Machine Vision and Applications
DOI 10.1007/s00138-011-0346-8

ORIGINAL PAPER

Efficient large-scale multi-view stereo for ultra high-resolution image sets

Engin Tola · Christoph Strecha · Pascal Fua

Received: 29 September 2010 / Revised: 11 February 2011 / Accepted: 9 May 2011

Abstract We present a new approach for large-scale multi-view stereo matching, which is designed to operate on ultra high-resolution image sets and efficiently compute dense 3D point clouds. We show that, using a robust descriptor for matching purposes and high-resolution images, we can skip the computationally expensive steps that other algorithms require. As a result, our method has low memory requirements and low computational complexity while producing 3D point clouds containing virtually no outliers. This makes it exceedingly suitable for large-scale reconstruction. The core of our algorithm is the dense matching of image pairs using DAISY descriptors, implemented so as to eliminate redundancies and optimize memory access. We use a variety of challenging data sets to validate and compare our results against other algorithms.

Keywords Multi-view stereo · 3D reconstruction · DAISY · High-resolution images

1 Introduction

Multi-view stereo reconstruction of complex scenes has made significant progress in recent years, as evidenced by the quality of the models now being produced. However, most state-of-the-art approaches do not exploit the very high resolution—20 Megapixel and more—that modern cameras can readily acquire. Instead, they rely on moderate sized images, 1–4 Megapixel in recent papers [6,10,17,26,28], and assume a small capture space that makes it possible to use visual hull constraints and volumetric optimization algorithms. Because these are computationally expensive and have large memory requirements, this, in turn, limits the size of the images that can be used. The few approaches that have been shown to scale up to larger images, 2–6 Megapixel, rely on local plane sweeping strategies [6–8,27], which tend to be very slow when applied to larger images. Admittedly, some algorithms are capable of operating on higher resolution images as a whole [8] or by dividing them into overlapping tiles of smaller sub-images and later combining the results [9] but none have used whole images as large as we consider here, mainly due to their inability to scale linearly in computation time for image resolution or for their high memory requirements.

In this paper, we show that higher resolution images, with their richer texture, can both provide more reliable matches than lower resolution ones and be handled fast enough for practical large scale use. To this end, we introduce an approach, which has a very low memory and computational cost requirement, for

- producing quickly point clouds such as the one depicted in Fig. 1, which contain far fewer outliers than those recovered by techniques that rely on plane sweeping [6–8,27];
- exploiting images as large as 40 Megapixel, such as the 31 images depicted in Fig. 2, while reducing the processing time to less than 15 min instead of the many hours that most state-of-the-art algorithms [6–8,27] would have required.

E. Tola (✉)
Aurvis Ltd, Ankara, Turkey
e-mail: engin.tola@gmail.com
URL: http://www.aurvis.com

C. Strecha · P. Fua
Computer Vision Laboratory, EPFL, Lausanne, Switzerland
e-mail: christoph.strecha@epfl.ch

P. Fua
e-mail: pascal.fua@epfl.ch

Published online: 28 May 2011

Fig. 1 Statue reconstruction. The data set contains 127 18-Megapixel images of a statue. The reconstruction time was 29.5 min and the final cloud contains 15.3 million points. *First row* Some images from the data set. *Second row* The depth maps computed for the images shown in the *first row*. *Last two rows* The renderings of the cloud in its raw and colored form from different viewpoints (color figure online)

Our approach operates directly at the highest resolution, which is key for achieving both accuracy and robustness. This is in complete contrast to typical hierarchical approaches that are popular in the field and the effectiveness of our approach therefore constitutes an important and novel result. What makes this possible is the DAISY descriptor [25] that has been shown to be very powerful for dense wide baseline matching. However, while in our earlier work [25], we focused on using it to compute a data term for use within a graph cut framework, we show here that processing high-resolution images reduces the need for smoothness priors to the point where such a computationally demanding optimization scheme becomes unnecessary. Instead, we introduce a much faster approach for checking match consistency across multiple frames and rejecting erroneous matches, and the resulting algorithm produces very dense and accurate 3D clouds such as those depicted in Figs. 1, 2, and 3 in very acceptable computation times, as can be seen in Table 2.

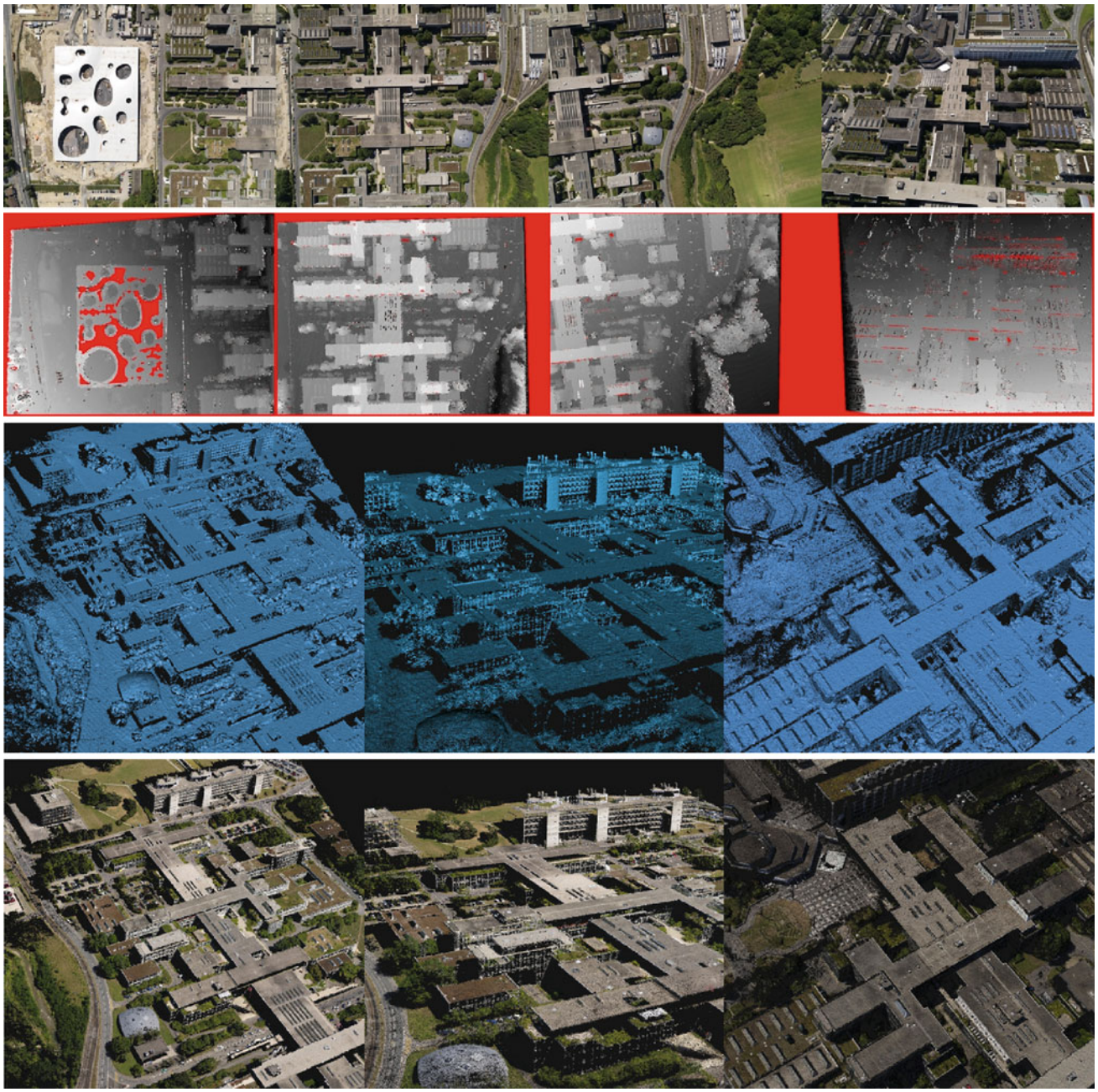

Fig. 2 EPFL campus reconstruction. The data set contains 31 40-Megapixel images of the EPFL campus shot from a helicopter. The reconstruction time was 14.2 min and the final cloud contains 11.3 million points. *First row* Some images from the data set. *Second row* The depth maps computed for the images shown in the *first row*. As can be seen, depth maps, while being mostly correct, contain some outliers. *Third row* The final point cloud from different view points in its raw form. *Last row* The colorized renderings of the same view points shown in *third row*. Notice that the few erroneous points present in the depth maps are filtered out and final cloud contains virtually no outliers (color figure online)

Our most important contribution in this work is the development of an extremely memory efficient and computationally simple MVS system that requires very little memory. Because of this our method can scale to very large scale, very high resolution image sets on standard desktop systems easily and can build reconstructions in very short computation times compared to other state-of-the-art approaches.

2 Previous work

Multi-view stereo (MVS) approaches can be roughly divided into small-scale and large-scale methods. Small-scale methods are those, such as [5,6,10,17,26,28], that assume a small capture space and can therefore take advantage of visual hull constraints or volumetric optimization algorithms.

