Machine Vision and Applications
DOI 10.1007/s00138-009-0209-8

ORIGINAL PAPER

Fast and automatic object pose estimation for range images on the GPU

In Kyu Park · Marcel Germann · Michael D. Breitenstein · Hanspeter Pfister

Received: 30 December 2008 / Revised: 20 April 2009 / Accepted: 23 May 2009

Abstract We present a pose estimation method for rigid objects from single range images. Using 3D models of the objects, many pose hypotheses are compared in a data-parallel version of the downhill simplex algorithm with an image-based error function. The pose hypothesis with the lowest error value yields the pose estimation (location and orientation), which is refined using ICP. The algorithm is designed especially for implementation on the GPU. It is completely automatic, fast, robust to occlusion and cluttered scenes, and scales with the number of different object types. We apply the system to bin picking, and evaluate it on cluttered scenes. Comprehensive experiments on challenging synthetic and real-world data demonstrate the effectiveness of our method.

Keywords Object pose estimation · Bin picking · Range image processing · General purpose GPU programming · Iterative closest point · Euclidean distance transform · Downhill simplex · CUDA

I. K. Park (✉)
School of Information and Communication Engineering,
Inha University, Incheon 402-751, Korea
e-mail: pik@inha.ac.kr

M. Germann
Computer Graphics Lab., Swiss Federal Institute
of Technology (ETH), 8092 Zurich, Switzerland

M. D. Breitenstein
Computer Vision Lab., Swiss Federal Institute
of Technology (ETH), 8092 Zurich, Switzerland

H. Pfister
School of Engineering and Applied Sciences,
Harvard University, Cambridge, MA 02138, USA

1 Introduction

The estimation of the 3D orientation and location (i.e., pose) of an object is a fundamental task in computer vision. Often, pose estimation is used during runtime (e.g., for robot navigation or human–computer interaction) or as a preprocessing step for other algorithms (e.g., for object recognition or tracking). However, pose estimation of objects in complex and cluttered scenes is challenging because the appearance of an object in a 2D image is sensitive to illumination, shadows, and lack of visual features. In addition, the objects may partially occlude each other. Some of these problems can be avoided using *depth information*. Since recent depth acquisition systems have reached a high level of reliability, the use of range images, i.e., images with per-pixel depth, to overcome some of these problems is promising.

We present an algorithm for model-based pose estimation from single range images that is based on comparing many pose hypotheses in parallel. It is completely automatic, does not require an initial guess, and consistently finds the global optimum. The algorithm has been developed to exploit the computational power of modern graphics hardware and runs entirely on the GPU. The GPU consists of many small processing units that are able to run many simple programs in parallel. Because not every algorithm is easily parallelizable, it is beneficial to specifically design the algorithm for implementation on the GPU.

In a pre-processing step, we build a database of *reference range images* by rendering a 3D CAD model of the object with many different orientations. A key question we are addressing in this paper is how these reference poses are optimally selected. During runtime, we automatically identify a large visible object close to the camera in the input range image. We then compare the alignment of every reference range image to the input range image in parallel. The

Published online: 05 August 2009

comparison is based on a novel *image-based error function* that can be evaluated very quickly. We adjust the translation of the reference range images by *iterative energy minimization* using a parallelized version of the downhill simplex algorithm. The pose hypothesis with the smallest error yields a rough pose estimation that is then being refined using ICP.

The system presented in this paper is a substantial extension of our previous work [11]. We improved the algorithm robustness by enhancing the initial object localization method so that the performance is independent of the complexity of the input scene. The error function has been modified to take the uncertainty of pixels into account and to give more preference for poses with larger visible area. Furthermore, we developed a novel adaptive pose sampling method to automatically build an optimal, yet compact reference pose database. Finally, we added ICP to refine the initial pose estimation, thereby increasing robustness and estimation accuracy. Also, to achieve greater speedup and code portability, we implemented the system using NVIDIA's Compute Unified Device Architecture (CUDA) [24] instead of the OpenGL Shading Language (GLSL). We apply our pose estimation algorithm to bin-picking, which is an important problem in machine vision and robotics.

Our system is reliable, runs completely automatic, and has a framerate of 3 fps. The runtime does not depend on the complexity of the object or scene, but only on the resolution of the reference images, thus on the desired estimation accuracy. In contrast to other methods (e.g. [2,5,29]), our method works also for partially occluded and complex objects (i.e., not consisting of planar surfaces) and for cluttered scenes. Different objects can be detected at the same time by increasing the number of objects in the reference database. In experiments, we demonstrate the effect of the adaptive reference pose database and the increase in accuracy from the pose refinement stage using ICP. We present extensive evaluations of the estimation error and the runtime for real-world and synthetic data. Finally, the influence of the algorithm parameters is discussed.

The contributions of this paper include (1) a framework for model-based pose estimation based on aligning and comparing many pose hypotheses in parallel, (2) an adaptive sampling method of the pose space to build the reference pose database, (3) a novel image-based error function to compare the alignment of two range images, and (4) an efficient data-parallel implementation of the algorithm on the GPU. Our method has specifically been designed to exploit the tremendous data-parallel processing performance of modern graphics cards, yielding a 30× speedup in contrast to a comparable CPU implementation.

2 Related work

2.1 Pose estimation

The main challenge in object pose estimation is invariance to partial occlusions, cluttered scenes, and large pose variations. Approaches based on 2D images and video generally do not overcome these problems due to their dependency on appearance and sensitivity to illumination, shadows, and scale. Among the most successful attempts are methods based on global appearance [17], and methods based on local (2D) features [31,34]. However, these methods usually require a large number of labeled training examples.

Recently, model-based surface matching techniques using 3D models have become popular due to decreasing costs of range scanners. The most popular method for aligning 3D models and range images is the ICP algorithm [3,7] and its variations (e.g. [9,26,32]). However, ICP is a pose refinement method that requires a sufficiently good initial pose estimation to avoid local minima [33]. To compute an initialization for ICP, e.g., Shang et al. [35] propose to use the Bounded Hough Transform (BHT) [13]. Similarly, we compute a coarse pose estimation that is then refined using ICP. Unlike other methods (e.g., [35]), our method does not rely on pose tracking over multiple frames, which may suffer from drift or jitter and need a restart after complete occlusion of the field of view.

A large class of methods use deformable (morphable) 3D models. Usually, these methods aim at minimizing a cost term such that projections of the model match the input images (e.g. [4,16]). This requires the optimization of many parameters and an initial pose guess or manual interaction, which is inefficient. To establish multi-view correspondence and object pose, geometric hashing [18] is an efficient method. However, the matching process is sensitive to image resolution and surface sampling.

Another approach is to match 3D features (or shape descriptors, respectively) to range images. Dorai et al. [8] use principal curvature features. This requires the surface to be smooth and twice differentiable, making the algorithm sensitive to noise. Spin-image surface signatures introduced by Johnson et al. [15] yield good results for 3D registration with cluttered scenes and occluded objects. However, the method is sensitive to image resolution and might lead to ambiguous matches. Mian et al. [22] build a multidimensional table representation (referred to as tensors) from multiple unordered range images. They use a hash-table voting scheme to match the tensor to objects in a scene. Compared to spin-images they report a higher success rate, but the method requires high-resolution geometry and has a runtime of several minutes. Similar to our approach, Greenspan [12] pre-computes

model range maps, but the computation time depends on the object size.

In Breitenstein et al. [6] we adopted our previous pose estimation method [11] to the specific task of pose estimation of faces. In contrast to the work presented here, the face pose estimation algorithm does not use adaptive sampling of the pose space, relies on shape signatures, and does not use ICP for the final pose estimation. The error function and optimization framework are different to overcome the variations of non-rigid faces. The algorithm presented in this paper is more general in handling different types of rigid objects and complex scenes.

Lately, Liebelt et al. [20] presented an algorithm for object class detection and rough pose estimation based on a similar approach. They render 3D CAD models of objects with different poses to 2D images, and compare the input image to a codebook of features extracted from this database. However, their method does not cover the full pose search space and the pose estimation accuracy is low compared to our method.

2.2 Bin picking

Bin picking is an important application of object pose estimation in machine vision and robotics. For this task, a robot arm needs to grasp previously known objects from a bin. One of the main issues is to automatically detect and localize an object and reliably estimate its pose. Most previous methods for bin-picking are limited by the type of objects or the complexity of the scene they can deal with.

Ikeuchi [14] proposed an interpretation tree of selected views to represent a CAD model. He uses visual edges, surface normals and a depth map to find dominant visible surfaces. However, a reliable interpretation tree does not exist for many types of complex objects without large planar surfaces. Similarly, Rahardja and Kosaka [29] rely on simple features which limit the application to simple, planar objects. Boughorbel et al. [5] proposed a system using laser and video cameras to reconstruct the 3D scene in a bin. The range image was then used to fit an object with super-quadric shapes. However, the system is limited to simple shapes and scenes.

Some algorithms are specifically designed for the application of a surface-adapting vacuum gripper (e.g. [1,2]). However, the gripper requires objects with large planar surfaces, and the algorithms are not designed to handle arbitrarily shaped objects.

2.3 General-purpose GPU processing

Traditionally, GPUs were designed to specifically accelerate and enhance computer graphics algorithms. However, there has been a substantial amount of work to apply the processing power of GPUs to computer vision and image processing applications (see, e.g. [10]). The main drawback of using graphics hardware for non-graphics applications are the lack of efficient scatter operations, where one process sends data to other processes. Therefore, it is often not possible to directly port CPU-based algorithms to the GPU, therefore algorithms have to be designed specifically for implementation on the GPU.

Recently, the release of the NVIDIA's CUDA development platform [24] for general purpose GPU computing attracted great attention. CUDA allows the user to dynamically control the scheduling of threads and memory assignment, which is especially important for image processing algorithms. To the best of our knowledge, our work on pose estimation using GPGPU is the first to this date.

3 Overview of the system

Our algorithm consists of three stages: an offline construction of the reference pose database, a rough pose estimation, and a pose refinement (see Fig. 1 for an overview). We assume that the objects whose locations and poses we want to estimate are rigid and that their 3D CAD models are available. We give an overview of the different stages and describe them in detail in the following sections.

