Depth-assisted Real-time 3D Object Detection for Augmented Reality [*]

Wonwoo Lee[†] GIST U-VR Lab.
Nohyoung Park[‡] GIST U-VR Lab.
Woontack Woo[§] GIST U-VR Lab.

Abstract

In this paper, we propose a novel method of real-time object detection that can recognize three-dimensional (3D) target objects, regardless of their texture and lighting condition changes. Our method computes a set of reference templates of a target object from both RGB and depth images, which describes the texture and geometry of the object, and fuses them for robust detection. Combining both pieces of information has advantages over the sole use of RGB images: 1) the capability of detecting 3D objects with insufficient textures and complex shapes; 2) robust detection under varying lighting conditions; 3) better identification of a target based on its size. Our approach is inspired by a recent work on template-based detection, and we show how to extend it with depth information, which results in better detection performance under varying lighting conditions. Intensive computations are parallelized on a GPU to achieve real-time speed, and it takes only about 33 milliseconds for detection and pose estimation. The proposed method can be used for marker-less AR applications using real-world 3D objects, beyond conventional planar target objects.

1 Introduction

Computer vision-based target detection techniques have been studied extensively and have been applied successfully to markerless augmented reality (AR) applications. A planar object has often been used as a tracking target due to its simple geometry [4, 13]. Results of recent work have shown that 3D objects can be used in AR applications with primitive-based modeling [12].

Local feature descriptors have been shown to have good performance for target detection [2, 9, 16] and hence they have been widely used for target detection. The descriptors are usually computed from local patches centered at keypoints; therefore, these methods can barely handle a textureless object with few keypoints on its surface. On the other hand, depth-based object recognition approaches have focused on a target's geometrical properties, such as normals and curvatures. A target object is identified from depth images by building local feature histograms from those properties [7, 10]. Local feature descriptors have also been introduced to depth-based object recognition [3, 15, 8, 1]. As RGB-D cameras, which capture color and depth images, become widespread recently, the RGB and depth information have been considered together for object recognition and pose estimation. In [5], depth information was employed to reduce influences from background and occlusion, but it was not explicitly used in object recognition. [14] reported that combination of visual features and shapes achieved higher performance and individual cues.

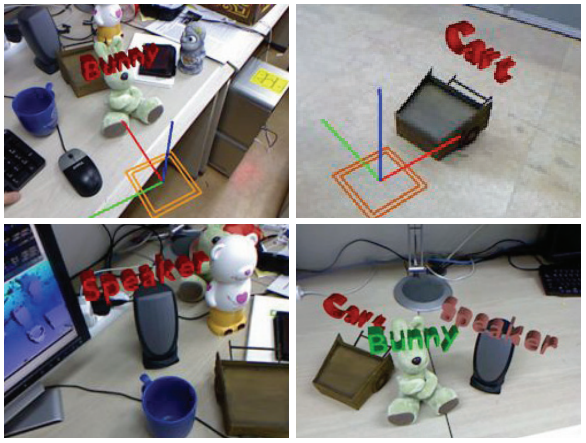


Figure 1: Detection and pose estimation of a 3D object. Our method can detect and estimate the pose of a 3D object. It also provides occlusion between the real and the virtual object based on depth information.

In this paper, we propose a novel method of object detection that can handle 3D target objects, regardless of their texture and lighting condition changes. As shown in Figure 1, our method can deal with a 3D object with a complex shape and insufficient textures in real-time. To do that, we combined information from both RGB and depth images to exploit both the texture and the geometry of a target. We modified a recent template-based detection method [11]. A set of reference templates of a target is computed not only from RGB images but also from depth images to reflect the target's textures and geometrical properties. In runtime, the reference templates are compared with incoming RGB and depth images, to identify a target object. Once the target is detected, its pose is estimated by aligning the 3D points of the current depth image with those of the reference templates.

By combining the texture and geometry information, a 3D object can be detected robustly under changing lighting conditions, and the two objects that have similar shapes and textures but different sizes can also be distinguished from each other. Both detection and pose estimation are computationally expensive and can barely run in real-time on a CPU. We adapted them for the use on a graphics processing unit (GPU) to achieve real-time speed. In our implementation, the overall detection and pose estimation take approximately 33 milliseconds.

In the remainder of the paper, we provide background information in Section 2 and describe our approach to target detection and pose estimation in Section 3. Experimental results are presented in Section 4. Finally, we offer our conclusions in Section 5.

2 Background

2.1 Overview

Figure 2, shows typical examples of when the sole use of an RGB image fails to detect a specific target. In Figure 2(a), an object with

[*]This work was supported in part by the Global Frontier R&D Program on <Human-centered Interaction for Coexistence> funded by the National Research Foundation of Korea grant funded by the Korean Government(MEST) (NRF-M1AXA003-20100029751), and in part by Ministry of Culture, Sports and Tourism(MCST) and Korea Creative Content Agency(KOCCA) in the Culture Technology(CT) Research & Development Program 2011.