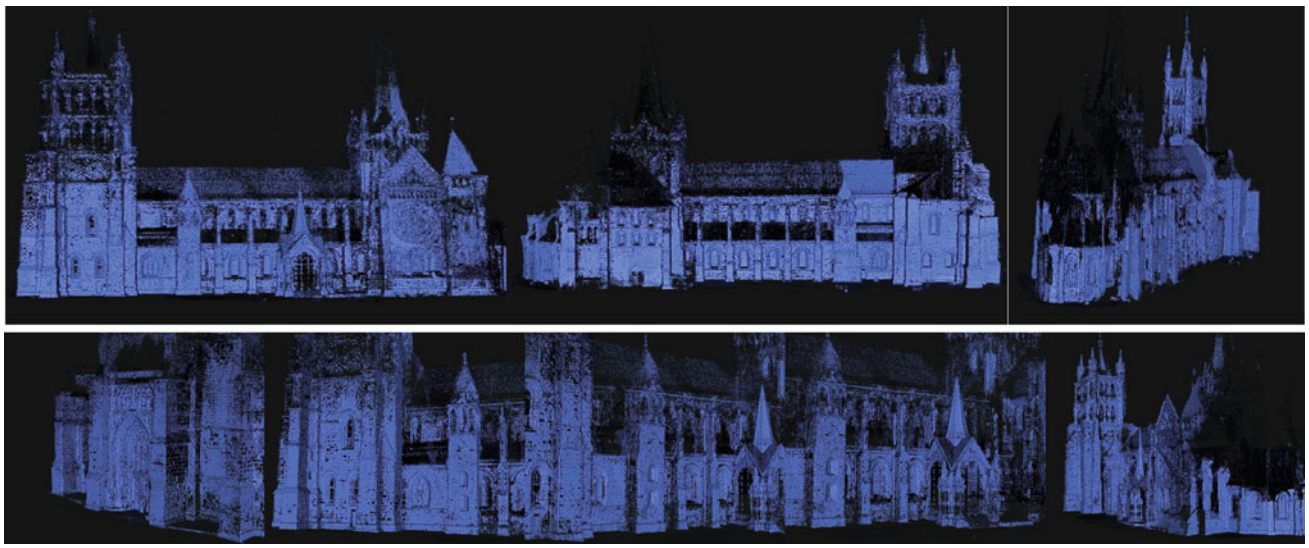

Fig. 3 Lausanne Cathedral-Ground reconstruction. The 3D point cloud is rendered from several viewpoints. It was constructed in 419 min from 1,302 21-Megapixel images and contains 148.2 million points

While being very effective, these methods do not scale up to modeling extended scenes because the memory and computational requirements would become prohibitive. Thus, large-scale methods use different representations, as discussed below.

A number of techniques replace the unified representation of volumetric methods by a set of depth maps [3,7,14,19,27]. They scale up to very large scenes, sometimes at the cost of losing some of the accuracy and completeness of volumetric methods [16,22]. This class of MVS methods include global approaches [20,23,25] that impose smoothness priors in the form of Markov Random Fields and local ones [1,19] that rely on gradient-based optimizations to enforce local smoothness. The accuracy of the global methods is limited by the fact that they can only handle relatively small resolution images. This is due to the very large amounts of memory required to store many potential depth states for many pixels. Memory issues also limit GPU-based implementations since it is hard to effectively deal with very large images that do not fit into the relatively small memories of GPUs [3,14]. By contrast, local approaches [1,19] can deal with high-resolution images, in principle at least. However, not only are they relatively slow, but they also require initial depth estimates and do not converge well if these are not sufficiently close to the desired solution.

A different approach for handling large-scale scenes and images of arbitrary sizes is to directly reconstruct oriented point clouds [6,8]. This is achieved by making it unnecessary to incorporate smoothness priors into the computation. Instead, the algorithms rely on visibility constraints to filter the points and reject erroneous ones. For example, in [6], a very dense and accurate point cloud is computed by locally estimating planar-oriented patches. A similar approach is proposed in [8] with the addition of an energy minimization-based method to compute and refine a mesh that represents the 3D points. Both produce very impressive results and outperform most other algorithms on many different data sets [18].

In spirit, our approach is very close to that of [6], except for the fact that using the DAISY descriptor [25] lets us handle much higher resolution images by removing the need for computationally intensive optimizations and produce higher quality point clouds in a fraction of the time. We use temporary depth maps while computing an oriented point cloud but do not impose any smoothness. The inherent smoothness and robustness of the DAISY descriptor makes it unnecessary and lets us skip many of the expensive computations required by other algorithms. In principle, our results could be further refined using a method such as [8], but their quality and density is such that this is barely necessary.

3 Approach

Ours is a two-step approach. We start by computing dense point clouds from image pairs and then check for consistency using additional ones. This is an effective way to ensure with high probability that only correct points are retained, especially if the pairwise clouds are of good quality. In the remainder of this section, we first sketch these two steps and then provide more specific implementation details.

3.1 Pairwise point clouds

Given an image pair, $\Upsilon_i = (I_s, I_t)$, whose baseline might be relatively large, we use the DAISY descriptor we introduced in earlier work [25] to measure similarity along epipolar lines

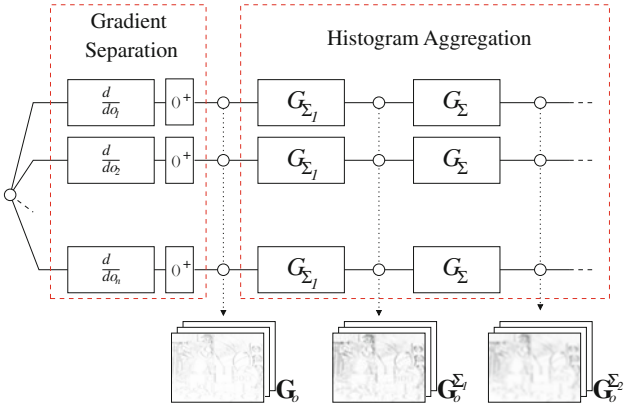

Fig. 4 Computing the DAISY descriptor. DAISY is composed of concatenated gradient histograms that are computed by first computing gradients in separate orientation layers and then aggregating the weighted responses within each layer into orientation histograms. Using Gaussian kernels for weighting allows to use separated 1D kernels in the implementation and chaining smaller convolution responses to get larger ones. This design pipeline allows for very efficient descriptor computation over the whole image

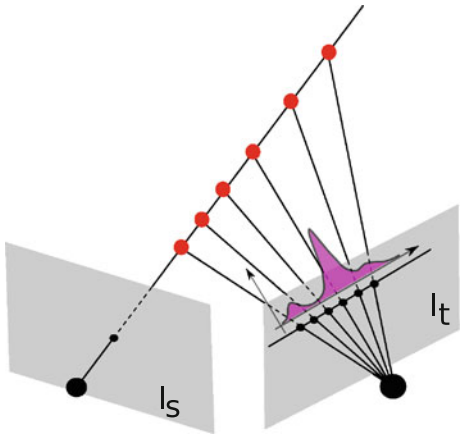

Fig. 5 Searching along an Epipolar line. The line of sight emanating from a pixel on I_s is discretized non-uniformly such that the resulting samples, depicted by *black dots*, project at uniform intervals on I_t. The probability distribution, represented in *purple*, is computed over the uniformly sampled epipolar line. Its maximum is only considered to correspond to a valid disparity if it is significantly larger than all others

and compute a dense depth map. DAISY is composed of concatenated gradient histograms that are computed by first estimating gradients into separate orientation layers and aggregating their magnitudes within each layer into orientation histograms, as shown in Fig. 4. This can be done by simply thresholding and convolving the oriented gradients with Gaussian filters of various sizes. It produces the same kind of invariance as SIFT [13] or SURF [2] histogram building but can be computed much more efficiently for every single image point and in every direction. DAISY is therefore ideally suited for dense matching.

In [25], we introduced an EM framework based on max-flow min-cut optimization to estimate both depths and occlusions, which made the complete approach computationally very intensive. By contrast, here we discard this optimization scheme. Instead, we directly use the DAISY matching score to compute the probability of a pixel x having a depth d in one image as

$$P(d) = \frac{1}{Z} \exp\left(-\frac{\| D_x^i - D_{x'(d)}^j \|^2}{\sigma}\right), \tag{1}$$

where D_x^i and $D_{x'(d)}^j$ are the descriptors at x in one image and at the corresponding point $x'(d)$ in the other, assuming that the depth is indeed d. The sharpness of the distribution is controlled by σ and Z is a normalizing constant that ensures that the probabilities sum to one.

To decide whether or not to assign a depth to a pixel, we look for the first two probability maxima along the uniformly sampled epipolar line and consider the ratio of their values:

$$R_X = \frac{P_{\text{best}}(d)}{P_{\text{second best}}(d)}. \tag{2}$$

We treat the depth as valid if it is above a threshold, which we take to be 0.8 for all results presented here. The choice for this threshold will be justified in Sect. 5.1.

This approach is depicted in Fig. 5. As will be shown in the results section, we tested the performance of this very simple decision rule under changing image resolution, camera baseline, and descriptor parameters and found it to be very reliable.

This produces a dense point cloud, which we denote as $\overline{X}^i$. These pairwise clouds may, of course, still contain a few spurious points.

3.2 Enforcing consistency

To eliminate these spurious points, we only retain those for which there is a consistent evidence in multiple pairs. Given a 3D point, X, computed from one specific pair, $\Upsilon_i = (I_s, I_t)$, consistency is measured by reprojecting the point in other images and computing

$$\epsilon_{i,j}(X) = \frac{|d(X,i) - d\text{map}_{i,j}(X)|}{d\text{map}_{i,j}(X)}, \tag{3}$$

where $d(X,i)$ is the depth of the 3D point X with respect to camera i and $d\text{map}_{i,j}(X)$ is the depth value computed at the projection of X in image i using the image pair (i,j). A 3D point is retained if this consistency measure is small enough for at least C depth maps. Formally, we write

$$\overline{\mathbf{X}} = \left\{ \{X_j\}, \text{iff} \left[\sum_{i \in Q_j} V(X_j \in \overline{X}^i) \right] > C \right\}, \tag{4}$$

Fig. 6 Lausanne Cathedral-Aerial Reconstruction. The 3D point cloud was constructed in 22 min from 61 24-Megapixel aerial images and contains 12.7 million points. *First row* Some images from the dataset. Courtesy of J.M. Zellweger. *Last two rows* The renderings of the colorized points from several viewpoints (color figure online)

where $\overline{\mathbf{X}}$ is the final point cloud, $\overline{X}^i$ is the 3D point cloud for the sub-maximization problem for Υ_i introduced above, $V(.)$ is a boolean function that returns 1 if its argument is true and 0 otherwise and Q_j represents a set of image pairs. This definition inherently subsumes the common left-to-right and right-to-left tests often used by correlation-based approaches [4]. This stems from the fact that the depth map obtained by reversing the roles of the cameras can be among the ones chosen for verification (Figs. 6, 7, 8).