3.1 Adaptive reference pose database construction

A *pose hypothesis* consists of a *reference range image* that is created by rendering a 3D CAD model of the object with a corresponding orientation. We build a database of reference range images for each object once in an offline process. Instead of sampling the pose space uniformly (as in our previous method [11]), we sample the pose space more densely for poses that are more probable (see Sect. 4). This allows to estimate typical object poses more accurately.

3.2 Rough pose estimation

First, the coarse location of the object in the input range image is estimated based on depth information and the Euclidean distance transform of the 2D image (see Sect. 6). Starting from this location, the translation parameters for each reference pose image are iteratively refined by an energy minimization procedure based on a parallelized version of the downhill simplex algorithm [23]. A novel error function (see Sect. 6.2) compares the alignment of each reference pose image to the input range image. This is done for all pose hypotheses in parallel on the GPU. Finally, the pose hypothesis with least alignment error is selected as the rough pose estimation. Its error value serves as a confidence measure, indicating how good the alignment is.

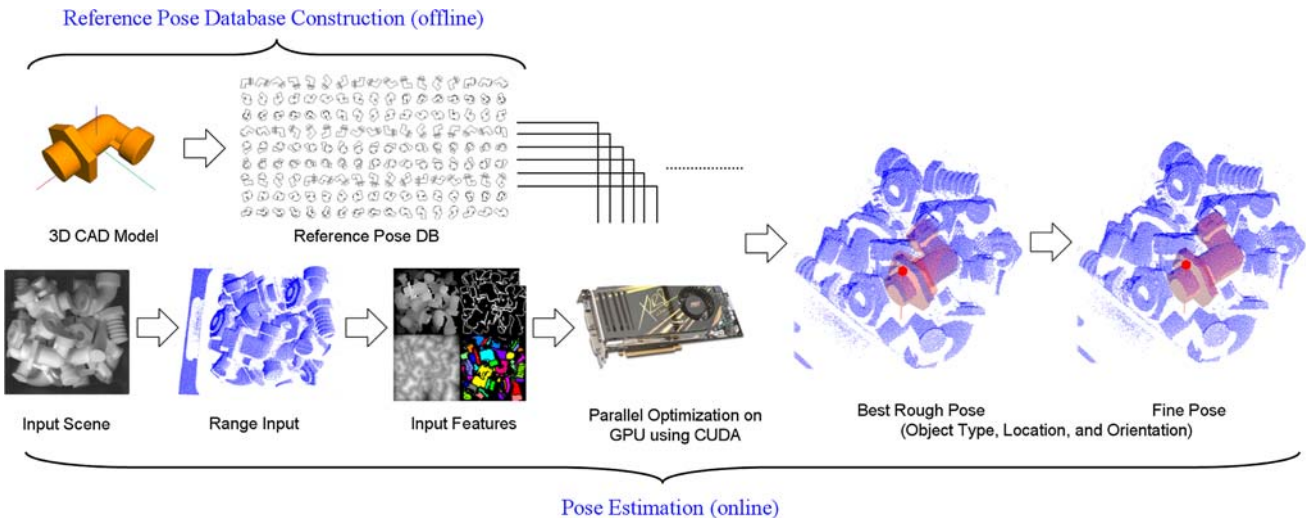

Fig. 1 Overview of our system

3.3 Fine pose estimation

The rough pose estimation is refined using ICP [3,7] (see Sect. 7). Because of the limited number of reference pose images in the database, ICP can improve the accuracy of the final alignment. Given the good initialization from the rough pose estimation ICP converges quickly. We evaluate the pose accuracy improvements due to ICP in Sect. 8.3.3.

4 Adaptive reference pose database construction

One way to create the reference pose database is to uniformly sample all possible (infinitely many) 3D orientations of an object. However, depending on the object or the application (e.g., picking), certain object poses appear more commonly than others. Instead of sampling orientations uniformly (as in our previous work [11]) we sample them more densely for poses that are more probable. This strategy is especially suitable for bin-picking, where the robot arm needs to select the object that is most easy to grasp, i.e., the topmost object or the object with the most reliable pose estimation.

4.1 Construction of one reference pose range image

For each pose to be tested, we render the 3D CAD model of an object using orthogonal projection. Because the depth values of the 3D scan and the physical size of the CAD models are known, we can scale the reference pose images such that the actual size of the object in both input and reference image is the same. These *reference range images* are then saved into one large image texture per object, consisting of all pose test samples. This *reference pose database* is uploaded to the memory of the graphics card during initialization of the pose estimation algorithm. In our experiments, we operate with a resolution of the reference range images of 64×64 and 32×32 pixels.

In addition to the depth values, we compute other values that are necessary for the pose estimation algorithm (Euclidean distance values, edge values, probability values; see Sects. 5 and 6.2.2). These are stored as 32 bit floating point values in the color and alpha channels of an image. A part of the reference pose database is illustrated in Fig. 2.

4.2 Estimation of pose probability density function

Because the shape of a 3D object is topologically uneven (unless the model is a sphere), some poses tend to appear more often than others if an object is dropped onto a surface (e.g., the situation in Fig. 3a is more likely than in Fig. 3b). To sample the pose space of an object non-uniformly, we estimate the probability density function (PDF) of the pose. The pose PDF is approximated by a histogram for each Euler angle by simulating dropping the object onto a surface or into a bin and observing the frequency of poses.

For our bin picking application, 3D objects are repeatedly dropped into bins with random initial location and pose. We used the PhysX library of NVIDIA [25] to simulate collisions between the objects and between objects and walls. An example of a virtual bin for the pipe object in Fig. 3 is shown in Fig. 4a. For each object model, we simulate about 500 bins with 50 to 80 objects each (about 35,000 to 40,000 objects in total) to get a robust estimation of the pose PDF. Theoretically, the PDF for each object should be estimated not only individually (i.e., by simulating bins consisting of only this object), but for all combinations of objects. However, the differences appeared to be marginal in our experiments.

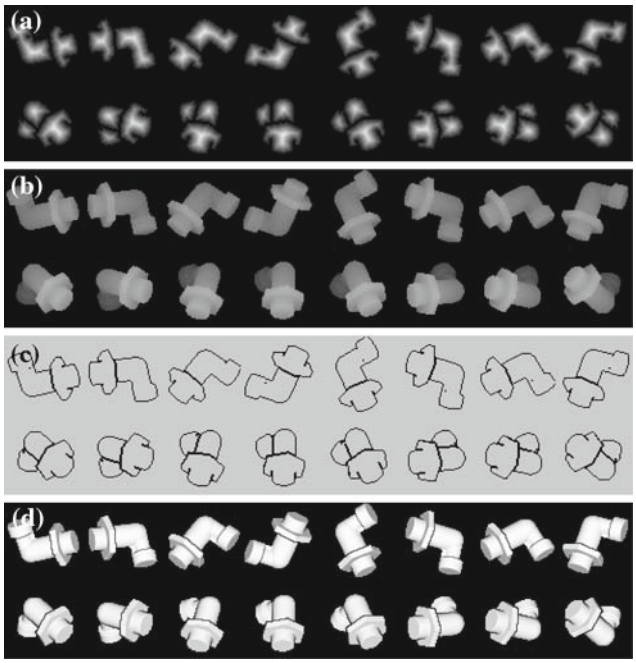

Fig. 2 A small section of the reference pose database: **a** the EDT values are stored in the red color channel of an RGB image, **b** the depth values are stored in the green channel, and **c** the edge pixels are stored in the blue channel, **d** the uncertainty value (see Sect. 6) is stored in the alpha channel

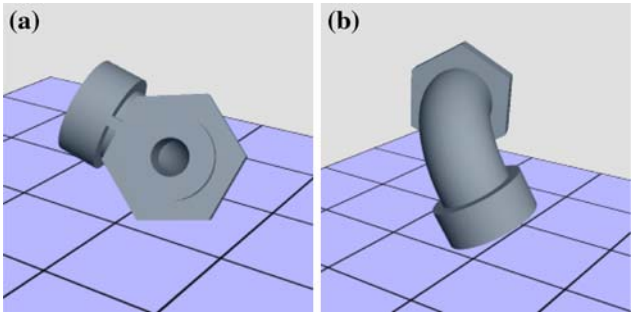

Fig. 3 Some poses of an object appear more often in a bin than others, e.g. (**a**) is more probable than (**b**)

In Figs. 4b–d, the results for the experiments with the pipe object from Fig. 3 are shown, which demonstrate a nonuniform distribution of the poses. Although Euler angles (i.e., successive rotations around three axis) are easy to use technically, a spherical parameterization (consisting of azimuth, elevation and in-plane rotation) is more intuitive for drawing. Dense and sparse pose regions are clearly distinguishable in Fig. 5a and e, where a position on the sphere represents azimuth and elevation angles, and the direction of the vector attached to a point represents the in-plane rotation.