[†]e-mail: wlee@gist.ac.kr
[‡]e-mail: npark@gist.ac.kr
[§]e-mail: wwoo@gist.ac.kr

The 21st International Conference on Artificial Reality and Telexistence
November 28-30, 2011, Osaka, Japan
ISSN: 1345-1278 © 2011 The Virtual Reality Society of Japan

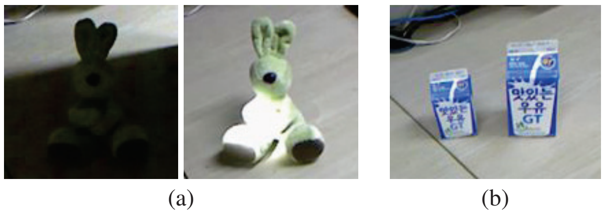


Figure 2: Examples where the sole use of an RGB image is unsuccessful in detection: (a) a target object is under different lighting conditions; (b) two objects have almost the same shapes and textures, but their sizes are different.

insufficient texture is under very different lighting conditions. It is difficult to handle these cases using RGB images only because the target's appearances look very different. The objects shown in Figure 2(b) have almost the same shapes and textures, while they have different sizes. It is also difficult to identify one of them solely using RGB images.

Thus, we attempted to overcome these limitations in 3D target detection by combining the shape and texture information. We modified a recent template-based detection method [11] to consider the texture and shape of a 3D object for robust detection We compute a set of reference templates from both RGB and depth images taken from different viewpoints. The gradients in both images are considered for template computation; the reference templates are built from the gradients with large magnitudes or frequent appearances in local patches. In runtime, the same gradient features are computed from incoming RGB and depth images and compared with the reference templates. As a result, a target's ID and 2D location in both images are retrieved, and a reference template with the greatest similarity is chosen as a match. Then, the detected target's pose is estimated by aligning the 3D points that are computed from the incoming depth image and those of the reference template chosen as a match. After point registration, the target's identity is finally verified from the registration error. Computationally expensive procedures in detection and pose estimation are parallelized by GPU programming for real-time speed. Figure 3 shows the overall procedure of our method.

2.2 RGB-Depth Calibration

The RGB-D camera we used consists of two cameras: an RGB camera for capturing color images and a depth camera for capturing depth images. The depth camera provides depth information as discretized values in a certain range, rather than actual distances. Depth-to-distance calibration is required to convert raw depth values into real distances before using depth information. On the other hand, the RGB and depth cameras have different characteristics, such as field of views and focal lengths; consequently, two pixels at the same location in RGB and depth images do not correspond to the same location in a scene. Thus, the RGB and depth images should be aligned to determine the depth of pixels in the RGB image. We first calibrated the RGB and depth cameras through a typical camera calibration method [19] using a chessboard pattern, and as a result, intrinsic parameter matrices of RGB and depth cameras, K_c and K_d, were estimated. Then, we performed the depth-to-distance calibration and the RGB-depth alignment.

Depth-to-distance calibration: We captured RGB and depth images of the chessboard pattern from different viewpoints, changing the distance between the pattern and the camera from 50 centimeters to 2.5 meters. Then, the actual depth values of the corners of the chessboard pattern were computed from extrinsic parameters. The raw depth values at the corner locations in the depth images were also collected. Finally, we estimated a polynomial of the 6-th degree from the collected data to represent a mapping between the raw depth values and the real distances.

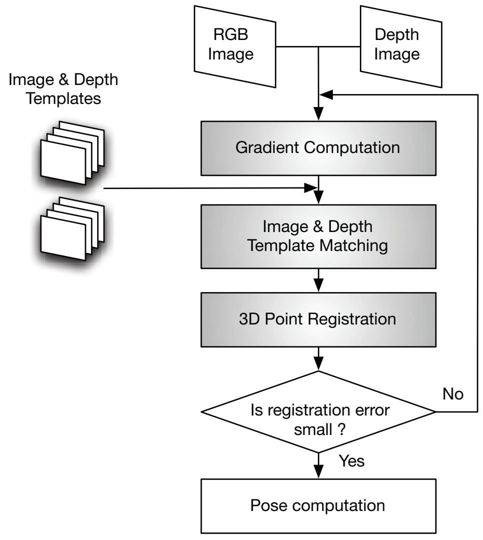


Figure 3: Overall procedure of the proposed method. The steps marked in shade runs in parallel on a GPU.

RGB-depth alignment: Let us denote by $M_{d\to c}$ the relative transformation between the RGB camera and the depth camera. The depth of a pixel in an RGB image is computed as follows. First, a pixel, $\mathbf{p}_d$, in the depth image is back-projected based on its depth value, z_p, to calculate a corresponding 3D point, $\mathbf{P}_d$.

$$\mathbf{P}_d = z_p K_d^{-1} \mathbf{p}_d \tag{1}$$

Then, P_d is transformed to the RGB camera's coordinate frame and projected onto the RGB image plane.

$$\mathbf{P}_c = M_{d\to c}\mathbf{P}_d \tag{2}$$
$$\mathbf{p}_c = K_c \mathbf{P}_c \tag{3}$$

Finally, the depth of a pixel at $\mathbf{p}_c$ in the RGB image is determined as the depth of $\mathbf{P}_c$. If two or more $\mathbf{P}_d$s correspond to the same $\mathbf{p}_c$, the closest one is chosen. As a result of the RGB-depth alignment, we obtain a depth image whose pixels correspond to the RGB image's [1].

3 Depth-assisted 3D Object Detection and Pose Estimation

3.1 Definitions

We define $\mathcal{O}_I$ and $\mathcal{O}_D$ as a target's reference patches taken from an RGB image and a depth image, respectively [2]. We denote by $\mathcal{T}_I$ and $\mathcal{T}_D$ the templates computed from $\mathcal{O}_I$ and $\mathcal{O}_D$, respectively. We call $\mathcal{T}_I$ 'image template' and $\mathcal{T}_D$ 'depth template'.