Since a point can be instantiated from several different image pairs, for optimum results, it is important to only retain the one whose precision can be expected to be highest and to ignore the others. This is done by considering three geometric factors: (i) baseline of the stereo pair, (ii) focal length of the camera, and (iii) distance of the point to the camera center. Factor (i) affects the expected precision of the point since larger baselines tend to yield more precise depth estimates. Factors (ii) and (iii) control the information content as a closer camera or a zoomed-in one typically yield more textured views. Thus, we take the precision estimate to be

$$q(X) = \frac{f * \sin(\theta)}{||X - C||}, \tag{5}$$

where f is the focal length, $sin(\theta)$ the baseline measure with θ being the angle between the two camera rays, and $||X-C||$ is the distance to the camera center. When merging the pairwise clouds, we cluster 3D points and only retain from each cluster the one with the highest precision. Not only does this ensure that our final cloud is formed of high precision points, it also provides expected precision measures that could be used to determine where additional images are needed so as to guide further processing.

4 Implementation details

The two main computational steps introduced in Sects. 3.1 and 3.2 are conceptually simple but require careful implementation to yield accurate results and to run fast on many big images, such as those presented in this paper. Here, we provide some of the critical details.

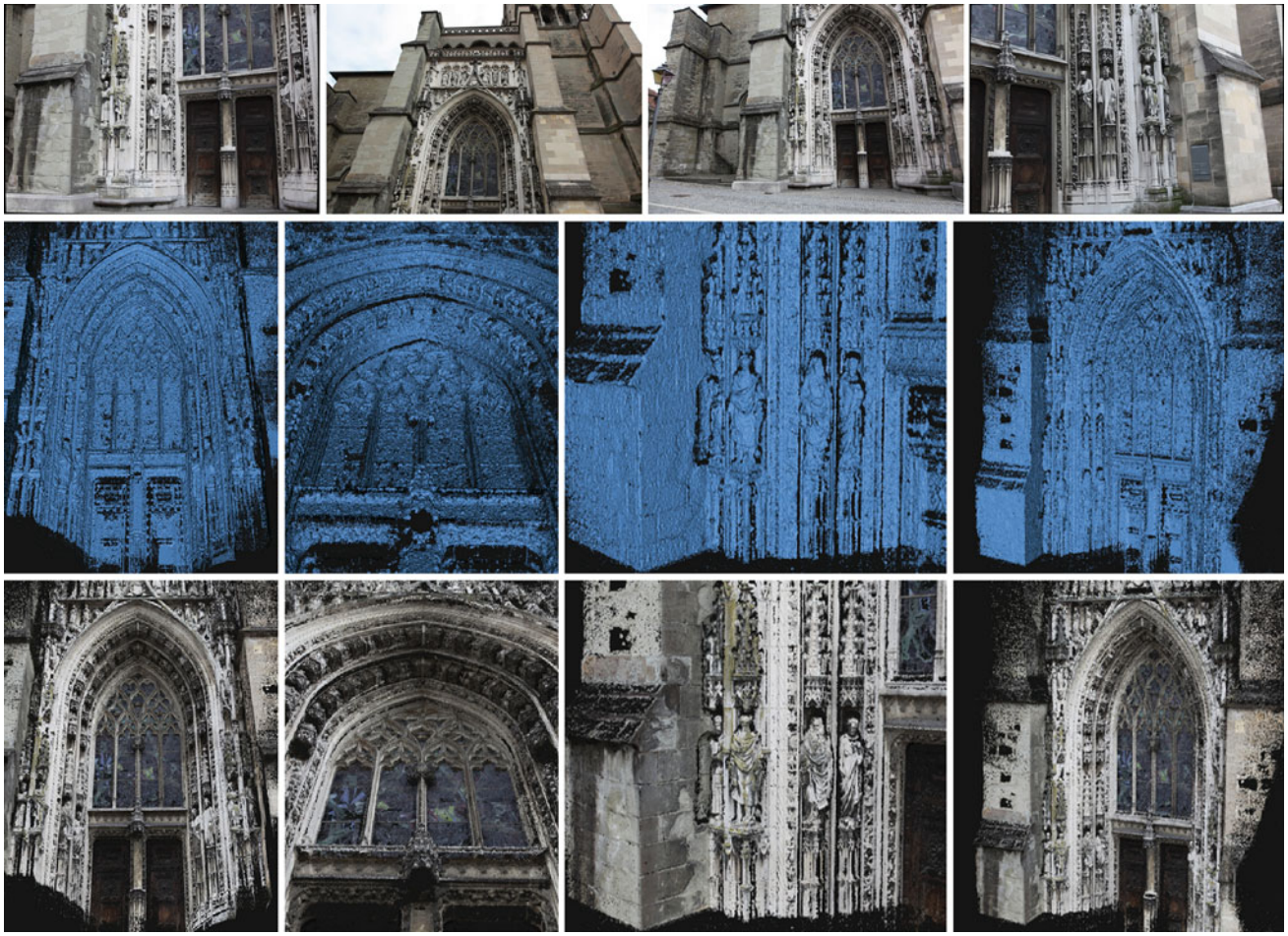

Fig. 7 Cathedral portal reconstruction. The 3D point cloud was constructed in 9.2 min from 13 21-Megapixel images and contains 20.3 million points. *Top row* Some example images of the data set. *Middle row* Computed 3D raw points are rendered from different view points and *bottom row* shows the colorized rendering of these view points (color figure online)

4.1 Viewpoint selection

View selection is very important since we need to pair images that our algorithm is suited for to achieve a good precision. To this end, we exploit the following metrics to decide if two images are suitable candidates:

- The images should be close enough to one another to be easily matched but should also be sufficiently separated for reliable depth estimation. In general, we quantify this by measuring the angle between the camera principal rays. However, if the 3D points computed as a by-product in the camera calibration stage are available, the angles between these *calibration points* and the camera centers are averaged for all the visible points to compute this metric.
- Since the DAISY descriptor is not scale invariant, images should be close to each other in scale. Rough scale estimates are computed by projecting five virtual spheres arranged in a cross-hair pattern at the mean value of the expected depth range. The ratio of the radii of the projected spheres is used to measure the scale difference between images. As before, if the calibration points are available, we place a sphere at all the visible calibration points, project them onto the cameras, and compute the scale estimate by averaging the ratio of these projections.

Cameras with a baseline larger than 10° and smaller than 30° and scale difference between 0.8 and 1.2 are assumed to be matchable. In practice, one single depth map is computed per image by pairing it up with an image that has the closest scale in the matchable set.

4.2 Depth map estimation

The matching probabilities, given in Eq. 1 for a pixel, are computed at regular intervals along the epipolar lines. If we were to sample the depth space uniformly instead

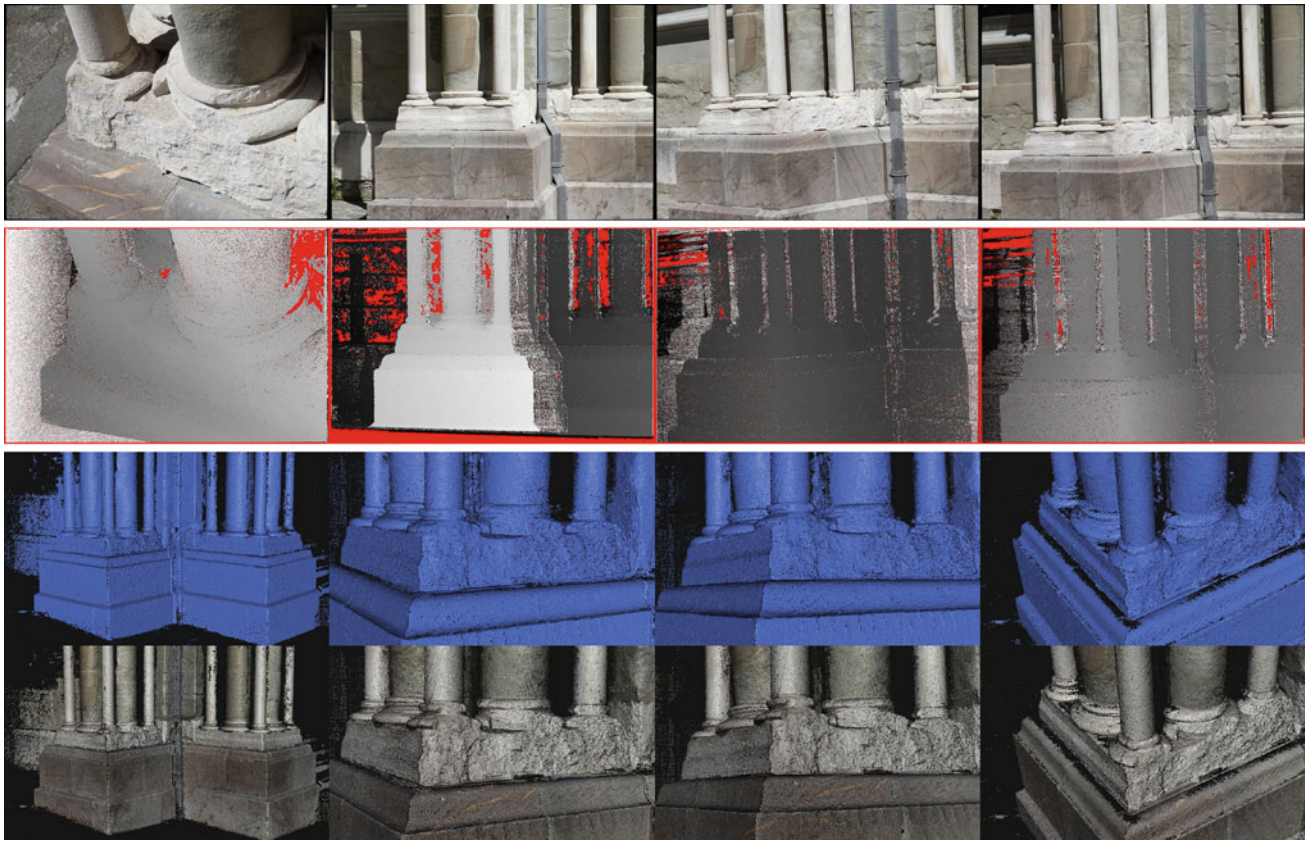

Fig. 8 Pillar reconstruction. The dataset contains 214 18-Megapixel images of a building pillar. The reconstruction time was 48.9 min and the final cloud contains 63.1 million points. *First row* Some example images, *second row* their depth maps, and *last two rows* the renderings of the cloud from different viewpoints in its raw and colored form, respectively (color figure online)

of its projection onto the image plane, as is done in most plane sweeping-based methods, the probability distribution would be compressed or expanded non-uniformly depending on camera positions and pixel locations. This sampling has several advantages: (i) it removes the dependency of the probability distribution from the geometry of the specific matching problem, (ii) it produces a better modeling of the distribution, (iii) it enables a finer control over the resolution of the search space, and (iv) it allows us to generically define rules on the distribution without taking care of special conditions. In addition, a preset number of depth states is not desirable as it might be too small for some pixels, resulting in the correct match being missed. Alternatively, it might be too large, resulting in unnecessary computation. A closed form solution for uniform sampling of the epipolar line is given in "Appendix".