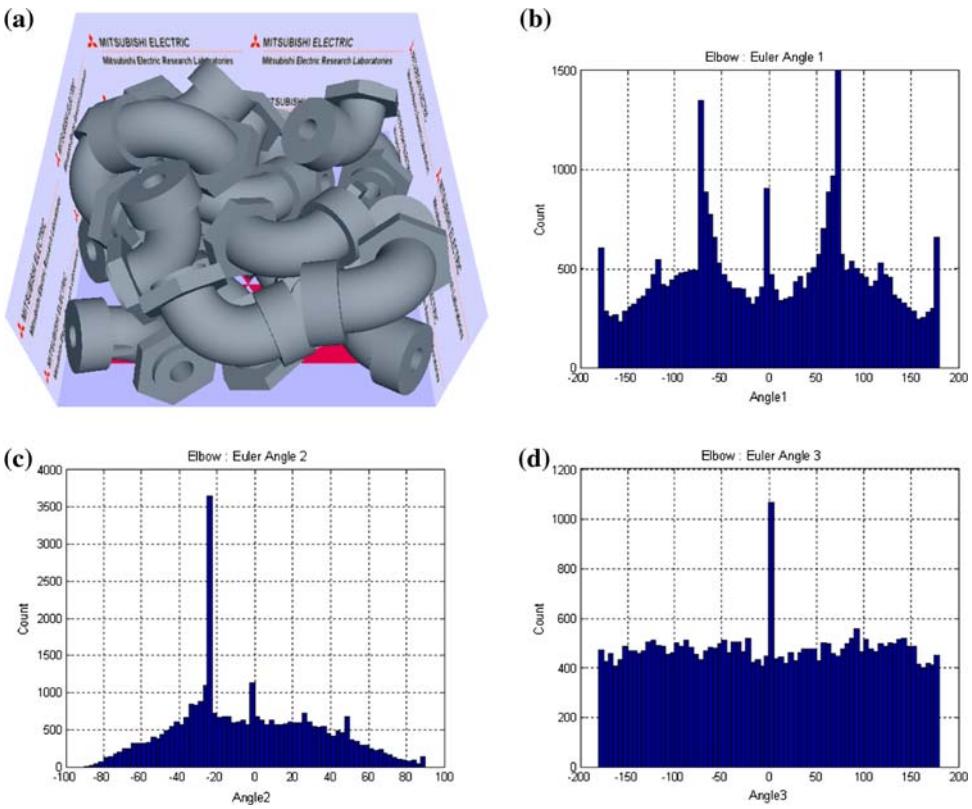

Fig. 4 The nonuniform pose histograms from the virtual bin simulation of the pipe object, using Euler angles: **a** example of a virtual bin, **b** pose histogram for rotation around the X axis, **c** around the Y axis, **d** around the Z axis

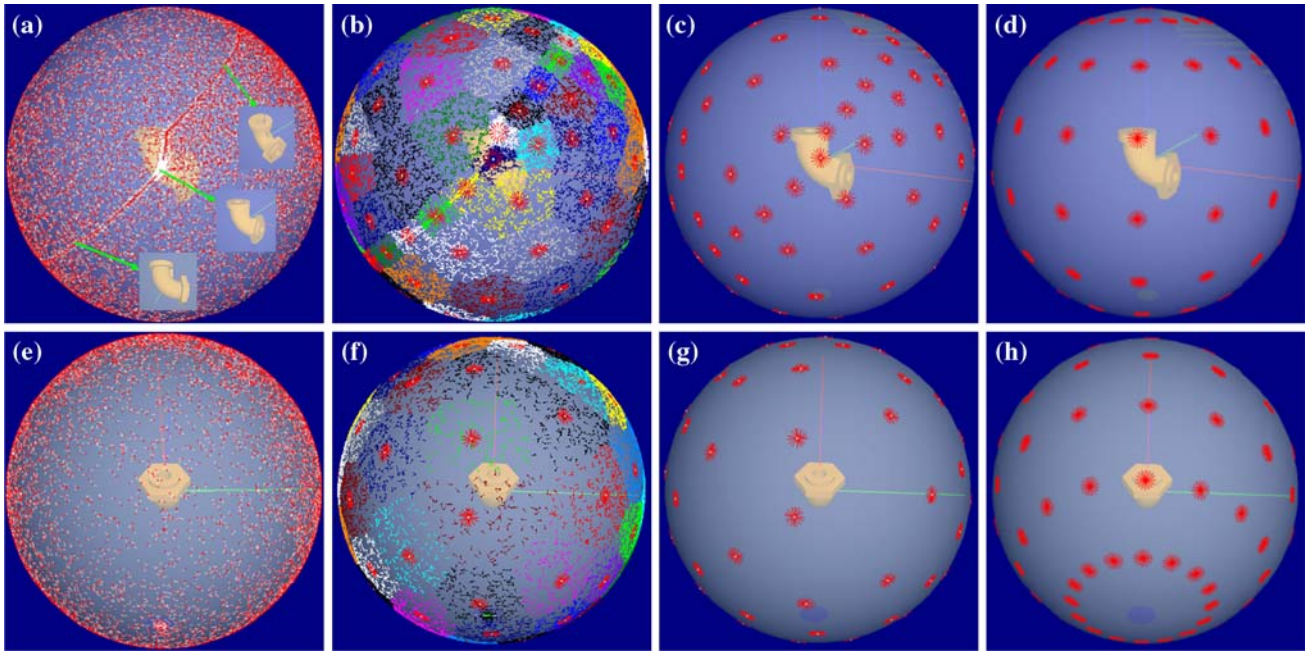

Fig. 5 Example of estimated pose distribution and clustering of poses using hierarchical K-means: In (**a**) and (**e**), each point on the unit sphere represents an observed pose, consisting of azimuth and elevation angle (position on the sphere) and in-plane rotation (direction of the vector attached to the point). In (**a**), a dense region with many observations of similar poses is shown (with some examples), whereas for the pose region in (**e**), not many observations have been made. In (**b**) and (**f**), we show the clustering results for the estimated pose distribution of the pose regions shown in (**a**) and (**d**). The observations belonging to one cluster center are colored. The images in (**c**) and (**g**) show the cluster centers in comparison to a regular sampling of the poses in (**d**) and (**h**). The poses represented by the cluster centers (azimuth and elevation) and *arrows* (in-plane rotation) are the poses in the reference pose database

4.3 Sampling poses using K-means clustering

To create more reference pose samples in pose areas of higher probability, we hierarchically cluster the observed poses using K-means [21]. First, we cluster with respect to the azimuth and elevation angles, i.e., the points on the unit sphere in Fig. 5a and e are clustered using the Euclidean distance as the distance measure. In a second step, we cluster the in-plane rotations using the distance between angles, taking into account all points from each cluster of the first step.

We create as many cluster centers as the desired number of reference pose range images for the pose database. Because the range of the elevation angle is $[0, \pi]$ compared to $[0, 2\pi]$ for azimuth and in-plane rotation, the cluster parameters (K_1 and K_2) are chosen proportionally to this range. In our case, to create 2,048 reference pose images (see Sect. 8), we chose $K_1 = 16 \times 8$ and $K_2 = 16$.

For the pose distribution illustrated in Fig. 5a and e, the clustering results are shown in Fig. 5b and f, where different clusters are marked with colors and with arrows (for the in-plane orientation). In Fig. 5c and g, the cluster centers are shown in comparison to the regularly placed samples in Fig. 5d and h.

5 Input range image processing

The input range image is typically acquired with an active light system (e.g., a laser range scanner), or with passive stereo methods (that are prone to more noise and holes). In our system the resolution of the input images are 128×128 or 64×64 pixels, i.e., double the resolution of the reference images to account for potential inexact initialization of the object position.

5.1 Edge detection

We reduce noise in the input image using a median filter with a mask of 3×3 pixels. The range image of the input scene in Fig. 6a is shown in Fig. 6b, and the result of the median filter in Fig. 6c. A simple heuristic is applied to detect edges in the image by comparing range values of neighboring pixels: If the range difference exceeds a predefined threshold (in our case 4% of the image width), a pixel is marked as an edge (see Fig. 6d).

5.2 Euclidean distance transform

Based on the edge image, we compute the signed Euclidean distance transform (EDT). The EDT is the distance of a pixel

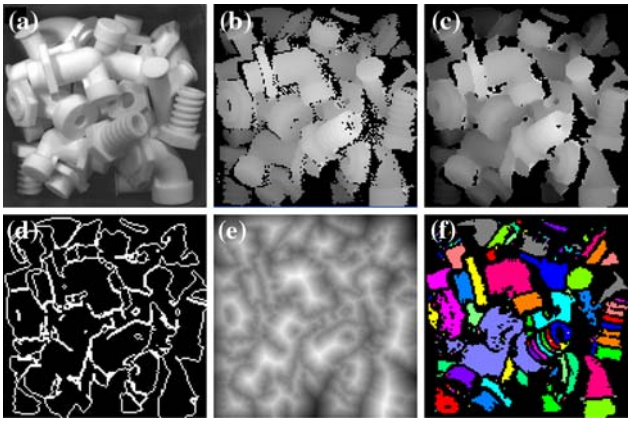

Fig. 6 Results of 3D scan processing: **a** input scene, **b** original range image, **c** median filtered range map, **d** result of edge detection, **e** signed Euclidean distance transform (EDT), **f** pixel grouping based on the of EDT image

Algorithm 1 Algorithm for parallelized signed EDT computation

$coord(p)$ = coordinates of the closest edge e found so far
$value(p)$ = signed distance value to e
Require: $value(b) = -(m+1) \quad \forall b \in$ background
Require: $value(f) = +(m+1) \quad \forall f \in$ foreground
Require: $value(e) = 0 \quad \forall e \in$ edge
Require: $coord(p) = (x_p, y_p) \quad \forall p \in$ image
for all iterations m **do**
  for all pixels p **do**
    for all direct neighbors n of p **do**
      if $distance(p, coord(n)) < |value(p)|$ **then**
        $value(p) = signed\_distance(p, coord(n))$
        $coord(p) = coord(n)$

to the nearest pixel containing an edge. An example of an EDT is shown in Fig. 6e, computed from Fig. 6d. In our GPU implementation of the EDT, we employ a variation of the *ping-pong rendering* algorithm [27,30]. We use two image buffers and consecutively switch their role as rendering source and target. Both buffers contain four channels per pixel and are allocated in the video memory. The values in the first two channels represent the coordinates of the closest edge pixel found so far, the third channel stores the signed distance, and the fourth channel indicates if an edge pixel is already found.