When computing a template, $\mathcal{T}$, related to a specific viewpoint, we also store its pose, $\mathcal{H}$, and depth. $\mathcal{T}$ is defined as

$$\mathcal{T} = \{\mathcal{T}_I, \mathcal{T}_D, \mathcal{H}, \mathcal{O}_D\}\ , \tag{4}$$

[1]In the remainder of the paper, 'depth image' means an aligned depth image instead of a raw depth image.

[2]Although we use grayscale images instead of color images, we keep calling it 'RGB image' to distinguish it from a depth image.

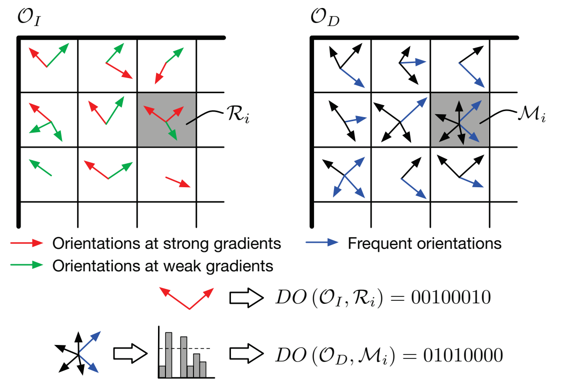

$\mathcal{O}_I$
$\mathcal{O}_D$
$\mathcal{R}_i$
$\mathcal{M}_i$
Orientations at strong gradients
Orientations at weak gradients
Frequent orientations
$DO(\mathcal{O}_I, \mathcal{R}_i) = 00100010$
$DO(\mathcal{O}_D, \mathcal{M}_i) = 01010000$

Figure 4: Building the image and depth templates from gradients

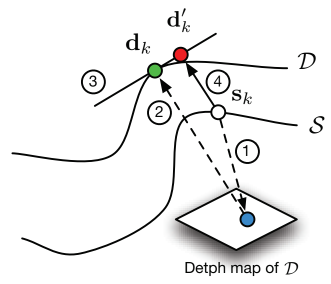


Figure 5: Point-to-plane registration: 1) a source point, $\mathbf{s}_k$ is projected onto the destination depth map; 2) a point on $\mathcal{D}$, $\mathbf{d}_k$, is determined by back-projecting the projection of $\mathbf{s}_k$; 3) a plane tangent to $\mathcal{D}$ on $\mathbf{d}_k$ is computed; 4) a match, $\mathbf{d}'_k$, is determined by projecting $\mathbf{s}_k$ onto the tangent plane.

3.2 Building Templates

Our image and depth templates are inspired by *Dominant Orientation Template* (DOT) algorithm [11]. The image and depth templates are computed from gradients in $\mathcal{O}_I$ and $\mathcal{O}_D$ but they represent different properties of the target object. Figure 4 depicts how image and depth templates are built from gradients.

Image templates are computed in the same way in [11]. The region of $\mathcal{O}_I$ is divided into $m \times n$ subregions, $\mathcal{R}_i$, and the orientations of strong gradients, whose magnitude is larger than a threshold, are collected from each $\mathcal{R}_i$. When gathering orientations, $\mathcal{R}_i$ is translated in a certain range to improve the robustness of detection to small deformations [11]. The collected orientations in $\mathcal{R}_i$ are denoted by $DO(\mathcal{O}_I, \mathcal{R}_i)$. The orientations of gradients are discretized to 7 bins to represent $DO(\mathcal{O}_I, \mathcal{R}_i)$ as an 8-dimensional binary vector. Each element of the vector indicates that strong gradients exist in a specific direction. The remaining 8-th element is set to 1 if there are no strong gradients in $\mathcal{R}_i$. Finally, an image template, $\mathcal{T}_I$, is represented as:

$$\mathcal{T}_I = \{DO(\mathcal{O}_I, \mathcal{R}_i) \,|\, \mathcal{R}_i \in \mathcal{O}_I\} \tag{5}$$

A depth template $\mathcal{T}_D$ is computed in a similar way as $\mathcal{T}_I$, but we consider the orientations that appear frequently in $\mathcal{O}_D$, rather than those of strong gradients because we are interested in the geometrical changes on the target's surface. In a depth image, strong gradients usually appear on the boundary between a target and the background, and thus, strong gradients barely provide useful information about the target's surface.

Given a depth patch, $\mathcal{O}_D$, and its subregions, $\mathcal{M}_i$, we build a histogram by accumulating orientations of gradients in $\mathcal{M}_i$. Orientations are discretized to 7 bins, and $\mathcal{M}_i$ is also translated to improve the robustness to small deformations, as we do for an image template. The histogram is then binarized by applying a threshold δ_h. Typically, δ_h is set to 30% of the number of subregions in $\mathcal{O}_D$. $\mathcal{T}_D$ is represented as an 8-dimensional vector from the resulting binary values. Each element indicates that a specific orientation appears frequently in $\mathcal{M}_i$. In case of the depth template, the remaining 8-th element represents the existence of a strong gradient in $\mathcal{M}_i$, where depth information is unreliable. A depth template, $\mathcal{T}_D$, is defined as:

$$\mathcal{T}_D = \{FO(\mathcal{O}_D, \mathcal{M}_i) \,|\, \mathcal{M}_i \in \mathcal{O}_D\}\,, \tag{6}$$

where $FO(\mathcal{O}_D, \mathcal{M}_i)$ represents orientations that frequently appear in $\mathcal{M}_i$.