In practice, we do not initially compute full probability distributions for each pixel. Instead, we compute them at sparse locations so that we can constrain the search for neighboring ones. This has both computational and algorithmic advantages: a smaller depth-range results in fewer operations and reduces the number of potential matches, thereby increasing robustness. This effectively simulates what global optimizers, such as belief propagation and graphcuts, do by enforcing smoothness, but at a much lower computational cost. This process may fail at depth discontinuities but the resulting erroneous depths will be eliminated by the consistency checks and, therefore, will not degrade the overall performance.

Given the probability distribution, we perform non-maxima suppression and require the best and second best probabilities to be separated by at least the size of the descriptor radius. This is necessary since the descriptor characteristics change smoothly and the probabilities of nearby pixels are correlated.

Finally, to compute the matching score, we modified the original DAISY source code in a number of ways. Since a scale estimate is associated to each image at the view point selection stage presented in Sect. 4.1, we compute the descriptors for each one at the correct scale. In addition, each descriptor is computed orthogonal to the epipolar line passing through it to account for the perspective deformation of the texture and instead of performing partial histogram normalization, as suggested in the original paper, we normalize

descriptors as a whole. The descriptors are computed over relatively small regions since input images have a narrow baseline setup and thus full normalization produces more stable descriptors compared to partial normalization over these small aggregation regions. Finally, as the images to be matched have relatively close viewpoints, we use a very short version of the descriptor, as will be discussed in the results section. This stage is very efficient as descriptors are precomputed at their respective orientation and scale and matching is performed at integer resolution.

4.3 Consistency computation

Once the depth maps have been computed, they are turned into a point cloud by verifying the accuracy of each 3D point on more than one depth map. This verification would be costly if we checked each point against all depth maps. Instead, for each point, the verification is performed on the closest ten images whose corresponding camera looks towards the point.

The depth projection error, given in Eq. 3, measures the error made between two depth maps computed from different view points using different image pairs and if this error for a pixel is within a given threshold for at least $C = 3$ depth maps, the point is included to the cloud. The threshold for this error is set according to the discretization error of the depth estimation procedure described above. Since the depth maps are computed at integer resolution and since they are computed using different camera configurations, there will always be a difference in the estimated location of the same point in two depth maps. This discretization error, however, can be computed using calibration parameters but it will cost valuable computation time. Therefore, instead of doing it for each pixel, we settle for a single value for all the pixels of an image by taking the maximum discretization error value computed at only image corners as the depth projection error threshold.

Since depth maps can be of different qualities depending on the baseline of the cameras used to compute them, within a given 3D volume, we retain only the 3D point whose expected precision, as computed by Eq. 5, is greatest. To this end, a sparse octree structure is employed. Points are placed in an octree and only the point with the biggest precision value within an octree cell is retained. This way, points of lower quality are discarded when there is a point source with more precision but they are kept within the reconstruction if no other reliable point source exists.

After this filtering, the normals for the points are computed by a plane fitting procedure over a kd-tree structure from the closest 32 points. The colors of the points are assigned to them using the images in which the points are initialized from.

4.4 Memory requirements

The most memory-intensive steps of the algorithm are depth-map computation and consistency estimation.

For depth map estimation, only two images need to be loaded memory. Individual descriptors must then be computed for each one of their pixels. Since each descriptor requires 36 floating point numbers, this means that for two $X = 6$ Megapixel images, the algorithm requires approximately $X \times 36 \times 4 \times 2 = 1{,}728$ MB of memory. The output is a floating point depth value per pixel which takes $X \times 4 = 24$ MB of memory.

To enforce consistency, a 3D point from one depth-map is instantiated if it is present in at lease C others. However, as explained above, instead of checking consistency against all depth-maps, we only use the ten closest ones. This means that only 11 depth-maps need to be loaded at any one time, which represents $11 \times X \times 4 = 264$ MB of memory for our 6-Megapixel images. In practice, if enough RAM is available, we maintain several more depth maps in memory. This reduces latency times since a depth map can be used by several others to validate their 3D points.

5 Results

In this section, we present results of our multi-view stereo algorithm. This section is divided into three subsections where

- we validate our choice of parameters and compare the discriminative power of DAISY against standard normalized cross correlation (NCC) in the framework of our approach,
- we compare our results against other approaches on publicly available benchmark data sets where we also present quantitative accuracy and computation time values,
- we show results on ultra high resolution and very large-scale image sets for which the method here is primarily designed for.

5.1 Parameter choices and image resolution

In Fig. 10, the Fountain-P11 sequence, given in Fig. 9, is used to demonstrate the effect of various DAISY parameters on the accuracy and density of the computed depth maps and to compare it against basic NCC. We used the six pairs of narrow-to-wide baseline images of $3{,}072 \times 2{,}048$ resolution and the curves are plotted for the averaged results. In these experiments, we used a standard baseline version of NCC instead of more sophisticated recent implementations that involve either warping the correlation window [6] or

Fig. 9 Fountain-P11. The 3D point cloud was reconstructed in 2.2 min from 11 6-Megapixel images and contains 2.5 million points. *First two rows* The raw 3D points and their colorized versions. *Last row* The mesh computed from these points using the Poisson reconstruction algorithm (color figure online)

introducing a multi-resolution approach [14]. This is because we are not trying to prove that DAISY performs much better than these methods. Instead, we want to show that these time consuming optimizations are not necessary when one uses higher resolution images.

As discussed in Sect. 3, the probabilities of each depth state along a uniformly sampled epipolar line is computed and a depth value is assigned to a pixel if the ratio, R_x, of the probability of the depth is larger than the probability of the next best depth state by a threshold. The detection rate is measured by changing this threshold and error rates are expressed as the ratio of the error of the assigned depth to the actual depth value defined as:

$$e = \frac{|d_{\text{assigned}} - d_{\text{ground truth}}|}{d_{\text{ground truth}}}. \tag{6}$$

Figure 10a depicts the influence of image resolution on depth accuracy for a fixed set of DAISY parameters, $R = 8$, $Q = 2$, $T = 4$, and $H = 4$. For a detailed description of these parameters, we refer the interested reader to [25]. For comparison purposes, we also plot results using NCC with patch size $P = 17$ which is roughly equivalent to the area used by DAISY.

Graphs show that resolution has a direct impact on the accuracy and motivates the use of higher resolution images to achieve a low error rate. In addition, using higher resolution images results in improved accuracy for both DAISY and NCC but DAISY consistently and significantly outperforms NCC at all resolutions. This largely stems from the fact that we compute depths at pixel resolution. DAISY is more resilient to pixel sampling errors than NCC and hence is not affected adversely by this fact.

Better NCC results could of course be obtained by taking surface orientation into account to warp the sampling grid appropriately [6]. Alternatively, as is done in [14], NCC scores could be improved by averaging different measurements for a set of surface normals to account for possible errors caused by incorrect normal estimates. However, the results depicted in Fig. 10 are nevertheless significant because they show that the DAISY descriptor is robust enough to make these time consuming optimizations unnecessary, which is what makes our approach both effective and fast. Moreover, in Sect. 5.2, we will show that our approach is as accurate as that of [6], which relies on plane sweeping and, in effect, warps the surface patches and therefore optimizes the NCC parameters.

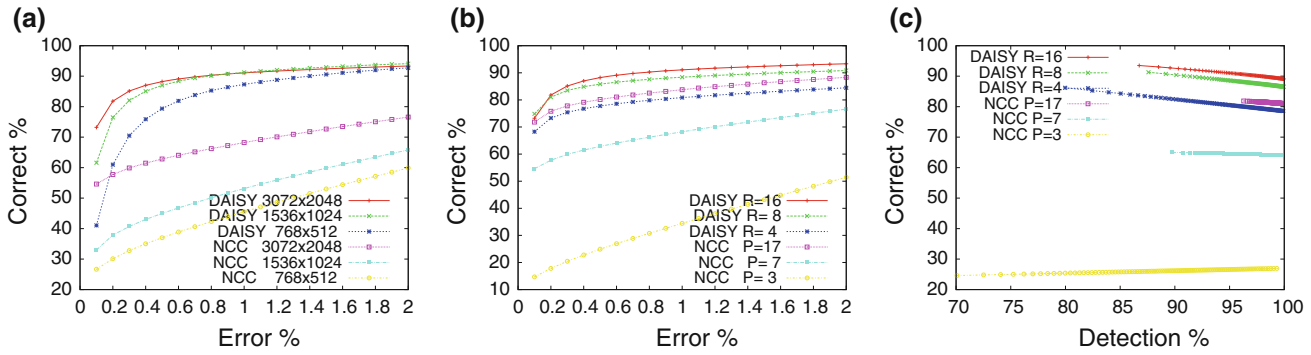

Fig. 10 Parameter choices and image resolution. We use the 3,072 × 2,048 images of the Fountain-P11 dataset. **a** Effect of the image resolution on depth map quality estimated by reducing the size of the original images. Note that higher resolution images yield more pixels with low error rates. **b** Varying the DAISY radius R and NCC patch size P computed at 100% detection rate on the original images. $R = 8$ and $P = 17$ perform best at low error rates. **c** Detection rate versus correct pixel percentage for varying descriptor radius