Algorithm 1 shows the pseudo-code of our EDT algorithm. The parameter m determines the number of iterations. The distance values are initialized to $-(m+1)$ for background pixels (i.e., range value $= 0$), to $m+1$ for foreground pixels (i.e., range value $\neq 0$), and to 0 for all edge pixels. The first two channels are initialized to the pixel coordinates. In each iteration, the distance value of each pixel is compared with the distance to the coordinates stored as closest edge pixels of its eight direct neighbors. The distance value and coordinates of the recent pixel p are updated if the distance from p to the edge pixel in a neighboring pixel n is smaller

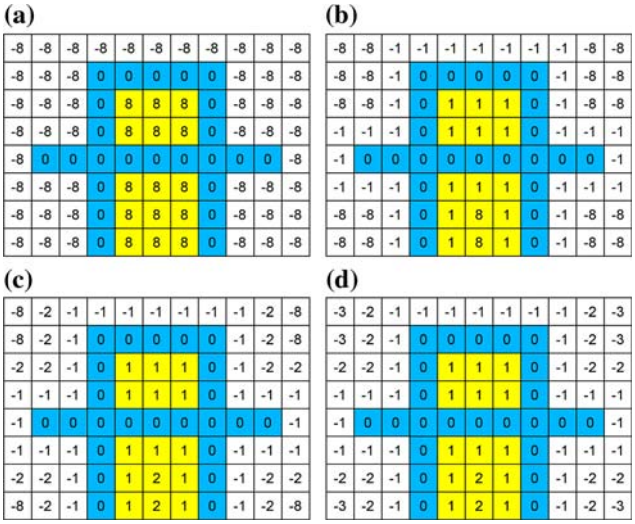

Fig. 7 Parallelized computation of the signed EDT: The values represent distances to the closest edge pixels. **a** shows the initialization step and **b–d** demonstrate the first three iterations

than the value saved at p. This information is iteratively propagated over the entire image at each step, as shown in Fig. 7.

The number of iterations m corresponds to the maximum distance of any pixel to its closest edge. For convergence, m needs to be equal to the length of the larger side of the image in pixels. However, to speed up the algorithm we make use of the fact that the distance of each pixel to an object edge is typically much smaller. Furthermore, an approximation of the exact EDT is sufficient for our purpose. In our experiments, we found that $m = 7$ for an image resolution of 64×64 pixels and $m = 15$ for 128×128 are sufficient.

5.3 Pixel grouping

To find proper initial search locations for the downhill simplex optimization, we perform a simple pixel grouping method in the input range image. Therefore, the pixels are grouped into patches such that each patch represents a (mostly) smooth segment of the object surface. They will be used for the initialization of the pose estimation algorithm (see Sect. 6.1). The initial positions for the downhill simplex will be searched within patches that are close to the camera.

Pixels in the input image are grouped using the edge image (see Sect. 5.1) by detecting connected non-edge pixel regions. First, different labels are assigned to each non-edge pixel. Then, non-edge pixels are iteratively merged if they have non-edge neighbors. This requires the same number of iterations (m) as for the EDT computation. Finally, the maximum EDT value of a pixel in one patch corresponds to the size of the patch. We filter out small patches (i.e., patches where the highest EDT value is low) to avoid over-segmentation. A pixel grouping result is shown in Fig. 6f.

6 Rough pose estimation

For the rough pose estimation step, each reference image is translated over the input image, and the corresponding alignment error is computed for each position. In the following sections, the rough pose estimation is described in detail. First, we present a method to find the initial translation of the object in the input image in Sect. 6.1. Then, we define the error function to measure the alignment of one reference range image R_i and an input range map I in Sect. 6.2. Finally, we present the optimization procedure to find the globally optimal pose match in Sect. 6.3. The optimization and error evaluation are performed in parallel for all reference range images.

6.1 Initial translation parameters

A good choice of initial translation parameters (x_0, y_0, z_0) influences the accuracy and the convergence rate of the downhill simplex optimization. Therefore, we look for a largely visible object that is close to the camera. This choice is particularly reasonable for the application of bin picking.

First, our algorithm computes the average depth values for each patch of the pixel grouping result (see Sect. 5.3). Then, it selects the patch P with the minimum average depth. The pixel $p \in P$ with highest EDT value is taken as an initial point that represents the "center" of the visible area.

To increase the robustness of the downhill simplex optimization we employ multiple initializations simultaneously by selecting the center pixels of the n closest patches. Examples of this method are shown in Fig. 8. In our experiments, we use $n = 1$ and $n = 3$ initial starting points to show the tradeoff between speed and accuracy (see Sect. 8).

6.2 Error function

Given an input range image I and the reference pose image R_i, the error value E_i for the translation parameters (x, y, z) is defined as:

$$E_i(I, R_i, x, y, z) = r_i \left(E_{\text{cover}}(x, y) + \lambda E_{\text{range}}(x, y, z)\right) \tag{1}$$

where

$$\begin{aligned} r_i &= \frac{A_i}{A_{\max}} \\ E_{\text{cover}}(x, y) &= \frac{\sum_{u,v} f(u+x, v+y)\delta_{\text{cover}}(u, v, x, y)}{\sum_{u,v} f(u+x, v+y)} \\ E_{\text{range}}(x, y, z) &= \frac{\sum_{u,v} c(u+x, v+y)\delta_{\text{range}}(u, v, x, y, z)}{\sum_{u,v} c(u+x, v+y)}. \end{aligned} \tag{2}$$

The error function in Eq. 1 consists of the cover error term $E_{\text{cover}}(x, y)$ and the depth error term $E_{\text{range}}(x, y, z)$. Both are evaluated at the pixel position (u, v) of I. The cover error term E_{cover} measures the error of aligning the silhouettes, inspired by [19], while the depth error term E_{range} compares the topologies of the reference and input range images. The term r_i is the ratio of the foreground area A_i of R_i compared to the maximum observed foreground area $A_{\max}$ across all the reference views. This term favors alignments (and thus poses) where a large part of the object is visible. The translation parameters (x, y, z) determine the relative position with respect to I. The two error terms are weighted by λ to balance the different scale of E_{cover} and E_{range}, and summed over all image pixels. λ is experimentally set to 20 by minimizing the position error in experiments with synthetic data where we have ground truth. The factors $f(u, v)$ and $c(u, v)$ make the error independent of the object size and assign less weight to less reliable pixels (see the next subsections). The alignment (x, y, z) of one reference pose image R_i is optimal if its error value is lower than any other alignment of R_i.

6.2.1 Cover error term

Given a pixel (u, v) in an alignment of R_i and I, the cover error is defined as the difference of the corresponding pixels in the Euclidean distance transform images of I and R_i, EDT_I and EDT_{R_i}:

$$\delta_{\text{cover}}(u, v, x, y) = \left|\text{EDT}_I(u, v) - \text{EDT}_{R_i}(u+x, v+y)\right| \tag{3}$$

where (x, y) is the translation vector of R_i with respect to I. The cover error term is minimal if the silhouettes and edges of the objects in I and R_i match perfectly. Only non-background pixels of R_i with positive range values are considered, enforced by the cover error weighting factor $f(u, v)$:

$$f(u, v) = \begin{cases} 1 & \text{if } EDT_{R_i}(u, v) > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

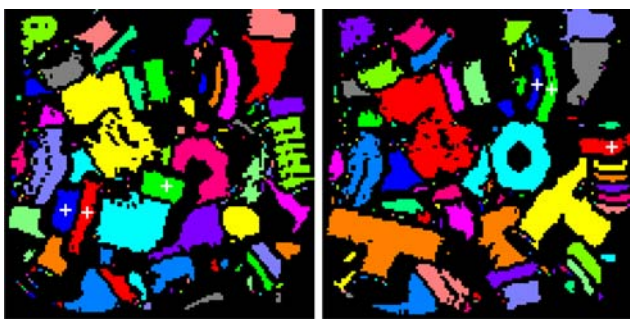

Fig. 8 Starting points (marked as white '+') for the optimization, selected by our algorithm

6.2.2 Depth error term

The depth error term compares the depth values z_I and z_{R_i} of the overlapping foreground pixels in I and R_i:

$$\delta_{\text{range}}(u, v, x, y, z) = \begin{cases} \left| z_I(u, v) - (z_{R_i}(u + x, v + y) + z) \right|, \\ \quad \text{if } z_I(u, v) \neq 0 \wedge z_{R_i}(u + x, v + y) \neq 0 \\ 0, \quad \text{otherwise.} \end{cases} \tag{5}$$

where R_i is translated by (x, y) and z is added to all range values of R.

If the angle of the surface normal is very large compared to the viewing direction (orthogonal to the image plane), the depth difference can become erroneously large, although the object in the reference image is well aligned to the input image. To put more weight on pixels where the alignment error is reliable, we introduce the term $c(u, v)$:

$$c(u, v) = \begin{cases} |\cos(\mathbf{V}, \mathbf{N}(u, v))| & \text{if } z_{R_i}(u, v) \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

where $\mathbf{V}$ is the viewing vector of R_i and $\mathbf{N}(u, v)$ is the surface normal vector at (u, v). $c(u, v)$ is computed offline for all reference pose images, and the values are stored in the alpha channel of the images in the database (see Fig. 2d; Sect. 4.1).

6.2.3 Implementation on the GPU

The error function in Eq. 2 is computed on the GPU using CUDA. In a naive implementation, all reference pose images would be stored in video memory, and the number of threads would be equal to the number of pixels in the reference view database. However, to avoid background pixels, only the image region around the bounding box of the object is transferred to video memory.

During online pose estimation, each thread computes the error terms of Eqs. 3 and 5 for one pixel. The current translation parameters (x, y, z) in Eqs. 1 and 2 and the error value are stored in separate arrays. Each pixel's score S_i is stored in an array of the global memory space on the graphics card.

In the next step, pixel-wise errors are summed up over all pixels of each reference view, yielding R_i's current error E_i in Eq. 1. To achieve parallelism, we sum the pixels using the scheme illustrated in Fig. 9: beginning with a step size $s = 1$, the values at the pixel positions (u, v), $(u+s, v)$, $(u+s, v+s)$, $(u, v + s)$ are added and stored at $S(u, v)$. Subsequently, s is doubled after each iteration. The final result of the error function is stored at pixel $S_i(0, 0)$ after $s = log(l)$ steps, where l is the image width in pixels.