3.3 Template matching

When there is an incoming RGB image patch, $\mathcal{P}_I$, the similarity between an image template, $\mathcal{T}_I$, and $\mathcal{P}_I$ is defined as

$$Sim(\mathcal{P}_I, \mathcal{T}_I) = \sum_{\mathcal{R}i} \delta\left(do(\mathcal{P}_I, \mathcal{R}_i), DO(\mathcal{O}_I, \mathcal{R}_i)\right)\,, \tag{7}$$

where $do(\mathcal{P}_I, \mathcal{R}_i)$ computes only the orientation of the strongest gradient in $\mathcal{R}_i$. The function $\delta(\cdot)$ is an element-wise *AND* operation between two 8-dimensional vectors.

The similarity between a depth image patch, $\mathcal{P}_D$, and a depth template, $\mathcal{T}_D$, is defined as

$$Sim(\mathcal{P}_D, \mathcal{T}_D) = \sum_{\mathcal{M}_i} \delta\left(fo(\mathcal{P}_D, \mathcal{M}_i), FO(\mathcal{O}_D, \mathcal{M}_i)\right), \tag{8}$$

where $fo(\mathcal{P}_D, \mathcal{M}_i)$ computes the most frequent orientation in a region $\mathcal{M}_i$.

From both similarity measures, the similarity function between a 2-tuple patch $\mathcal{P} = (\mathcal{P}_I, \mathcal{P}_D)$ and a template, $\mathcal{T}$, is defined as

$$Sim(\mathcal{P},) = (1-\alpha)\,Sim(\mathcal{P}_I, \mathcal{T}_I) + \alpha Sim(\mathcal{P}_D, \mathcal{T}_D)\,, \tag{9}$$

where α controls the weights of similarity values. In practice, we set $\alpha = 0.4$, which was experimentally determined.

All possible image regions are compared with the reference templates by changing the location of $\mathcal{P}$, and the most similar template is chosen as a match:

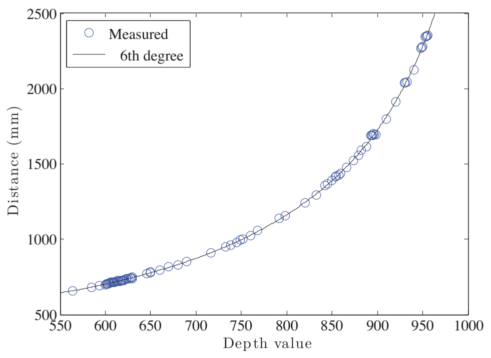
$$(\mathcal{P}^*, \mathcal{T}^*) = \underset{i,j}{\mathrm{argmax}} Sim\left(\mathcal{P}^i, \mathcal{T}^j\right)\,, \tag{10}$$

where $\mathcal{P}^i$ and $\mathcal{T}^j$ represent a 2-tuple patch at i-th locations of RGB and depth images and the reference template corresponding to the j-th viewpoint, respectively.

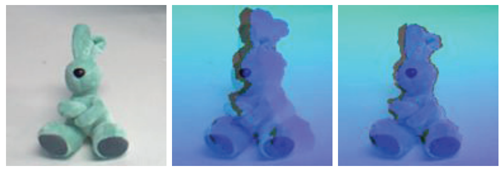
As a result of the template matching, the location of a target, $\mathcal{P}^* = (\mathcal{P}_I^*, \mathcal{P}_D^*)$, and a matching reference template $\mathcal{T}^*$ are obtained.

3.4 Pose Estimation of a 3D Object

We assume the target's pose is initially identical to the pose of $\mathcal{T}^*$. The initial pose is refined by aligning 3D points computed from the current depth region, $\mathcal{P}_D^*$, with those of the reference template, $\mathcal{T}^*$. We adopt the point-to-projection approach for point registration because it is faster than the other iterative closest points (ICP) methods [18]. Let us denote by $\mathcal{S}$ the 3D points on a source surface, $\mathcal{D}$ a 3D points on the destination surface, and $\Delta W = [\Delta R|\Delta t]$ an incremental transformation between the source surface and the destination surface. In our problem, $\mathcal{S}$ and $\mathcal{D}$ correspond to the 3D



(a)



(b)

Figure 6: RGB-depth calibration: (a) Mapping between the raw depth values and real distances through the 6-th order polynomial; (b) RGB-depth alignment result (from left to right: RGB image, raw depth image, and aligned depth image).

points of $\mathcal{T}^*$ and those of $\mathcal{P}_D^*$, respectively. The initial pose is $\mathcal{H}^*$ of $\mathcal{T}^*$ and ΔW is initialized as $[I|0]$.

In the point-to-projection algorithm, a point $\mathbf{s}_k$ of $\mathcal{S}$ is projected onto the depth map of $\mathcal{D}$. The projection of $\mathbf{s}_k$ is back-projected, and a 3D point, $\mathbf{d}_k$, on $\mathcal{D}$ is determined from the depth map. Then, a plane that is tangent to $\mathcal{D}$ on $\mathbf{d}_k$ is computed from $\mathbf{d}_k$'s normal vector. Finally, $\mathbf{s}_k$ is projected onto the tangent plane, and its projection, $\mathbf{d}'_k$, is considered as a match. Figure 5 illustrates this procedure. After matches for all $\mathbf{s}_k$s are found, we obtain a new surface $\mathcal{D}'$ consisting of $\mathbf{d}'_k$s.