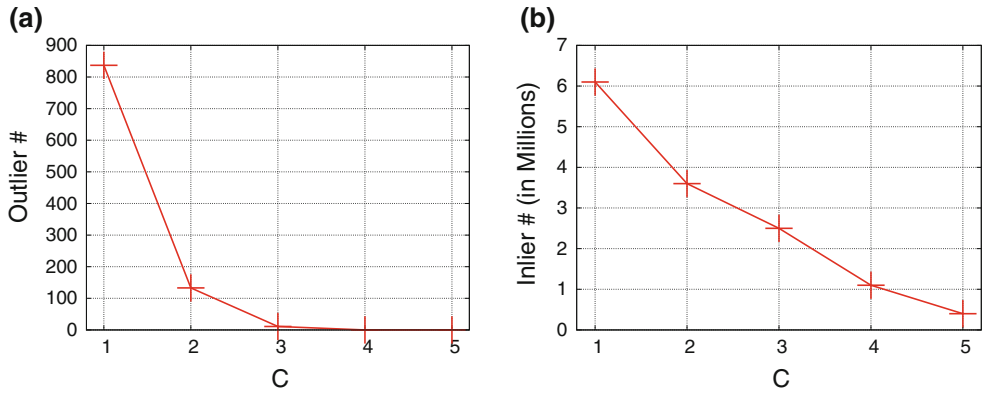

Fig. 11 Consistency threshold selection. We use the the Fountain-P11 dataset to see the effect of confidence threshold C on the quality of the point clouds. A point is assumed an inlier if its distance to the ground truth is smaller than 0.1% of the actual depth. **a** Number of outliers with respect to varying C threshold values. **b** The inlier number for the same thresholds

In Fig. 10b, c, to highlight the influence of various parameters, we vary the descriptor radius size, R and NCC patch size P. In these experiments, even though $R = 16$ produces more complete maps with higher detection rates, $R = 8$ yields more accurate depth values. Because of this, the parameters $R = 8$, $Q = 2$, $T = 4$ and $H = 4$ are used for all results presented in this paper.

In Fig. 11, we tested different values for the visibility consistency threshold, C, of Sect. 3.2. The experiments are performed on the Fountain-P11 sequence. The number of inliers and outliers for different C values are computed. A point is assumed to be an inlier if its distance to the ground truth is smaller than 0.1% of the actual depth. As can be seen from the graphs, enforcing consistency using only $C = 1$ additional depth map allows many outliers to survive. However, for $C = 2$ the number is much reduced and decays almost to zero for $C = 3$. Increasing C further, while ensuring no outliers will seep into the reconstruction, will result in less complete reconstructions as points will be forced to be visible in more images.

5.2 Comparative results

To compare our algorithm against other approaches, we used a publicly available benchmark data set [22], which contains both close-range shots of highly textured scenes and more distant views of less textured ones. Ground truth data are obtained using a laser scanner and it consists of a mean depth and its variance for each 3D point.

5.2.1 Accuracy

As indicated in Table 1, we ran our algorithm on the *Fountain-P11*, *HerzJesu-P25*, *Entry-P10*, and *Castle-P30* image sets to produce the results of Figs. 9, 12, 13 and 14. Since the benchmarking algorithm relies on comparing triangulated meshes, we used the Poisson reconstruction algorithm [12] to turn our point clouds into meshes.

Figure 15 compares our algorithm against the best methods reported in the benchmarking site [6,8,15,27]. The performance of each algorithm is represented by a cumulative

Table 1 Computation times and scene details for benchmark datasets

Data set	Image no.	Res.	Comp. time, 1 core	Comp. time, 8 core	Point no.
HerzJesu-P8	8	6	8.9	1.5	3.2
Entry-P10	10	6	9.3	1.6	1.4
Fountain-P11	11	6	12.2	2.2	2.5
Castle-P30	30	6	20.8	4.2	1.8
HerzJesu-P25	25	6	23.1	4.3	4.9

Resolutions are given in megapixels, point numbers are in millions, and computation times are in minutes

histogram of deviations of recovered depth from the ground truth. These deviations are expressed as multiples of the ground truth variance and the percentage of recovered depths that fall below this error threshold. On these moderate sized images, we perform roughly equivalently to [6], albeit much faster as will be discussed below. We are a little less accurate than the best approach [8], which was to be expected for two reasons. First, we only triangulate for benchmarking purposes and use a standard triangulation algorithm that does not enforce global consistency constraints as is done in [8]. Second, Ref. [8] includes a variational mesh refinement scheme, whereas we discretize the depth states as discussed in Sect. 3. This does not influence the algorithm adversely on the high-resolution image sets, like the ones that will be presented in the next section, for which our method is designed for, but handicaps it on lower resolution ones. So, one can deduce that to achieve the accuracy of [8] run for 6 Megapixel images, our method would need higher resolution images to reduce the discretization error inherit in our approach. However, since our method runs very efficiently for high-resolution images, this handicap for lower resolution images does not constitute a major problem as long as higher-resolution images can be acquired. In case no high-resolution images exist, Ref. [8] seems to be a better solution than ours.

In a sense, the benchmarking site compares two things that are not comparable: The raw output of the matching algorithm in our case, against a refined version in the other cases. For a more direct comparison, we therefore used the publicly available code of one of the best performing point cloud-based algorithms [6] and ran it using either 7×7 or 17×17 patches, on the Fountain and Herzjesu sequences, which consist of $3{,}072 \times 2{,}048$ images. In Fig. 16, we plot the number of correctly found points versus the allowed error threshold where a point is assumed correct if its distance to the ground truth is smaller than the error threshold. To compute point error, both the reconstructed 3D points and the ground truth are projected onto the images. The error is then taken as the smallest ratio of the depth error to the actual depth value as given in Eq. 6 across all the input images. Using the 17×17 patches result in denser clouds than using the 7×7 ones but they remain far sparser than ours for any given accuracy level.

5.2.2 Speed

Running [6] using 17×17 patch size takes 843 min for Fountain and 508 min for Herzjesu. These numbers become 338 and 204 min for 7×7 patches. When using our method with descriptor size $R = 8$ they drop to 12.2 and 8.9 min, respectively. For consistency, we used a single core in all cases. Our approach is about 40–50 times faster than [6] for parameters chosen so as to produce clouds as dense as ours. The speed difference would have been much more dramatic for large-scale sequences like the ones presented in the next section such as the Lausanne Cathedral-Ground sequence shown in Fig. 3 or the Lausanne City sequence shown in Fig. 17.

There are two main reasons for the speed difference. First, the use of NCC as a similarity measure in [6] forces an optimization over the surface normal for all the points besides an optimization over depth. This is necessary for NCC since failure to do so will result in poor performance as it is known to be very sensitive to perspective deformations. This additional optimization over the surface normal is avoided in our algorithm mainly because DAISY is quite robust to both perspective and sampling errors and thus a single optimization over depth is sufficient. Second, memory access pattern of our depth estimation framework is very efficient. In [6], authors propose a diffusion-like approach where starting from some seed points, new points are included into the cloud by iteratively searching, expanding and filtering new points near already included points. In addition, more than two images are used for measuring the photo-consistency score of a point and this may mean to load and release an image many times if the image set is large enough not to fit to memory. In our case, however, depth estimation is done only on a pair of images and the consistency check is done in the cloud computation stage. This separation of jobs is what enables us to access memory very efficiently.

We are unfortunately unable to give timing estimates for the other methods discussed above because the corresponding publications do not mention them and no source code is publicly available. However, there is little reason to believe that any of those implemented on CPUs would be any faster. Furthermore, since the DAISY descriptor relies on convolutions, it could easily be implemented on a GPU. This should yield further increase in speeds such as those reported in [14] without a need for a computer cluster as is done in that work.

In Fig. 18, we present the computation times for depth map estimation using different image resolutions and different number of cores. All the experiments are done on a 12 GB Intel Xeon 2.5 GHz Quad Core machine. When all 8 cores are used, depth maps are computed for 40, 20, 10, and 5 Megapixel images in 35, 20, 11, and 5 s with an average 2,100 depth tests per pixel for the highest resolution. From these

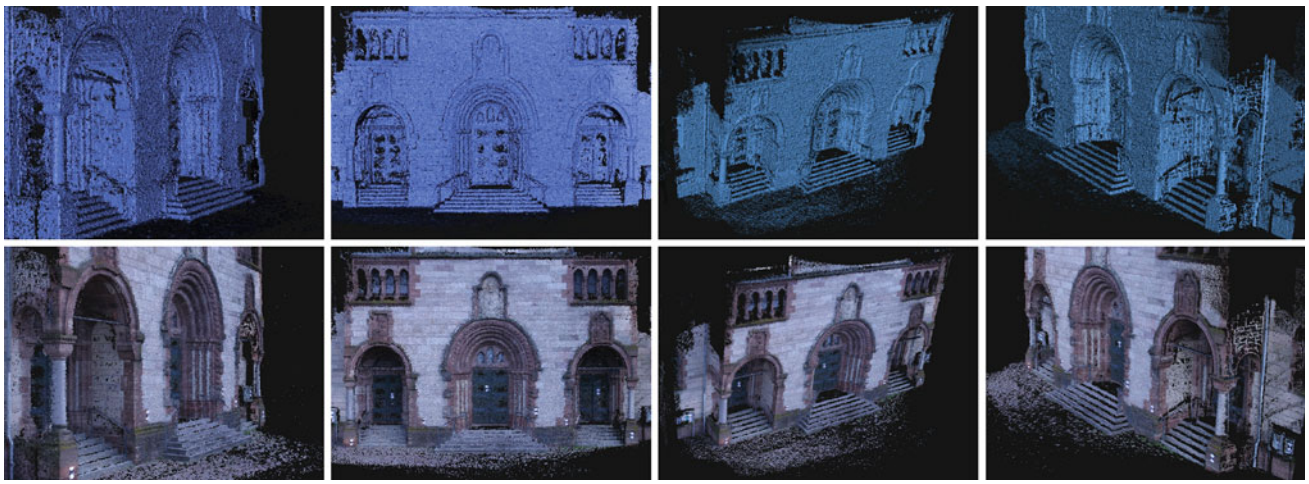

Fig. 12 HerzJesu-P25. The 3D point cloud was constructed in 4.3 min from 25 6 Megapixel images and contains 4.9 million points. *Top row images* The raw 3D points and *bottom row* ones show their colorized renderings (color figure online)

Fig. 13 Entry-P10 The 3D point cloud was constructed in 1.6 min from 10 6 Megapixel images and contains 1.4 million points. The *left images* The raw points and the *right images* The colorized versions of these points (color figure online)