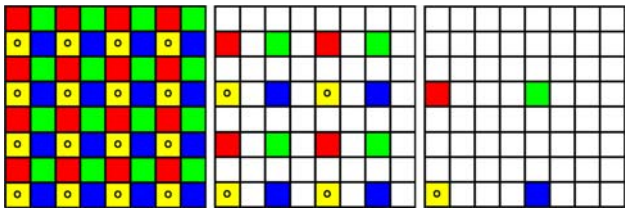

Fig. 9 Parallelized error computation on the GPU: in each iteration k, information at the current pixel position (marked with *circle*) is collected from the upper, upper right, and right neighbor at distance $s = 2^k$

6.3 Parallel optimization framework

We formulate pose estimation as the problem of finding the location parameters $(\hat{x}, \hat{y}, \hat{z})$ and orientation parameters $(\hat{\theta}, \hat{\phi}, \hat{\sigma})$ of an object in an image according to this 6-DOF optimization problem:

$$(\hat{x}, \hat{y}, \hat{z}, \hat{\theta}, \hat{\phi}, \hat{\sigma}) = \underbrace{\arg\min_i \left(\underbrace{\min_{x,y,z} E_i(I, R_i, x, y, z)}_{\text{step 1}} \right)}_{\text{step 2}} \tag{7}$$

In step 1, the translation parameters (x, y, z) are found, which minimize the error E_i (from Eq. 1), measuring the alignment of one reference pose image R_i and the input range image I. We use a parallelized version of the downhill simplex algorithm [23] (see Sect. 6.3.1 for implementation details). This is done for every pose hypothesis i and corresponding reference range image R_i in parallel. In our experiments, it took about 20 to 30 iterations for the downhill simplex algorithm to converge.

In step 2, we select the best pose hypothesis $\hat{i}$ (corresponding to the orientation parameters $(\hat{\theta}, \hat{\phi}, \hat{\sigma})$) with the lowest error, together with $(\hat{x}, \hat{y}, \hat{z})$ from step 1 belonging to $\hat{i}$.

6.3.1 Data-parallel downhill simplex on the GPU

To find the best translation parameters for one reference pose image in step 1 (Eq. 7), we use the downhill simplex algorithm [23,28]. Although the downhill simplex algorithm can require more iterations than other optimization algorithms based on gradients, it is well suited for a GPU implementation because the computational cost for each iteration is low and the read/write operations are well distributed across memory. An n-dimensional simplex consists of n+1 vertices, in our case ($n = 3$) it is a tetrahedron. After initialization, the downhill simplex algorithm changes the positions of the vertices in every iteration step towards a minimum of a given n-dimensional function. There are four types of

such changes: *reflection* where the vertex p_h with the highest function value is mirrored at the plane P defined by the three other points, *reflection and expansion* where the reflected vertex is also moved away from the plane, *contraction* where p is moved towards P, and *multiple contractions* where more than one vertex is moved towards the others. Only the parameters of the simplex (vertex positions) are updated, and the error function in Eq. 1 is evaluated using pixel-wise operations. Furthermore, there are no complex logical branches.

If we use a database with 2,048 reference pose images, 2,048 downhill-simplex algorithms run in parallel. For a reference pose image resolution of 64×64 pixels, over 8 million threads are processed in one iteration of our optimization procedure, exploiting the massive data-parallel processing power of modern GPUs.

The vertices of the simplex are initialized to (x_0, y_0, z_0), $(x_0 + d, y_0, z_0)$, $(x_0, y_0 + d, z_0)$ and $(x_0, y_0, z_0 + d)$, where x_0, y_0 and z_0 are the initial parameters described in Sect. 6.1. In our experiments, an adequate value for the step size of the simplex is $d = 2$ pixels.

The complete optimization procedure is implemented using three different CUDA programs. The first one implements the downhill simplex algorithm as described above. It takes care of the shape changes of the simplex. The second program computes the error terms in the Eqs. 3 and 5, and the third one computes the final error value (Eq. 1). These three programs are executed for each evaluation of the error function in the downhill simplex algorithm. If the normalized difference between maximum and minimum error values in one iteration is below a threshold of 5% (see [28]), the downhill simplex thread stops and the parameters (x, y, z) are stored for this reference pose. Finally, the reference pose image with the least error is returned together with its stored parameters as the result.

7 Fine pose estimation

Although the accuracy of our rough pose estimation is adequate for most applications (see Sect. 8), we can refine the estimation using ICP.

The vertex densities of the CAD mesh model and a mesh model of the input image are not equal. Also, a simple mesh of the input is often irregular. Therefore, a direct application of ICP often does not work correctly. To overcome this problem, we use a partial mesh model obtained by rendering the CAD model in the roughly estimated pose. This adjusts the vertex resolution of the reference to the input model, and provides significant overlap between the meshes. If the rough pose estimation results in a correctly estimated pose, ICP further reduces the remaining pose error (see Sect. 8; Fig. 10).

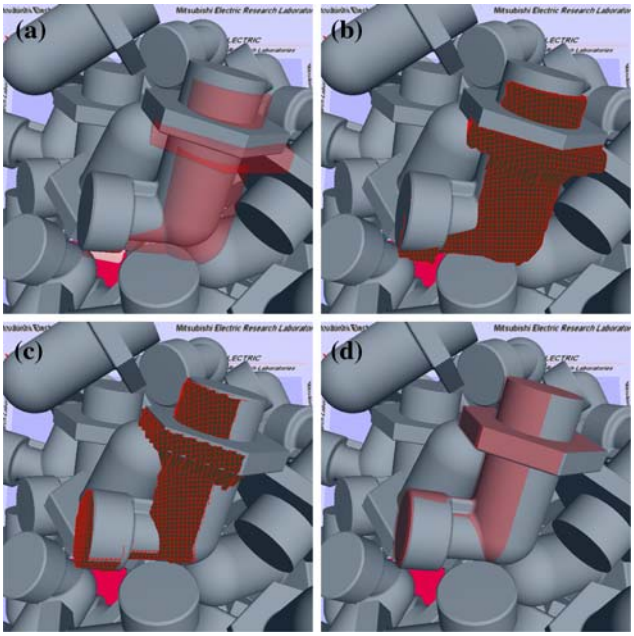

Fig. 10 Pose estimation results: **a** the estimated rough pose (shown overlaid), **b** the partial mesh constructed from the depth map based on the estimated pose, **c** the result of ICP using the partial mesh in (**b**) and the mesh from depth map of the input scene, **d** the estimated fine pose

8 Experiments and discussion

We evaluated the algorithm extensively on different synthetic and real input scenes. In this section, we present both quantitative and qualitative results. Furthermore, we illustrate parameter choices and discuss the performance and the memory footprint of our algorithm. All results were computed using a PC with a 2.83 GHz Intel Core2 Quad CPU (Q9550) and a NVIDIA GTX 280 GPU with 1GB of video memory.

8.1 Datasets

We used 243 different datasets consisting of bins with the same and with different objects.

8.1.1 Synthetic datasets

We constructed 210 virtual bins using a physics-based simulation of objects falling into a bin. We used this data as ground truth to precisely analyze the accuracy of our algorithm. Figure 11 shows nine virtual bins consisting of 20 to 30 objects. In this example, each bin is filled with objects of the same type. In our experiments, the objects whose pose had been estimated successfully were removed from the bin one by one until the bin was empty.

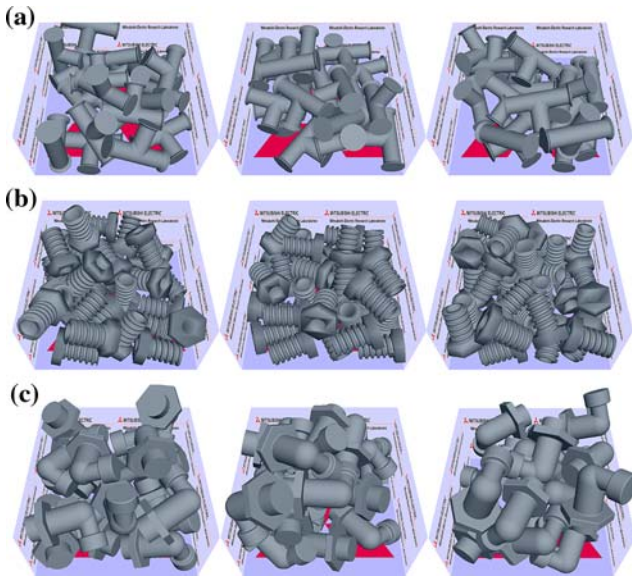

Fig. 11 Synthetic test scenes: **a** Three different bins with 20 T-pipe models, **b** three bins with 30 bolt models, **c** three bins with 20 elbow pipe models in each bin

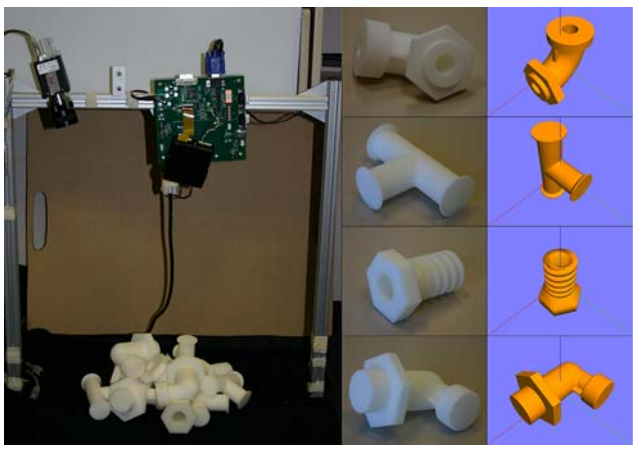

Fig. 12 Experimental setup: the structured light range scanner (*left*). Real objects printed by a 3D printer (*middle*). Corresponding 3D CAD models (*right*)

8.1.2 Real datasets

To record real-world data we used a structured light range scanner consisting of an IEEE 1394 camera and a pocket-size laser projector (see Fig. 12). The bins and objects were printed on a 3D printer. Figure 12 shows the hardware setup, the 3D CAD models, and the printed objects.