A correlation matrix, H, relating $\mathcal{S}$ and $\mathcal{D}'$ is defined as

$$H = \sum_k \left(\mathbf{d}'_k - \mathbf{c}_{d'}\right)\left(\mathbf{s}_k - \mathbf{c}_s\right)^T \ , \tag{11}$$

where $\mathbf{c}_s$ and $\mathbf{c}_{d'}$ represent the centroids of $\mathcal{S}$ and $\mathcal{D}'$, respectively.

ΔW between $\mathcal{S}$ and $\mathcal{D}'$ is computed from H:

$$\Delta R = VU^\top \tag{12}$$

$$\Delta t = \mathbf{c}_s - R\mathbf{c}_{d'} \ , \tag{13}$$

where V and U are determined by applying singular value decomposition (SVD) to H ($H = U\Sigma V^T$).

By iteratively estimating and accumulating the incremental transformation, ΔW, the refined pose after the i-th iteration, W_i, is computed as

$$W_i = \Delta R_i \Delta R_{i-1} + \Delta R_i \Delta t_{i-1} + \Delta t_i \ . \tag{14}$$

The iteration is terminated if the difference between W_{i-1} and W_i is small.

The target's identity is verified based on the registration error defined as the average distance between points on $\mathcal{S}$ and $\mathcal{D}'$. If the registration error is smaller than a threshold, detection is considered successful. Typically, the threshold is set to 7 millimeters in our experiments.

3.5 Parallelization on a GPU

To increase the speed of detection, we parallelized the gradient computation and template comparison steps. As pixel- or patch-wise operations, they are suitable for running in parallel on a GPU.

Gradient computation consists of two steps. First, the magnitude and the orientation of each pixel's gradient is computed from both RGB and depth images. In the GPU, a thread is assigned to a pixel location, and gradients are computed by a 3×3 Sobel mask in parallel. Then, $do(\cdot)$ and $fo(\cdot)$ are computed from the gradients in RGB and depth patches that would be compared with the reference templates. In the second step, each region is independently processed in a thread. The resulting $do(\cdot)$ and $fo(\cdot)$ are stored as an array of 2D vectors in the GPU's memory to use them in template matching on the GPU.

When conducting template matching on the GPU, the reference templates of a target are copied to the GPU's memory as a 2D image block, where each row corresponds to a template related to a viewpoint. A thread is assigned to an image region and compares the previously computed orientations, $do(\cdot)$ and $fo(\cdot)$, with the reference templates. The threads in the same thread block are synchronized and they access the same reference template concurrently. After all of the reference templates are compared, the index of the most similar reference template and the similarity value are stored as a 2D vector in the memory for each image region.

Pose estimation step was also implemented on the GPU for real-time speed because point registration is computationally expensive and can barely run in real-time on a CPU. We parallelized two steps of the registration procedure, finding matches and computing the correlation matrix, H.

When computing point matches, each match is computed independently; therefore, it is also good to be parallelized. The source points converted to 3-channels image and the depth map of the destination surface are copied to the GPU's memory. In the GPU, a thread is assigned to each source point, $\mathbf{s}_k$, and the matches to $\mathbf{s}_k$s are computed in parallel. The resulting matches, $\mathbf{s}_k$s and $\mathbf{d}'_k$s, are kept in the GPU for computing the correlation matrix, H. Summation is a major operation in computing the centroids, $\mathbf{c}_{d'}$ and $\mathbf{c}_s$, and in calculating the correlation matrix H from the centroids as well. Summations in those computations were accelerated by a memory reduction technique [17]. The reduction consists of two steps. In the first step, we launch 100 thread blocks, each containing 256 threads. The summation of the 256 elements is computed in each block. Then, in the second step, 100 threads are launched in a block to compute the final sum from the 100 partial sums.

4 Experimental Results

4.1 Setup

We performed experiments on a PC with a 2.93GHz CPU and an NVIDIA Geforce GTX580 GPU. For GPU programming, we used NVIDIA's CUDA. The Microsoft KINECT sensor was employed to capture RGB and depth images (in 640×480 at 30 Hz). The size of a reference template was 154×154 and the number of total templates was 256. The size of a subregion was 7×7 as it was reported that good performance was achieved with it [11]. The maximum number of iterations in the point registration step was 50.

Reference template acquisition When building reference templates from a 3D object, we placed the target object on a flat surface. A simple background subtraction method was applied to identify the image regions that belonged to the object. We took image patches from different viewpoints and computed the image and depth templates from them. The poses of the reference templates were computed from a known planar tracking target, which was located beside the object and used as a reference coordinate frame.

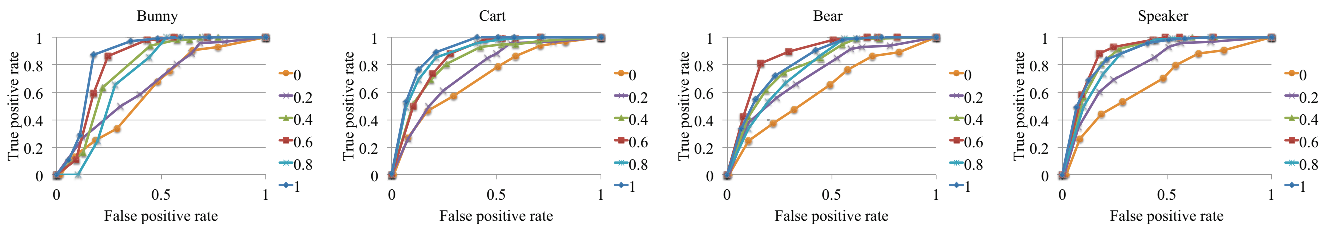


Figure 7: ROC curve with the varying scale factor α.

