and Daniel Cohen-Or, “Bilateral Mesh Denoising,” in *Proc. ACM SIGGRAPH*, 2003.

[14] Michael Wand, Alexander Berner, Martin Bokeloh, Philipp Jenke, Arno Fleck, Mark Hoffmann, Benjamin Maier, Dirk Stanecker, Andreas Schilling, and Hans-Peter Seidel, “Processing and Interactive Editing of Huge Point Clouds from 3D Scanners,” *Computers & Graphics*, vol. , pp. , 2008, to appear.

[15] Alexei A. Efros and Thomas K. Leung, “Texture Synthesis by Non-parametric Sampling,” in *Proc. IEEE International Conference on Computer Vision (ICCV)*, 1999.

[16] Yong Seok Heo, Kyoung Mu Lee, and Sang Uk Lee, “Simultaneous depth reconstruction and restoration of noisy stereo images using Non-local Pixel Distribution,” in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2007.

[17] Oliver Schall, Alexander Belyaev, and Hans-Peter Seidel, “Feature-preserving Non-Local Denoising of Static and Time-Varying Range Data,” in *Proc. ACM Symposium on Solid and Physical Modeling*, 2007.

[18] A. P. Dempster, N. M. Laird, and D. B. Rubin, “Maximum Likelihood from Incomplete Data via the EM Algorithm,” *Journal of the Royal Statistical Society*, vol. 39, pp. 1–38, 1977.

[19] Antoni Buades, *Image and film denoising by non-local means*, Ph.D. thesis, Universitat de les Illes Balears, 2005.

[20] Alexander Kharlamov and Victor Podlozhnyuk, “Image Denoising,” Tech. Rep., NVIDIA, Inc., 2007.

[21] Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar, *Introduction to Parallel Computing*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, second edition, 2002.