We constructed 12 different scenes of real bins with the same type of objects (see Fig. 13, corresponding to the synthetic bins shown in Fig. 11), and 21 scenes with four different objects (see Fig. 15 for the bin and Fig. 12 for an image of the four objects). In Fig. 15 we demonstrate our bin picking experiment, performing object detection and pose estimation,

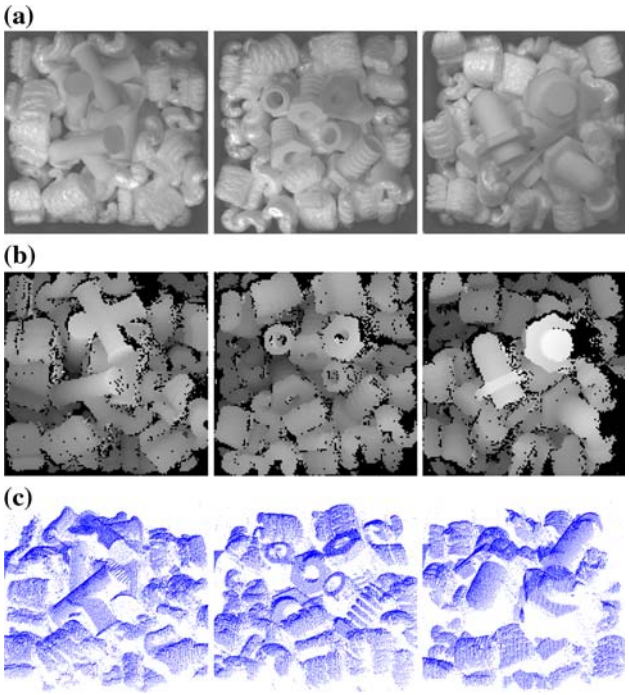

Fig. 13 Examples of real test scenes: **a** bins with single type of objects (corresponding to the synthetic scenes in Fig. 11), **b** the input depth images, and **c** the 3D point clouds (slightly rotated for visualization)

and subsequently removing the corresponding object one by one until the bin is empty.

8.2 Error analysis

For test data with ground truth (i.e., the synthetic dataset) we present results using two error metrics: geometric distance and Euler angles. The geometric distance error (vertex error) is computed as the vertex-to-vertex distance between the corresponding vertices of the object in the input scene (or its ground truth, respectively) and the object model with the estimated pose. The distance is normalized by the diagonal length of the bounding box. The Euler angle error consists of the difference for each of the three rotational angles around the x, y, and z axes.

We classify the estimated rough pose as correct if the geometric distance error is less than 10%, which is accurate enough for ICP to converge. The angular errors are computed only for objects with correct pose estimates because wrong estimations are often caused by object symmetries (see Sect. 9) and thus can be as large as 180°.

8.3 Results for the synthetic datasets

A few examples of typical results are shown in Fig. 14, where the estimated pose is shown using semi-transparency. The model is mostly well-aligned. The small red symbol shows

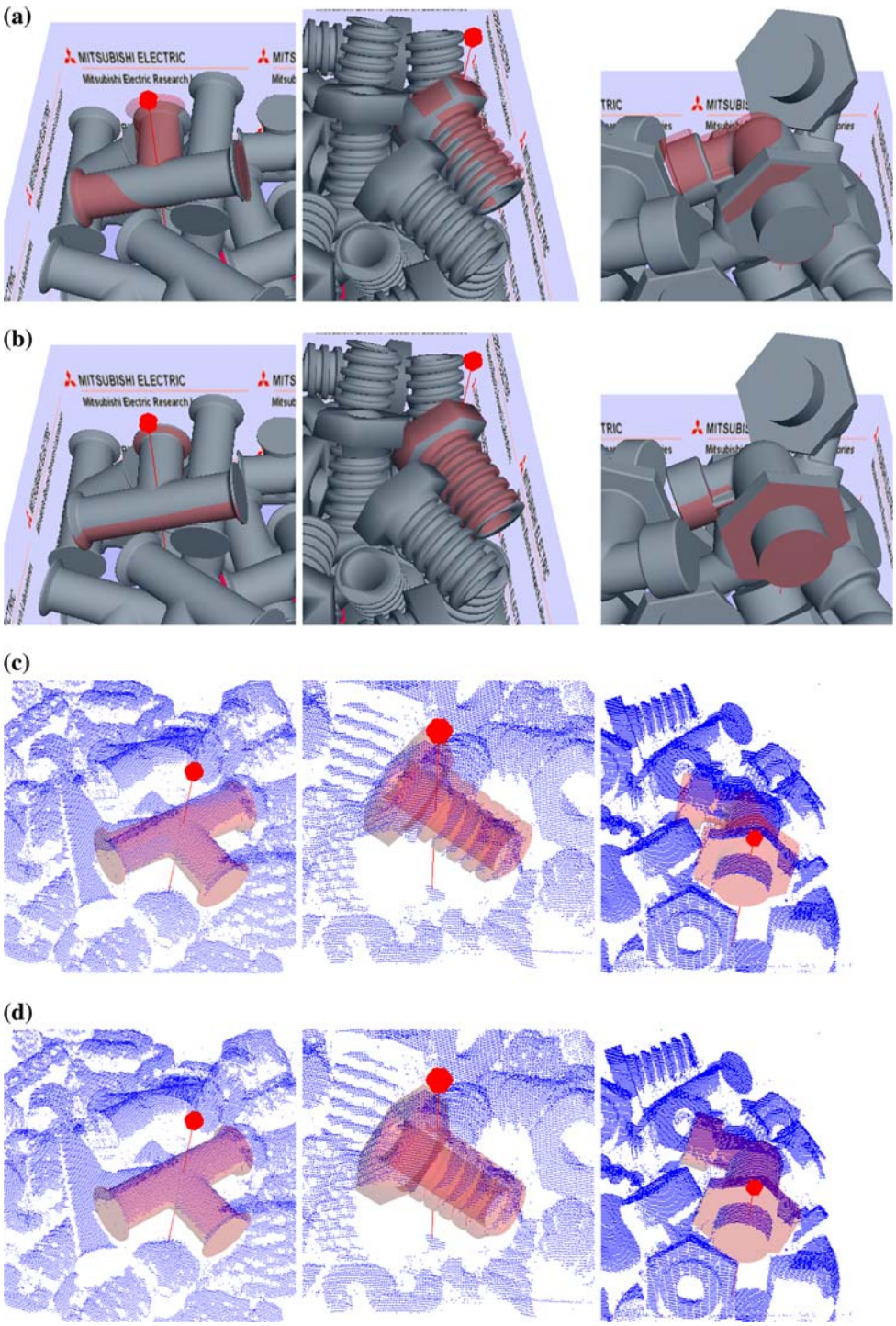

Fig. 14 Pose estimation results for synthetic (**a**, **b**) and real scenes (**c**, **d**): (**a**, **c**) after rough pose estimation, (**b**, **d**) after fine pose estimation

the location of the initial search point. In general, the rough pose estimation is successful in 96% of the test cases, and the axis-angle error for those cases is around 1 degree after the ICP stage.

8.3.1 Effect of reference pose image resolution

We quantitatively evaluated the accuracy of our algorithm without the pose refinement stage (see Table 1) for the

Table 1 Success rates and errors of rough pose estimations for the synthetic datasets with parameters $N_{\text{ref}} = 1{,}024$ and $N_{\text{init}} = 3$ using uniformly and adaptively sampled databases

Model	Trials	Sampling method	Resolution					
			32 × 32			64 × 64		
			Correct poses	Vertex error	Angular error (in Euler angles)	Correct poses	Vertex error	Angular error (in Euler angles)
T-Pipe	60	Uniform	48	6.78	(7.75, 6.91, 5.55)	49	6.18	(8.60, 6.27, 5.40)
		Adaptive	52	5.24	(5.58, 4.87, 3.46)	56	5.11	(6.56, 5.73, 3.06)
Bolt	90	Uniform	88	6.73	(7.10, 6.70, 6.44)	89	4.98	(6.42, 5.43, 5.42)
		Adaptive	89	5.99	(6.41, 5.92, 6.94)	90	4.59	(6.04, 4.81, 4.67)
Elbow	60	Uniform	38	5.67	(7.40, 8.10, 5.75)	44	5.23	(10.72, 6.75, 4.64)
		Adaptive	55	4.55	(5.60, 5.03, 4.99)	57	5.03	(5.87, 5.97, 5.26)
Average	210	Uniform	174 (83%)	6.44	(7.37, 7.16, 5.99)	182 (87%)	5.40	(8.27, 6.05, 5.19)
		Adaptive	196 (93%)	5.26	(5.86, 5.27, 5.13)	203 (97%)	4.91	(6.16, 5.50, 4.33)

Table 2 Fine pose estimation error for the synthetic dataset with parameters $N_{\text{ref}} = 1{,}024$ and $N_{\text{init}} = 3$

Data	Trials	32 × 32		64 × 64	
		Vertex error	Angular error (in Euler angles)	Vertex error	Angular error (in Euler angles)
T-Pipe	60	0.62	(0.20, 0.52, 0.28)	0.58	(0.61, 0.66, 0.26)
Bolt	90	2.48	(1.58, 1.56, 4.06)	1.28	(0.27, 0.35, 1.48)
Elbow	60	0.35	(0.11, 0.12, 0.11)	0.18	(0.04, 0.04, 0.04)
Average	210	1.15	(0.63, 0.73, 1.48)	0.68	(0.31, 0.35, 0.59)

synthetic dataset. For these experiments, the reference pose database consists of $N_{\text{ref}} = 1{,}024$ poses. We use $N_{\text{init}} = 3$ starting positions for the downhill simplex algorithm (see Sect. 6.3). The estimation success rate is 97% for a reference pose resolution of 64 × 64 pixels, and 93% for a resolution of 32 × 32 pixels.

8.3.2 Effect of adaptive pose database construction

As can be seen in Table 1, the effect of adaptively constructing the pose database according to the estimated pose distribution is significant; the success rate increases by about 10%.

8.3.3 Effect of ICP

Table 2 shows the pose estimation result using ICP. There is a substantial improvement, and the refinement results in an almost perfect alignment. This effect is slightly higher for reference pose images with a resolution of 64 × 64 pixels. The reason is that the mesh resolution is important for the precision of ICP in contrast to the rough pose estimation algorithm.

8.3.4 Effect of the reference database size

We increased the size of the reference database from 1,024 reference pose range images to 2,048 pose samples. The effect is marginal, as can be seen in Table 3 from both the success rate as well as the distance and angular errors. We made the same observation for the real dataset. The reason is that the smaller reference database already samples the pose space effectively using the proposed adaptive pose sampling algorithm.

The memory size of the reference pose database depends on the object type, image resolution and database size (see Table 4). It is well manageable on modern graphics cards.