RGB-Depth calibration Figure 6(a) shows the relationship between the depth values of a raw depth image and those computed from extrinsic parameters. The 6-th order polynomial function estimated from them represents the relationship between raw depth values and actual distances quite well. Note that the estimated mapping function is effective only in the distance range where the data were obtained. In Figure 6(b), the result of RGB-depth alignment is depicted. The raw depth image was misaligned with the RGB image due to the parallax and different characteristics of the cameras. After the alignment, we acquired a new depth image that was correctly aligned with the RGB image.

4.2 Results

Fig 7 shows detection performance with varying α values. We recorded a video sequences under a varying lighting conditions and measured the performance. The α varied from 0 to 1 by the step of 0.2. When using depth only ($\alpha = 0$), detection performance was worse than using RGB only, and many false detection cases were observed. Thus, the depth template is not much discriminate. However, when both information is fused, it was comparable or better than single cue cases, i.e., using RGB or depth only. Depending on the target object, the best result is achieved with differnt alpha values ranging from 0.4 to 0.6.

We show the target detection results under changing lighting conditions in Figure 9. Detection was unsuccessful when using RGB images only, because gradients of the target's textures largely changed due to spotlights and strong shadows. In case of the RGB-only detection, the similarity values decreased when the spotlights were projected or the shadows were drawn. In contrast, our method detected the targets successfully by taking advantages from the depth information, which is not disturbed by changing lighting conditions. The similarity values were maintained high when the depth information was combined with the RGB information.

Thanks to the depth information, the two objects that had almost the same shape and textures but different sizes could also be distinguished successfully as depicted in Figure 8. RGB-only detection failed to identify one of them due to their similar textures, and hence, false detection and wrong pose estimation occurred. More target detection and augmentation results are shown in Figure 10. Most of targets have poor textures (e.g., *Speaker*, *Building*, *Cart*, and *Bunny*). Our method successfully detected them and estimated their poses. It also worked well with a target with sufficient textures as shown in row 4 (*Harubang*). The last row demonstrates that the capability of multiple target detection.

Table 1 shows the overall speed of template matching running on both a CPU and a GPU. In the case of CPU implementation, template clustering was applied to increase the speed of template matching as in [11]. As we can see, template matching ran much faster on the GPU than on the CPU. In the template comparison step, the GPU version was approximately 4 times faster than the CPU version. Without template clustering, template matching took about 80 ms on the CPU. In the 3D registration, the GPU implementation was approximately 25 times faster than the CPU version

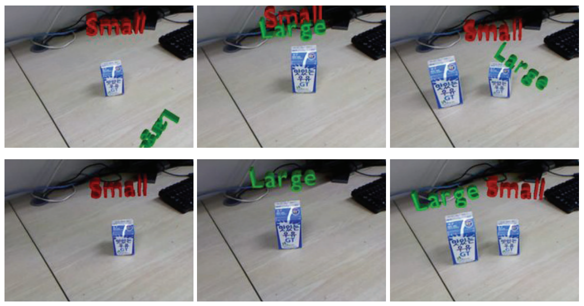


Figure 8: Detection of targets that have almost the same textures and shapes: (top row) False detection occurs when using RGB images only, and hence the estimated poses are wrong; (bottom row) two objects are identified correctly by our method.

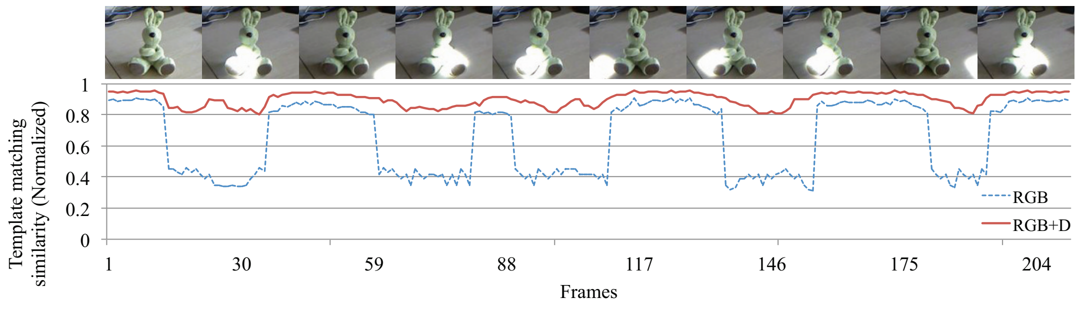
(see Table 2). A major speed improvement was achieved in the point match step. The speed of correlation matrix computation was also improved on the GPU, but it became slower than the point match step due to multiple launches of GPU kernels for reduction. Note that the $[R|t]$ computation was conducted on the CPU, and consequently, its speed was the same on both cases. The overall speed of single target detection and pose estimation was 33.5 ms, which was adequate for real-time AR applications. In the case of multiple targets, our method was able to handle up to 3 targets in real-time.