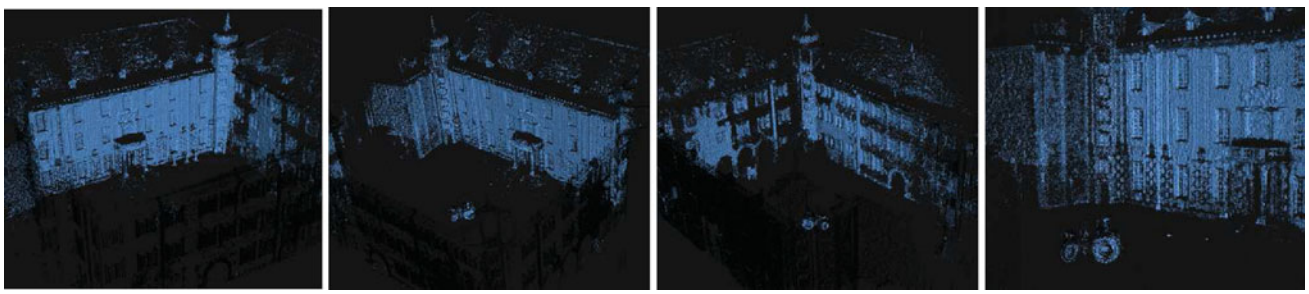

Fig. 14 Castle-P30. The 3D point cloud was constructed in 4 min from 30 6 Megapixel relatively low textured images and contains 1.8 million points. We show the raw 3D points as seen from several viewpoints

experiments, we see that doubling the core number, roughly halves the computation time. Naturally, computation time can be further reduced using larger number of cores.

Note that running our approach on all eight cores for the whole pipeline is on average 6 times faster than running on a single one as seen in Table 1, even when using only simple OpenMP preprocessor instructions for for-loop parallelizations, thus indicating that our method is inherently parallel and could easily be made to run even faster on a cluster if additional speed was required. Albeit minor, there is a speed loss compared to depth map computation stage for parallel operation and this is mostly due to disk read/write operations when storing intermediate results.

5.3 Handling large data sets

We now present the results on large scale sets of very high resolution images. We used the algorithm outlined in [21] to register these images. For each set, we render the final point cloud, which is computed by combining individual depth maps as explained in Sect. 3.2, from different view points. The points are shaded according to their estimated normals

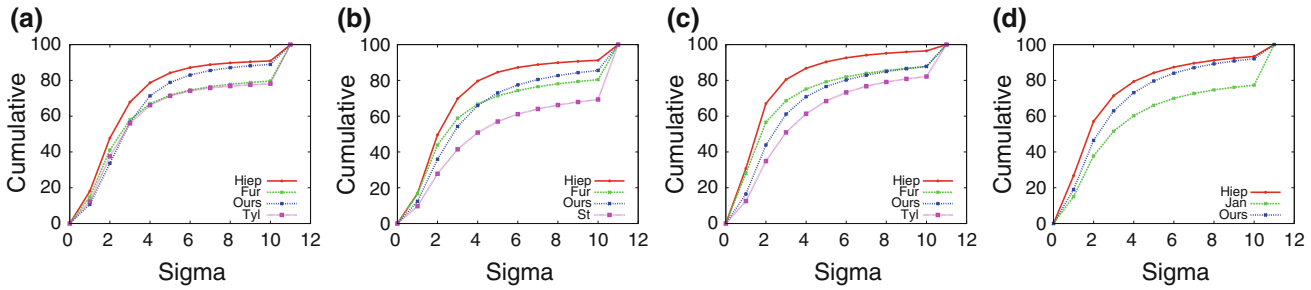

Fig. 15 Comparing against state-of-the-art. We compare our reconstructions to those of other methods reported in [22] on **a** Fountain-P11, **b** HerzJesu-P25, **c** Entry-P10 and **d** Castle-P30. We plot the cumulative error histograms and the legend is Hiep [8], Fur [6], Tyl [27], St [19] and Jan [11]. Our own method is labeled as "Ours". On these moderate-resolution images, our accuracy is comparable to that of [6] and a little worse than that of [8], which can be attributed to the fact that, unlike [8], we do *not* refine the point clouds

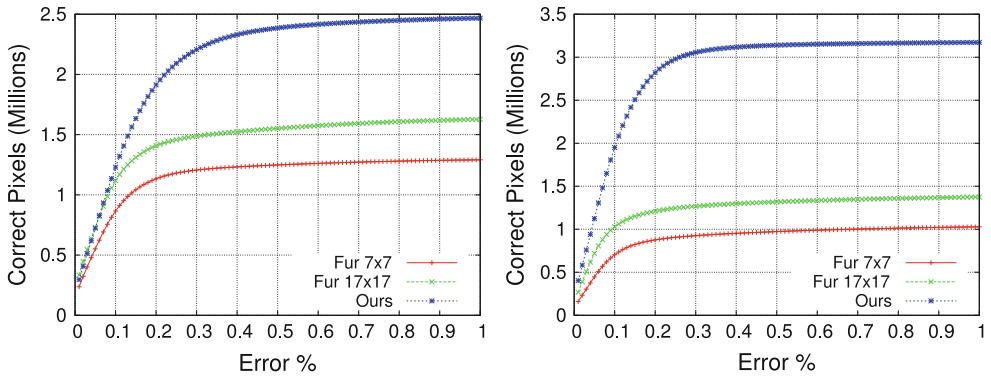

Fig. 16 Comparison against [6]. For Fountain-P11 on the *left* and HerzJesu-P8 on the *right*, we plot the total number of reconstructed pixels whose depth is within a given distance of the ground-truth value as function of that distance. Note that for any such distance, our method produces many more points and therefore far denser clouds. Furthermore, it is also faster since, when using a single core, it only requires 12.2 min for Fountain and 8.9 min for Herzjesu, whereas [6] using 17×17 patches, requires 843 and 508 min, respectively

as explained in Sect. 3. We also show colorized point clouds, where the color of each point is assigned from the image where the point is initialized from. For some data sets, we also show the depth maps computed from selected image pairs in inverse depth representation. The red color denotes the pixels for which the algorithm decides that there is no good match.

Each set was processed on a 12 GB Intel Quad Core Xeon 2.5 GHz machine and computation times and input characteristics are listed in Table 2. For more extensive, detailed and animated results, please visit our website [24].

Figure 2 depicts the *EPFL Campus* dataset. It consists of 40 Megapixel images taken from a helicopter, which represents the highest resolution we tested our algorithm on. The campus is fully reconstructed including trees, grass walkways, parked cars, and train tracks. The only exceptions are some building façades that were not seen from any viewpoint.

Next, in Figs. 1 and 8, we tested our algorithm on two datasets, *Statue* and *Pillar*, that contain images of very different scales. The viewpoint selection algorithm successfully paired up images that are similar in scale and thanks to the *precision measure* of Eq. 5, only the highest-precision 3D points are retained.

Table 2 Computation times and scene details for large scale datasets

Data set	Image no.	Resolution	CPU time	Point no.
Portal	13	21	9.2	20.3
EPFL Lausanne Cathedral-Aerial	61	24	22.1	12.7
Statue	127	18	29.5	15.3
Pillar	214	18	48.9	63.1
Lausanne Cathedral-Ground	1,302	21	419	148.2
Lausanne City	4,484	6–21	1,632	272

Resolutions are given in megapixels, point numbers are in millions, and computation times are in minutes using 8 cores

Figures 3 and 6 show the reconstructions of the *Lausanne Cathedral* computed using ground-level and aerial images, respectively. The ground-level reconstruction is the more accurate but some of the roofs are missing because they were not visible in any of the images. By contrast, the aerial

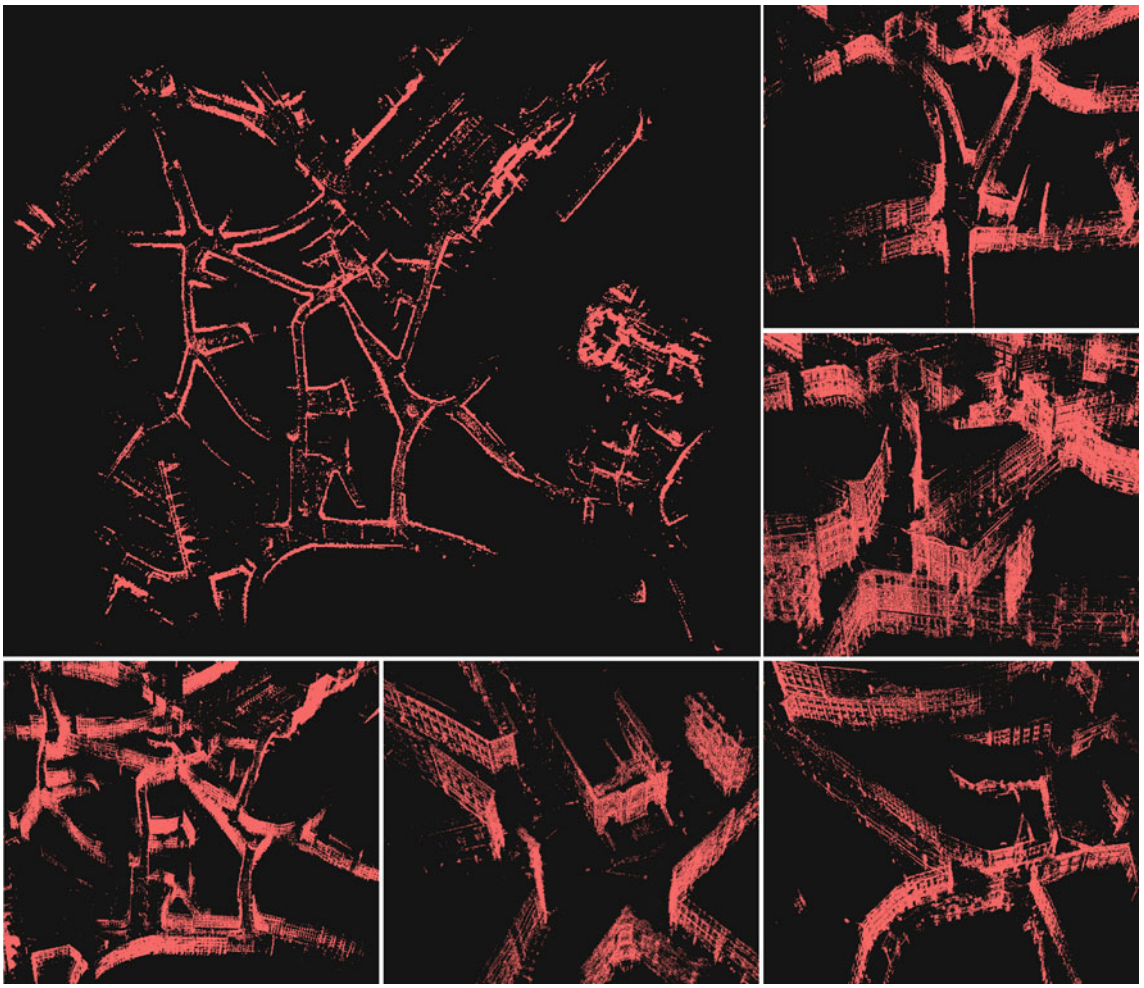

Fig. 17 City scale reconstruction. The 3D point cloud was constructed from 4,484 6 to 21 Megapixel ground-level images in 27 h and contains 272 million points. For lack of a machine able to display the whole cloud, this figure depicts a decimated version of it. The *top left figure* is a top view that shows the extent of the reconstruction and the others are close-up views seen from different perspectives