8.3.5 Runtime

Table 5 shows the runtimes for the experiments described so far, for different image resolutions and database sizes. The runtime does not primarily depend on the object type. Instead, it is proportional to the resolution and the size of the reference view database. As discussed before, using a database with an image resolution of 64 × 64 pixels and 2,048 pose samples did not result in a significant increase of accuracy. For the

Table 3 Success rates and errors of pose estimations for the synthetic datasets with parameters $N_{\text{ref}} = 2{,}048$ and $N_{\text{init}} = 3$

Model	Trials	Error measure	Resolution					
			32 × 32			64 × 64		
			Correct poses	Vertex error	Angular error (in Euler angles)	Correct poses	Vertex error	Angular error (in Euler angles)
T-Pipe	60	Rough pose	57	5.24	(5.67, 5.73, 2.89)	57	4.74	(5.66, 5.69, 2.59)
		Fine pose		0.70	(0.43, 0.35, 0.30)		0.18	(0.05, 0.06, 0.04)
Bolt	90	Rough pose	89	6.37	(9.17, 6.94, 6.47)	90	0.05	(3.70, 4.27, 6.06)
		Fine pose		2.26	(2.17, 1.93, 3.05)		0.01	(0.29, 0.32, 1.89)
Elbow	60	Rough pose	54	4.63	(6.68, 5.17, 4.70)	55	3.63	(5.14, 4.97, 4.42)
		Fine pose		0.32	(0.12, 0.12, 0.14)		0.14	(0.04, 0.03, 0.03)
Average	210	Rough pose	200 (95%)	5.41	(7.17, 5.95, 4.69)	202 (96%)	2.81	(4.83, 4.98, 4.36)
		Fine pose		1.09	(0.91, 0.80, 1.16)		0.11	(0.13, 0.14, 0.65)

Table 4 Required video memory for different reference pose databases (in MBytes)

	1024		2048	
	32 × 32	64 × 64	32 × 32	64 × 64
T-Pipe	25.61	73.22	47.00	142.17
Bolt	20.40	50.13	36.41	95.74
Elbow	27.79	82.15	51.48	160.80

Table 5 Computation times T_{Rough} and T_{Fine} for the synthetic dataset (in seconds)

Data	Resolution			
	1024		2048	
	32 × 32	64 × 64	32 × 32	64 × 64
T-Pipe				
Rough pose	0.33	0.46	0.65	0.95
Fine pose	0.01	0.02	0.01	0.02
Bolt				
Rough pose	0.26	0.32	0.50	0.63
Fine pose	0.01	0.02	0.01	0.02
Elbow				
Rough pose	0.37	0.51	0.72	1.02
Fine pose	0.02	0.02	0.02	0.02
Average				
Rough pose	0.32	0.43	0.62	0.87
Fine pose	0.01	0.02	0.01	0.02

database with 1,024 samples our algorithm needs between 0.3 and 0.5 s.

In all the experiments presented so far we used $N_{\text{init}} = 3$ starting positions for the downhill simplex algorithm (see Sect. 6.3). This parameter strongly depends on the degree of occlusion in the scene. If we use just one starting position ($N_{\text{init}} = 1$) the runtime drops by a factor of three.

8.4 Results for the real datasets

For the real scene shown in Fig. 13, the result is shown in Fig. 14. For the experiments with the more realistic dataset we used a reference pose database with $N_{\text{ref}} = 4{,}096$ poses (1,024 views per object type) and an image resolution of 64 × 64 pixels. The result of sequential object detection and removal

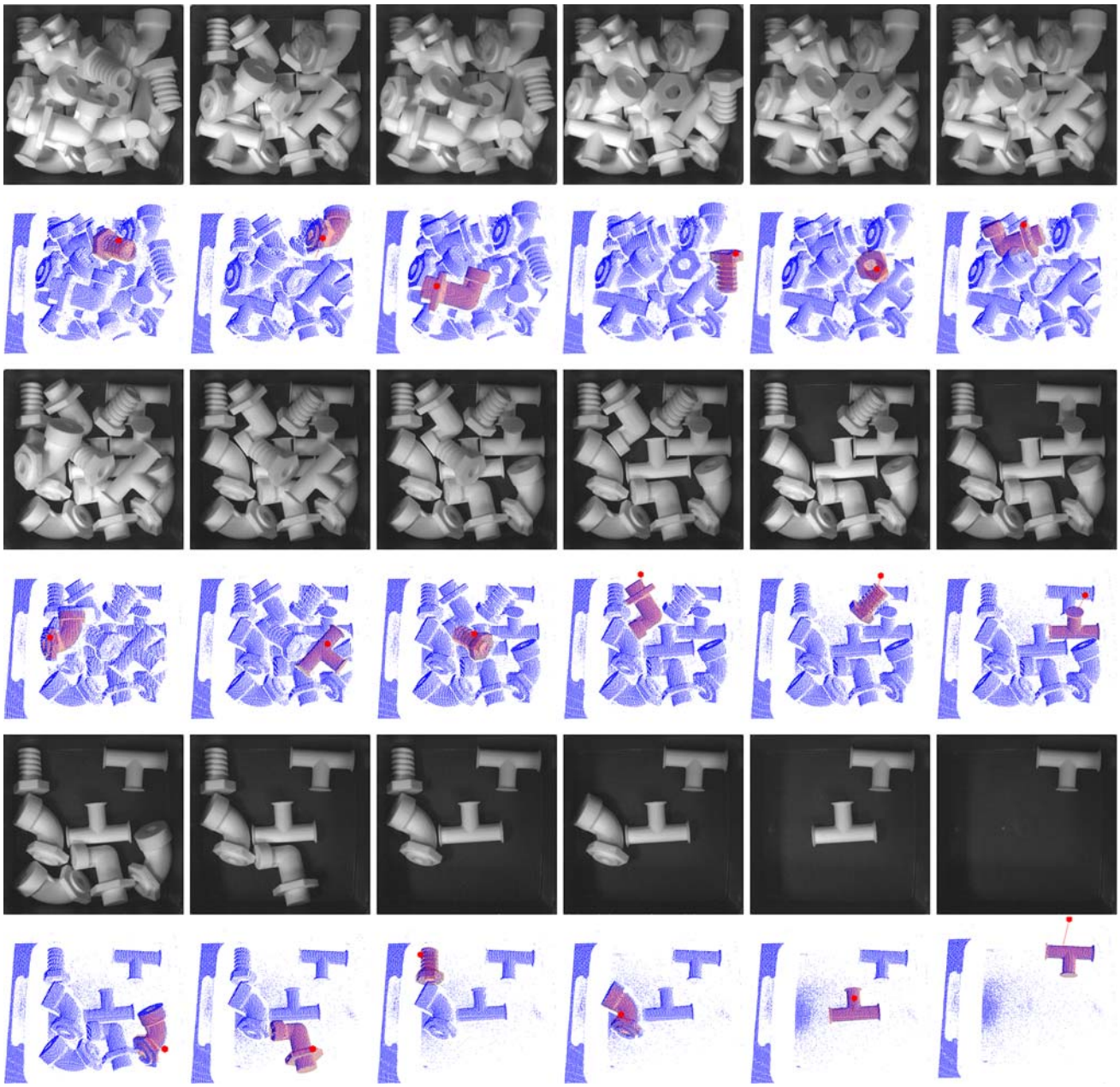

Fig. 15 Results of consecutive pose estimation and object removal for a complex bin with different types of objects. For every step, the bin and the corresponding depth scan are shown. The detected object which is removed afterwards is marked *overlaid*

is shown in Fig. 15, where in each step the pose of an object was estimated. Because no ground truth is available, the pose estimation accuracy can not be evaluated quantitatively but only qualitatively (compare the pose of the projected model with the estimated pose (marked red) in Fig. 15).

Our approach is applicable to scenes with different objects, as long as the object types in the scene are known and 3D CAD models are available beforehand. Since computation complexity increases proportionally to the number of reference views, the processing time in this experiment increases to 0.57 seconds for $N_{\text{init}} = 1$ initial positions for the downhill simplex algorithm, and to about 1.7 s for $N_{\text{init}} = 3$ initial search points.

8.5 Comparison to a CPU implementation

To compare the runtime between GPU and CPU we ported the source code to the CPU. Parallel computations are now executed as double loops. On average, the CPU runtime is about 115 times higher than on the GPU (note that we run the

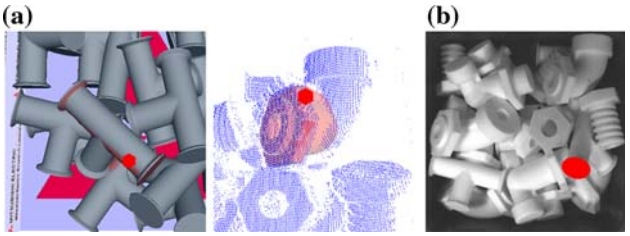

Fig. 16 Typical failure cases: **a** ambiguous self occlusion, **b** bad initial position for the Downhill Simplex algorithm (on the surface patch marked as *solid circle*)

code on a quad core CPU). However, since the CPU implementation is not optimized, this comparison is not entirely fair.

8.6 Failure cases

In general, the robustness and success rate of our algorithm is very high. Typical reasons for a wrong pose estimation are either large self-occlusions or ambiguity of the object location because of several similar-looking objects in the scene (see Fig. 16a). Additionally, heavy occlusions around the initial search position for the Downhill Simplex algorithm can lead to a wrong initialization and thus to an erroneous pose estimation. An example for this case is shown in Fig. 16b), where the initial search point (red) is at the very border of an object.

In rare cases, the visible area of an object is too small such that several objects with different poses are aligned well to the visible part. However, our algorithm is trying to avoid such cases by favoring poses with a largely visible object surface (using the factor r_i in Eq. 1 in Sect. 6.2). Additionally, the objects used in our experiments have some similar parts that can cause confusion, and the range scans could contain large holes that make it difficult to unambiguously align an object.

9 Conclusions and future work

We presented a model-based algorithm for automatic 3D pose estimation that has been designed for efficient GPU implementation. The method consists of three stages: adaptive construction of a reference pose database, rough pose estimation without initialization, and pose refinement using ICP.