The accuracy of point registration was measured under smooth camera motions by computing the average distance between points of the source and destination surfaces. As we can see in Table 3, the registration quality was good and the pose estimation result could be used for overlaying virtual objects on video sequences. In our experiments, 3D point registration tended to be more accurate with the objects having planar geometries (e.g., *Cart* and *Building*), than those with more complex shapes (e.g., *Bunny*, *Harubang*, and *Bear*).

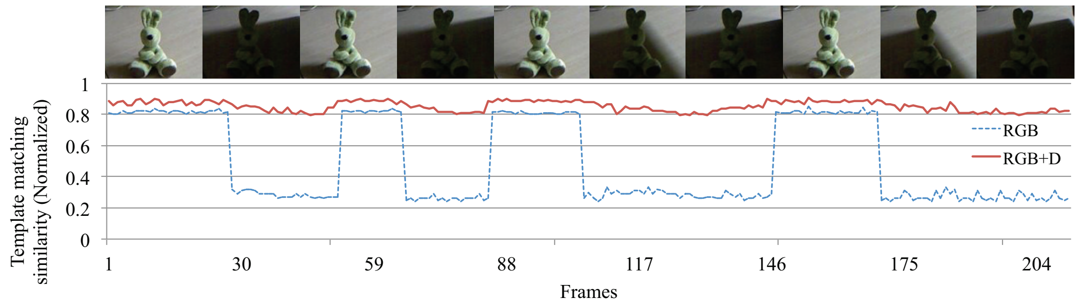
The RGB-D camera used in our experiments was unable to compute the depth of a location that was excessively close to the camera. The distance between the camera and a target object should be more than approximately 60 centimeters. Depth information provided by the camera was noisy, which caused jittering in the estimated pose. Applying noise reduction filters [6] can be a possible solution to reduce the noise. On the other hand, the depth information became inaccurate and unstable if a target had thin structures; therefore, it was difficult to estimate the pose of such a target.

5 Conclusions and Future Works

We proposed a novel 3D target detection method that exploits both the target's texture and geometry information. Our method can handle 3D objects with complex shapes and insufficient textures, and detects targets under changing lighting conditions by combining information from RGB and depth images. Real-time speed



(a)



(b)

Figure 9: Template matching results under changing lighting conditions: (a) the target is under a moving spotlight; (b) the target is under shadow.

is achieved by parallelizing computationally expensive steps on a GPU. In the current method, jitters occur in the estimate pose when a target's surface is occluded by other objects. Consequently, we will focus on stable pose estimation under partial occlusion in the future.

Table 1: Template matching speed on a CPU and a GPU (unit: ms)

Procedure	CPU	GPU
Gradient computation	10.7	0.5
Template comparison	39	10.2
Total	49.7	10.7

Table 2: 3D Registration speed on a CPU and a GPU (unit: ms)

Procedure	CPU	GPU
Point match	422.8	6.3
Correlation matrix computation	160	15.6
$[R\|t]$ computation	0.9	0.9
Total	583.7	22.8

Table 3: Point registration error (unit: millimeters)

Object	Mean	Stdev
Speaker	3.75	0.43
Building	3.13	0.44
Cart	2.15	0.54
Bunny	4.49	0.55
Harubang	3.89	0.32
Bear	4.21	0.29

References

[1] P. Bariya and K. Nishino. Scale-hierarchical 3d object recognition in cluttered scenes. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 1657–1664, 2010.

[2] H. Bay, A. Ess, T. Tuytelaars, and L. VanGool. Speeded-Up Robust Features. *Computer Vision and Image Understanding*, 110(3):346–359, June 2008.

[3] N. Bayramoglu and A. A. Alatan. Shape index sift: Range image recognition using local features. In *Proceedings of the International Conference on Pattern Recognition*, pages 352–355, 2010.

[4] S. Benhimane and E. Malis. Homography-Based 2d Visual Tracking and Servoing. *International Journal of Robotics Research*, 26(7):661–676, 2007.

[5] N. Burrus, M. Abderrahim, J. Garcia, and L. Moreno. Object reconstruction and recognition leveraging an rgb-d camera. In *The 12th IAPR Conference on Machine Vision Applications*, June 2011 (in press).

[6] D. Chan, H. Buisman, C. Theobalt, and S. Thrun. A Noise-Aware Filter for Real-Time Depth Upsampling. In *Workshop on Multi-camera and Multi-modal Sensor Fusion Algorithms and Applications 2008*, 2008.

[7] H. Chen and B. Bhanu. 3d free-form object recognition in range images using local surface patches. *Pattern Recogn. Lett.*, 28:1252–1262, July 2007.

[8] B. Drost, M. Ulrich, N. Navab, and S. Llic. Model globally, match locally: Efficient and robust 3d object recognition. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 998–1005, 2010.

[9] D. G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[10] G. Hetzel, B. Leibe, P. Levi, and B. Schiele. 3d object recognition from range images using local feature histograms. In *Proceedings of CVPR 2001*, pages 394–399, 2001.

[11] S. Hinterstoisser, V. Lepetit, S. Ilic, P. Fua, and N. Navab. Dominant Orientation Templates for Real-Time Detection of Texture-Less Objects. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2010.

[12] K. Kim, V. Lepetit, and W. Woo. Keyframe-based Modeling and Tracking of Multiple 3D Objects. In *Proceedings of the International Symposium on Mixed and Augmented Reality*, 2010.