reconstruction is more complete, even though fewer images were used, because almost all the input images contain the full façade of the cathedral as can be seen in the first row of Fig. 6. It is less accurate because the images were taken from further away. In addition, in Fig. 7, we reconstructed the main portal of the cathedral using a smaller subset of the ground-level data set. In this case, to increase the density of the computed cloud, we computed two depth maps per image of the data set by pairing it up with the closest two cameras.

Finally, in Fig. 17, we present results on the the *Lausanne City* sequence [21], which is the largest data set we tested our algorithm on. It contains 3,504 6-Megapixel images and 980 21-Megapixel images of the downtown area of Lausanne, seen at different scales. There is much clutter, such as people and cars, and some images were taken at different times of day. In addition, since the cameras are distributed more or less uniformly across the whole area, instead of being clustered at a few landmark locations, there is sometimes only a small overlap between them and only a few images see the same location. This means that, unlike in the case of community photo collection datasets [3,7], we cannot rely on the same place being imaged many times over. Instead, our MVS has to operate effectively even when the scene is only sparsely covered, which it does. It took 1,632 min to compute a cloud containing 272 million points. This may seem long but represents only 27 h or a little over 1 day on a single PC, as opposed to a cluster and without using GPU processing. In other words, this remains manageable on an ordinary computer even though the dataset involved is quite large.

6 Conclusion

In this paper, we presented a novel multi-view stereo algorithm that can handle large-scale very high resolution images at relatively very low computational costs. In contrast to many state-of-the-art methods that use moderate sized

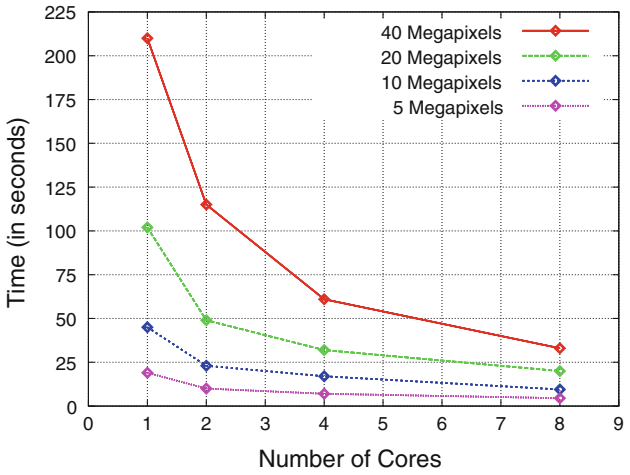

Fig. 18 Depth map computation times. Computation times for depth maps of different resolution images with respect to different core numbers. The experiments are performed on a 12 GB Intel Xeon 2.5 GHz Quad Core machine. Using 8 cores, we can compute depth maps for 40, 20, 10 and 5 Megapixel images in 35, 20, 11 and 5 s, respectively. In the experiments, on average 2,100 depth states are tested per pixel for the highest resolution. Computation times decrease almost linearly with the number of the cores, i.e., increasing the core number twice its size, reduces the computation time by half

images, we advocate the use of larger resolution ones. We showed that the rich texture of such images removes the need for expensive optimization algorithms and makes it possible to obtain very dense high-quality 3D point clouds using a computationally very simple scheme. This in part is due to the DAISY descriptor being more reliable than normalized cross correlation, making the kind of plane sweeping strategies used in many competing approaches unnecessary.

We validated our approach on various large-scale high-resolution data sets acquired under different conditions such as from a helicopter, from an airplane, or from the ground with different view point densities and scale changes, also compared our results to that of state-of-the-art methods on benchmark data sets and showed that we produce reconstructions of similar accuracy but much faster.

In short, using a robust metric for matching purposes, one can bypass the complex processing other algorithms have to use and reduce computational load immensely. This makes for a very efficient algorithm to process ultra high-resolution images in minutes on a standard desktop machine as evidenced by the experiments we have presented.

The accuracy of the proposed approach could be further improved by performing sub-pixel sampling of the epipolar line for depth estimation but this would slow down the algorithm by allocating CPU time unnecessarily across the whole disparity range. A better approach would be a multi-resolution sampling of the range by moving in smaller steps for plausible depth values and in larger steps otherwise. Using a descriptor for this type of algorithm has advantages over intensity-based measures where a descriptor would be more robust to errors in translation or sampling.

Additional speed improvements can be achieved by constraining depth ranges per pixel instead of per image as is done in this work. This could be accomplished easily using an initial rough mesh computed from the calibration points. Also, preventing unnecessary computation for low contrast regions such as sky or saturated areas will definitely help.

Another natural extension of this work would be the computation of a triangulated mesh from the reconstructed point cloud that takes visibility into account, both for compact representation and to eliminate the few remaining outliers. Finally, since we believe that the output of our algorithm could serve as input to point cloud refinement techniques such as [8], we make our software available from our website [24].

Appendix: Uniform step sampling

Given two camera calibrations, we present a closed form method to compute a sequence of 3D points such that they all project to a single location on one camera and that their projection forms a uniformly sampled line on the other camera.

Let a camera be parametrized by its intrinsic parameters $\mathbf{K}$, and extrinsic parameters of rotation matrix $\mathbf{R}$ and camera center $\mathbf{C}$. The projection of a 3D point $\mathbf{X}$ is defined as

$$\lambda \mathbf{x} = \mathbf{K}\mathbf{R}(\mathbf{X} - \mathbf{C}) \tag{7}$$

with $\mathbf{x} = [x\ y\ 1]^{\mathrm{T}}$ its image coordinates and λ its depth. Then, the back projected ray emanating from point $\mathbf{x} = [x\ y\ 1]^{\mathrm{T}}$ is parametrized in terms of the depth variable λ as:

$$\mathbf{X}(\lambda) = \lambda \mathbf{R}^T \mathbf{K}^{-1}\mathbf{x} + \mathbf{C}. \tag{8}$$

Given two cameras $P_0 = (\mathbf{K}_0, \mathbf{R}_0, \mathbf{C}_0)$ and $P_1 = (\mathbf{K}_1, \mathbf{R}_1, \mathbf{C}_1)$, we would like to sample the back projected line $\mathbf{X}(\lambda)$ so that the projected samples on camera P_1 are uniformly separated (see Fig. 19). The two points separated by depth $d\lambda$ on $\mathbf{X}(\lambda)$ is equal to:

$$\begin{aligned} \mathbf{X}(\lambda) &= \lambda \mathbf{R}_0^T \mathbf{K}_0^{-1}\mathbf{x} + \mathbf{C}_0 \\ \mathbf{X}(\lambda + \mathrm{d}\lambda) &= (\lambda + \mathrm{d}\lambda)\mathbf{R}_0^T \mathbf{K}_0^{-1}\mathbf{x} + \mathbf{C}_0. \end{aligned} \tag{9}$$

Then, the projection of these points on image P_1 are

$$\omega \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \lambda \mathbf{a} + \mathbf{b}, \tag{10}$$

$$(\omega + \mathrm{d}\omega) \begin{bmatrix} u + r\mathrm{d}u \\ v + r\mathrm{d}v \\ 1 \end{bmatrix} = (\lambda + \mathrm{d}\lambda)\mathbf{a} + \mathbf{b}, \tag{11}$$

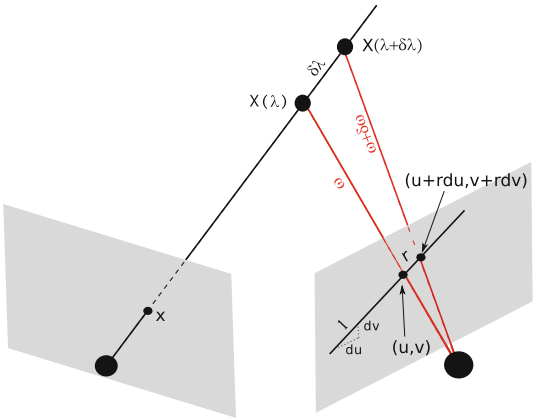

Fig. 19 Uniform step sampling framework

with $\mathbf{a} = \mathbf{K}_1\mathbf{R}_1\mathbf{R}_0^T\mathbf{K}_0^{-1}\mathbf{x}$ and $\mathbf{b} = \mathbf{K}_1\mathbf{R}_1(\mathbf{C}_0 - \mathbf{C}_1)$. In the equations above, (u, v) and ω are the coordinates and depth of $\mathbf{X}(\lambda)$ on image P_1, respectively, $w + \mathrm{d}\omega$ the depth of the point $\mathbf{X}(\lambda + \mathrm{d}\lambda)$, $(\mathrm{d}u, \mathrm{d}v)$ the slope of the epipolar line and r is the sampling resolution.