We applied our method to bin picking, and demonstrated that our system works fast, accurately, and reliably. Furthermore, we illustrated the influence of different algorithm parameters and evaluated the performance using challenging synthetic and real data. The algorithm is scalable with respect to the number of objects in the database, which is only limited by the memory on the graphics card.

Future work will include the investigation of automatic symmetry detection for the objects to reduce the pose search space. This is currently done by hand, which is feasible for a typical bin-picking setting where the algorithm usually has to deal only with a few different object types. Furthermore, we plan to apply our system to a robot assembly environment. Therefore, small modifications due to mechanical restrictions could be useful, e.g., to reject objects that a real robot arm cannot reach, or to take into account the velocity of the arm to favor objects near its current position.

Acknowledgments This work was supported in part by an INHA UNIVERSITY Research Grant. This work was performed when In Kyu Park and Marcel Germann were visiting Mitsubishi Electric Research Laboratories (MERL) as a visiting scholar and student intern, respectively. The authors would like to thank Dr. J. Katz, Dr. J. Thornton, Dr. K. Kojima at MERL and Dr. H. Okuda at Mitsubishi Electric Corporation for their consistent support of the project, M. Bächer and J. Barnwell for their help in 3D range scanning and 3D printing, and other researchers at MERL for the valuable discussions.

References

1. Al-Hujazi, E., Sood, A.: Range image segmentation with applications to robot bin-picking using vacuum gripper. IEEE Trans. Syst. Man Cybern. **20**(6), 1313–1325 (1990)
2. Berger, M., Bachler, G., Scherer, S.: Vision guided bin picking and mounting in a flexible assembly cell. In: Proceedings of the 13th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, pp. 109–117 (2000)
3. Besl, P., McKay, N.: A method for registration of 3-d shapes. IEEE Trans. Pattern Anal. Mach. Intell. **14**(2), 239–256 (1992)
4. Blanz, V., Vetter, T.: A morphable model for the synthesis of 3D faces. In: Proceedings of ACM SIGGRAPH, pp. 187–194 (1999)
5. Boughorbel, F., Zhang, Y., Kang, S., Chidambaram, U., Abidi, B., Koschan, A., Abidi, M.: Laser ranging and video imaging for bin picking. Assembl. Autom. **23**(1), 53–59 (2003)
6. Breitenstein, M.D., Kuettel, D., Weise, T., Gool, L.V., Pfister, H.: Real-time face pose estimation from single range images. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (2008)
7. Chen, Y., Medioni, G.: Object modeling by registration of multiple range images. Comput. Vis. Image Underst. **10**(3), 145–155 (1992)
8. Dorai, C., Jain, A.K.: Cosmos—a representation scheme for 3d free-form objects. IEEE Trans. Pattern Anal. Mach. Intell. **19**(10), 1115–1130 (1997)
9. Gelfand, N., Mitra, N., Guibas, L., Pottmann, H.: Robust global registration. In: Proceedings of Eurographics Symposium on Geometry Processing, pp. 197–206 (2005)
10. General purpose gpu programming (gpgpu) website. http://www.gpgpu.org
11. Germann, M., Breitenstein, M.D., Park, I.K., Pfister, H.: Automatic pose estimation for range images on the gpu. In: Proceedings of International Conference on 3-D Digital Imaging and Modeling, pp. 81–88 (2007)
12. Greenspan, M.: Geometric probing of dense range data. IEEE Trans. Pattern Anal. Mach. Intell. **24**(4), 495–508 (2002)
13. Greenspan, M., Shang, L., Jasiobedzki, P.: Efficient tracking with the bounded hough transform. In: Proceedings of IEEE Confer-

ence on Computer Vision and Pattern Recognition, pp. I-520–I-527 (2004)
14. Ikeuchi, K.: Generating an interpretation tree from a cad model for 3d object recognition in bin-picking tasks. Int. J. Comput. Vis. **1**(2), 145–165 (1987)
15. Johnson, A.E., Hebert, M.: Using spin images for efficient object recognition in cluttered 3d scenes. IEEE Trans. Pattern Anal. Mach. Intell. **21**(5), 433–449 (1999)
16. Jones, M., Poggio, T.: Multidimensional morphable models: a framework for representing and matching object classes. Int. J. Comput. Vis. **29**(2), 107–131 (1998)
17. Jones, M.J., Viola, P.: Fast multi-view face detection. Technical Report TR2003-96, Mitsubishi Electric Research Laboratories (2003)
18. Lamdan, Y., Wolfson, H.: Geometric hashing: a general and efficient model-based recognition sceme. In: Proceedings of International Conference on Computer Vision, pp. 238–249 (1988)
19. Lee, J., Moghaddam, B., Pfister, H., Machiraju, R.: Finding optimal views for 3d face shape modeling. In: Proceedings of IEEE International Conference on Automatic Face and Gesture Recognition, pp. 31–36 (2004)
20. Liebelt, J., Schmid, C., Schertler, K.: Viewpoint-independent object class detection using 3d feature maps. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (2008)
21. MacQueen, J.B.: Some methods for classification and analysis of multivariate observations. In: Proceedings of the fifth Berkeley Symposium on Mathematical Statistics and Probability, vol. 1, pp. 281–297 (1967)
22. Mian, A.S., Bennamoun, M., Owens, R.: Three-dimensional model-based object recognition and segmentation in cluttered scenes. IEEE Trans. Pattern Anal. Mach. Intell. **28**(12), 1584–1601 (2006)
23. Nelder, J.A., Mead, R.: A simplex method for function minimization. Comput. J. **7**(4), 308–313 (1965)
24. NVIDIA Corporation: Compute Unified Device Architecture (CUDA). http://developer.nvidia.com/object/cuda.html
25. NVIDIA Corporation: PhysX SDK. http://developer.nvidia.com/object/physx.html
26. Okuda, H., Kitaaki, Y., Hashimoto, M., Kaneko, S.: Hm-icp: fast 3-d registration algorithm with hierarchical and region selection approach of m-icp. J. Robot. Mechatron. **18**(6), 765–771 (2006)
27. Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A.E., Purcell, T.J.: A survey of general-purpose computation on graphics hardware. Comput. Graph. Forum **26**(1), 80–113 (2007)
28. Press, W., Teukolsky, S., Vetterling, W., Flannery, B.: Numerical Recipes in C: The Art of Scientific Computing. Cambridge University Press, Cambridge (1992)
29. Rahardja, K., Kosaka, A.: Vision-based bin-picking: recognition and localization of multiple complex objects using simple visual cues. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems pp. 1448–1457 (1996)
30. Rong, G.D., Tan, T.S.: Jump flooding in gpu with applications to voronoi diagram and distance transform. In: Proceedings of ACM Symposium on Interactive 3D Graphics and Games, pp. 109–116 (2006)
31. Rothganger, F., Lazebnik, S.J., Ponce, C.S.: 3d object modeling and recognition using affine-invariant patches and multi-view spatial constraints. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 272–277 (2003)
32. Rusinkiewicz, S., Hall-Holt, O., Levoy, M.: Real-time 3d model acquisition. ACM Trans. Graph. **21**(3), 438–446 (2002)
33. Rusinkiewicz, S., Levoy, M.: Efficient variants of the icp algorithm. In: Proceedings of International Conference on 3-D Digital Imaging and Modeling, pp. 145–152 (2001)
34. Schmid, C., Mohr, R.: Combining greyvalue invariants with local constraints for object recognition. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 872–877 (1996)
35. Shang, L., Jasiobedzki, P., Greenspan, M.: Discrete pose space estimation to improve icp-based tracking. In: Proceedings of International Conference on 3-D Digital Imaging and Modeling, pp. 523–530 (2005)

Author Biographies

In Kyu Park received B.S., M.S., and Ph.D. degrees from Seoul National University (SNU) in 1995, 1997, and 2001, respectively, all in electrical engineering and computer science. From September 2001 to March 2004, he was a Member of Technical Staff at Samsung Advanced Institute of Technology (SAIT). Since March 2004, he has been with the School of Information and Communication Engineering, Inha University, where he is an assistant professor. From January 2007 to February 2008, he was an exchange scholar at Mitsubishi Electric Research Laboratories (MERL). Dr. Park's research interests include the joint area of computer graphics and vision, including 3D shape reconstruction from multiple views, image-based rendering, computational photography, and GPGPU. He is a member of IEEE and ACM.

Marcel Germann received the M.Sc. degree in Computer Science at ETH Zurich in 2007. He is currently a Ph.D. candidate and research assistant at the Computer Graphics Laboratory of Prof. Markus Gross at ETH Zurich. In 2006 he was a graduate intern at Mitsubishi Electric Research Laboratories (MERL) in Cambridge, USA. His research interests are in video-based reconstruction and rendering methods of human bodies and objects, combining elements from both, computer vision and computer graphics.

Michael D. Breitenstein received the M.Sc. degree in Computer Science at ETH Zurich in 2006, and is expected to finish his Ph.D. in Information Technology and Electrical Engineering at ETH Zurich by 2009. He is currently a research assistant in the Computer Vision Lab of Prof. Luc Van Gool. From 2005 to 2006, he was a graduate intern at MERL in Cambridge, US, and in 2007, he was a visiting researcher at the Active Vision Lab of Oxford University. His current research interests are in visual surveillance and learning from dynamic scenes. They include multi-object tracking, unusual event detection, and scene structure learning. Michael published more than ten papers in major international conferences and journals, and he holds several patents. During his Ph.D., he co-founded a start-up company for knowledge transfer and innovation management.

Hanspeter Pfister is Gordon McKay Professor of the Practice in the School of Engineering and Applied Sciences at Harvard University. His research lies at the intersection of visualization, computer graphics, and computer vision. Before joining Harvard he worked for 11 years at Mitsubishi Electric Research Laboratories where he was most recently Associate Director and Senior Research Scientist. Pfister has a Ph.D. in Computer Science from the State University of New York at Stony Brook and an M.S. in Electrical Engineering from the Swiss Federal Institute of Technology, Zurich, Switzerland. He is a senior member of the IEEE Computer Society and member of ACM, ACM SIGGRAPH, and the Eurographics Association.