[13] K. Kim, V. Lepetit, and W. Woo. Scalable real-time planar targets tracking for digilog books. *Vis. Comput.*, 26:1145–1154, June 2010.

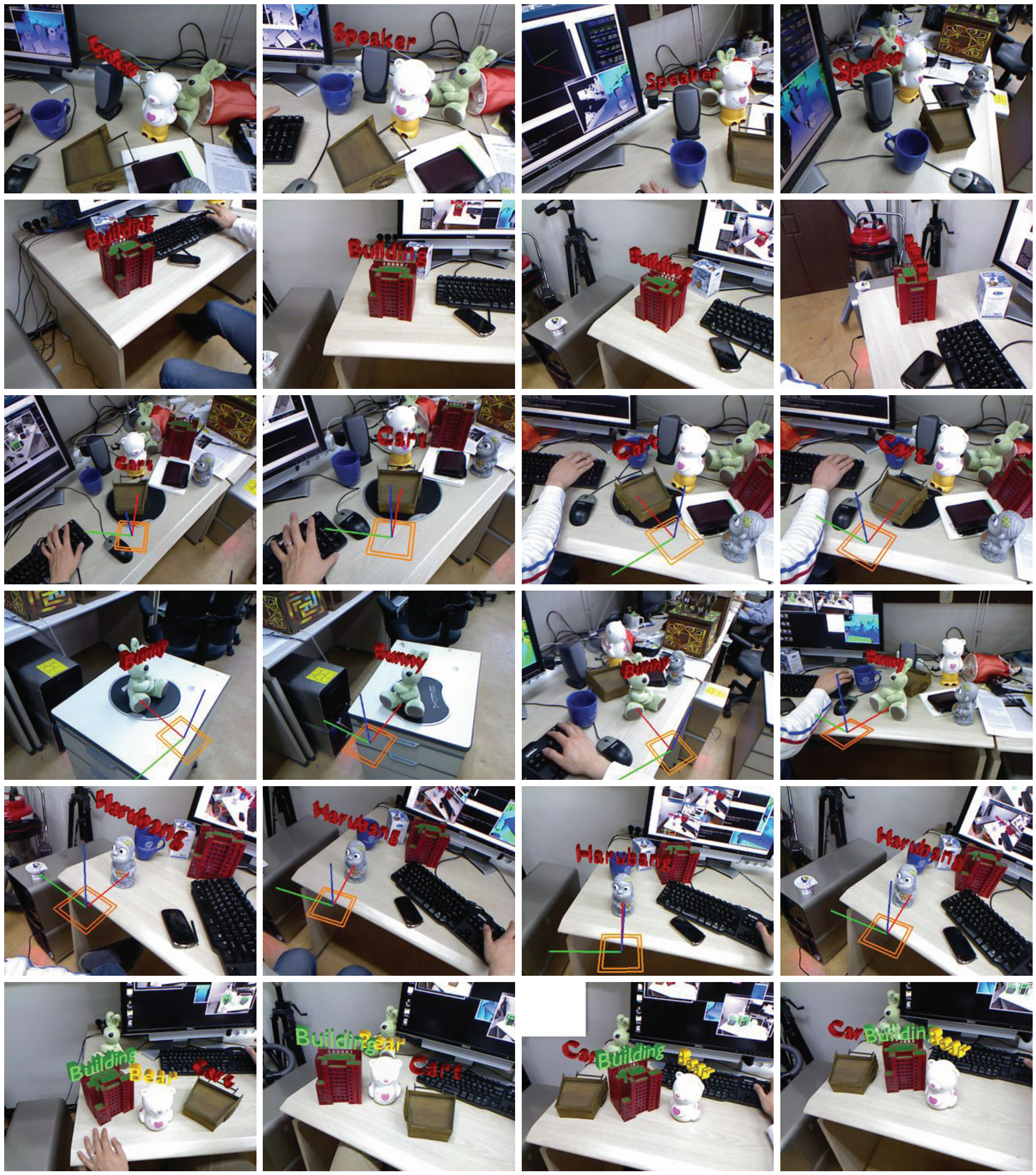


Figure 10: Target detection examples: (from rows 1 to 5) *Speaker*, *Building*, *Cart*, *Bunny*, *Harubang*; (row 6) multiple target detection. Our method handles the targets with complex shapes and insufficient textures successfully.

[14] K. Lai, L. Bo, X. Ren, and D. Fox. Sparse distance learning for object recognition combining rgb and depth information. In *IEEE International Conference on Robotics and Automation*, 2011.

[15] T.-W. R. Lo and J. P. Siebert. Local feature extraction and matching on range images: 2.5d sift. *Comput. Vis. Image Underst.*, 113:1235–1250, December 2009.

[16] M. Ozuysal, M. Calonder, V. Lepetit, and P. Fua. Fast Keypoint Recognition Using Random Ferns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(3):448–461, March 2010.

[17] D. Roger, U. Assarsson, and N. Holzschuch. Efficient stream reduction on the GPU. In *1st Workshop on General Purpose Processing on Graphics Processing Units (GPGPU)*, 2007.

[18] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *Third International Conference on 3D Digital Imaging and Modeling (3DIM)*, pages 145–152, June 2001.

[19] Z. Zhang. Flexible camera calibration by viewing a plane from unknown orientations. In *Computer Vision, IEEE International Conference on*, volume 1, page 666, 1999.