The slope of the epipolar line can be found by projecting any two points $\mathbf{X}(\lambda_{\min})$ and $\mathbf{X}(\lambda_{\max})$ onto P_1. Let the coordinates of the projections to be $(u_{\min}, v_{\min})$ and $(u_{\max}, v_{\max})$. The slope is then computed as:

$$\begin{aligned} \mathrm{d}l &= \sqrt{(u_{\max} - u_{\min})^2 + (v_{\max} - v_{\min})^2} \\ \mathrm{d}u &= (u_{\max} - u_{\min})/\mathrm{d}l \\ \mathrm{d}v &= (v_{\max} - v_{\min})/\mathrm{d}l. \end{aligned} \tag{12}$$

Rearranging Eqs. 10 and 11, one can see the relationship between the changes in the depths for two views as:

$$\mathbf{a}\mathrm{d}\lambda = \omega \begin{bmatrix} r\mathrm{d}u \\ r\mathrm{d}v \\ 0 \end{bmatrix} + \mathrm{d}\omega \begin{bmatrix} u + r\mathrm{d}u \\ v + r\mathrm{d}v \\ 1 \end{bmatrix}. \tag{13}$$

If we denote the ith element of vector $\mathbf{a}$ with a_i and expand the third row of the Eq. 13 as $a_2\mathrm{d}\lambda = \mathrm{d}\omega$, the first two rows of Eq. 13 becomes

$$\mathrm{d}\lambda = \frac{\omega r\mathrm{d}u}{a_0 - a_2(u + r\mathrm{d}u)} = \frac{\omega r\mathrm{d}v}{a_1 - a_2(v + r\mathrm{d}v)}. \tag{14}$$

Equation 14 is the update in the depth with respect to camera P_0 that one must make in order to move r pixels in the direction of $(\mathrm{d}u, \mathrm{d}v)$ on camera P_1 where (u, v) and ω is the current projection and current depth with respect to camera P_1 (see Fig. 19). By iterating this process, the sequence of 3D points with a uniform sampling projection can be computed. Algorithm 1 presents the pseudo-code for this closed form solution.

Algorithm 1: Non-uniform Space Sampling

Require: Camera Parameters: $(\mathbf{K}_0, \mathbf{R}_0, \mathbf{C}_0)$ and $(\mathbf{K}_1, \mathbf{R}_1, \mathbf{C}_1)$
Require: Point location $\mathbf{x}$
Require: Depth range $(\lambda_{min}, \lambda_{dmax})$
Require: Sampling resolution r

$\mathbf{a} \Leftarrow \mathbf{K}_1\mathbf{R}_1\mathbf{R}_0^T\mathbf{K}_0^{-1}\mathbf{x}$
$\mathbf{b} \Leftarrow \mathbf{K}_1\mathbf{R}_1(\mathbf{C}_0 - \mathbf{C}_1)$
$u_{min} \Leftarrow \frac{\lambda_{min}a_0+b_0}{\lambda_{min}a_2+b_2}$, $v_{min} \Leftarrow \frac{\lambda_{min}a_1+b_1}{\lambda_{min}a_2+b_2}$
$u_{max} \Leftarrow \frac{\lambda_{max}a_0+b_0}{\lambda_{max}a_2+b_2}$, $v_{max} \Leftarrow \frac{\lambda_{max}a_1+b_1}{\lambda_{max}a_2+b_2}$
$dl \Leftarrow \sqrt{(u_{max} - u_{min})^2 + (v_{max} - v_{min})^2}$
$du \Leftarrow (u_{max} - u_{min})/dl$, $dv \Leftarrow (v_{max} - v_{min})/dl$
$\omega_{min} \Leftarrow \lambda_{min}a_2 + b_2$

$\omega \Leftarrow \omega_{min}$
$\lambda \Leftarrow \lambda_{min}$
$(u, v) =\Leftarrow (u_{min}, v_{min})$
while $\lambda < \lambda_{max}$ **do**
  if $|du|$ not equal to 0 **then**
    $d\lambda \leftarrow \dfrac{\omega r du}{a_0 - a_2(u + r du)}$
  else
    $d\lambda \leftarrow \dfrac{\omega r dv}{a_1 - a_2(v + r dv)}$
  end if
  $\lambda \leftarrow \lambda + d\lambda$
  $(u, v) \leftarrow (u + r du, v + r dv)$
  $X \leftarrow \lambda \mathbf{R}^T\mathbf{K}^{-1}\mathbf{x} + \mathbf{C}$
  $\omega \leftarrow \lambda a_2 + b_2$
end while

References

1. Alvarez, L., Deriche, R., Weickert, J., Sanchez, J.: Dense disparity map estimation respecting image discontinuities: a PDE and scale-space based approach. J. Mach. Learning Res. **13**(1/2), 3–21 (2002)
2. Bay, H., Tuytelaars, T., Van Gool, L.: SURF: speeded up robust features. In: European Conference on Computer Vision (2006)
3. Frahm, J.-M., Georgel, P., Gallup, D., Johnson, T., Raguram, R., Jen, Y.-H., Wu, C., Dunn, E., Clipp, B., Lazebnik, S., Pollefeys, M.: Building Rome on a Cloudless Day. In: European Conference on Computer Vision (2010)
4. Fua, P.: A parallel stereo algorithm that produces dense depth maps and preserves image features. Mach. Vis. Appl. Winter **6**(1), 35–49 (1993)
5. Fua, P.: From multiple stereo views to multiple 3D surfaces. Comput. Vis. Image Underst. **24**(1), 19–35 (1997)
6. Furukawa, Y., Ponce, J.: Accurate, dense, and robust multi-view stereopsis. IEEE Trans. Pattern Anal. Mach. Intel. **99** (preprint, 2009)
7. Goesele, M., Snavely, N., Curless, B., Hoppe, H., Seitz, S.M.: Multi-view stereo for community photo collections, pp. 1–8. In: International Conference on Computer Vision (2007)
8. Hiep, V.H., Keriven, R., Labatut, P., Pons, J.-P.: Towards high-resolution large-scale multi-view stereo, pp. 1430–1437. In: Conference on Computer Vision and Pattern Recognition (2009)
9. Hirschmüller, H.: Stereo processing by semi-global matching and mutual information. IEEE Trans. Pattern Anal. Mach. Intell. **30**, 328–341 (2008)
10. Hornung, A., Hornung, E., Kobbelt L.: Hierarchical volumetric multi-view stereo reconstruction of manifold surfaces based on

dual graph embedding, pp. 503–510. In: Conference on Computer Vision and Pattern Recognition (2006)
11. Jancosek, M., Shekhovtsov, A., Pajdla, T.: Scalable multi-view stereo. In: 3DIM (2009)
12. Kazhdan, M., Bolitho, M., Hoppe H.: Poisson surface reconstruction, pp. 61–70. In: SGP '06: Proceedings of the fourth Eurographics Symposium on Geometry processing (2006)
13. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. Int. J. Comput. Vis. **20**(2), 91–110 (2004)
14. Pollefeys, M., Nistéer, D., Frahm, J.-M., Akbarzadeh, A., Mordohai, P., Clipp, B., Engels, C., Gallup, D., Kim, S.-J., Merrell, P., Salmi, C., Sinha, S., Talton, B., Wang, L., Yang, Q., Stewéenius, H., Yang, R., Welch, G., Towles, H.: Detailed real-time urban 3D reconstruction from video. Int. J. Comput. Vis. **78**(2–3), 143–167 (2008). doi:10.007/s11263-007-0086-4
15. Salman, N., Yvinec, M.: Surface reconstruction from multi-view stereo. In: Asian Conference on Computer Vision (2009)
16. Seitz, S.M., Curless, B., Diebel, J., Scharstein, D., Szeliski, R.: A comparison and evaluation of multi-view stereo reconstruction algorithms, pp. 519–528. In: Conference on Computer Vision and Pattern Recognition (2006)
17. Starck, J., Miller, G., Hilton, A.: Volumetric stereo with silhouette and feature constraints. In: British Machine Vision Conference (September 2006)
18. Strecha, C.: Multi-view evaluation (2008). http://cvlab.epfl.ch/data
19. Strecha, C., Fransens, R., Van Gool L.: Wide-baseline stereo from multiple views: a probabilistic account. In: Conference on Computer Vision and Pattern Recognition (2004)
20. Strecha, C., Fransens, R., Van Gool, L.: Combined depth and outlier estimation in multi-view stereo. In: Conference on Computer Vision and Pattern Recognition (2006)
21. Strecha, C., Pylvanainen, T., Fua, P.: Dynamic and scalable large scale image reconstruction. In: Conference on Computer Vision and Pattern Recognition (June 2010)
22. Strecha, C., von Hansen W., Van Gool, L., Fua, P., Thoennessen, U.: On Benchmarking camera calibration and multi-view stereo for high resolution imagery. In: Conference on Computer Vision and Pattern Recognition (2008)
23. Sun, J., Li, Y., Kang, S.B., Shum, H.-Y.: Symmetric stereo matching for occlusion handling, pp. 399–406. In: Conference on Computer Vision and Pattern Recognition (2005)
24. Tola, E.: Efficient large scale multiview stereo for ultra high resolution images (2011). http://www.engintola.com/research/emvs
25. Tola, E., Lepetit, V., Fua, P.: Daisy: an efficient dense descriptor applied to wide baseline stereo. IEEE Trans. Pattern Anal. Mach. Intell. **32**(5), 815–830 (2010)
26. Tran, S., Davis, L.: 3D surface reconstruction using graph cuts with surface constraints, pp. 219–231. In: European Conference on Computer Vision (2006)
27. Tylecek, R., Sara, R.: Depth map fusion with camera position refinement. In: CVWW (2009)
28. Vogiatzis, G., Hernandez Esteban, C., Torr, P.H.S., Cipolla, R.: Multiview stereo via volumetric graph-cuts and occlusion robust photo-consistency. IEEE Trans. Pattern Anal. Mach. Intell. **29**, 2241–2246 (2007)

Author Biography

Engin Tola received both his Bachelor and Master of Science degrees in Electrical and Electronics Engineering from the Middle East Technical University (METU), Turkey, in 2003 and 2005, respectively. Afterwards he joined the computer vision laboratory of EPFL (Swiss Frederal Institute of Technology) and gained his PhD degree in Computer Science in 2010. In 2011, he co-founded Aurvis to continue his research in developing efficient 3D reconstruction techniques and Argutek for mobile augmented reality applications.