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, 11]. 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]

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.

1 e-mail: wlee@gist.ac.kr
2 e-mail: npark@gist.ac.kr
3 e-mail: wwoo@gist.ac.kr

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.

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 typical examples.
as a result, intrinsic parameter matrices of RGB and depth cameras, 
cameras calibration method [19] using a chessboard pattern, and 
age. We first calibrated the RGB and depth cameras through a typi-
should be aligned to determine the depth of pixels in the RGB im-
the RGB and depth images do not correspond 
such as field of views and focal lengths; consequently, two pixels 
hand, the RGB and depth cameras have different characteristics, 
uclei into real distances before using depth information. On the other 
ning depth images. The depth camera provides depth information as 
The RGB-D camera we used consists of two cameras: an RGB 
2.2 RGB-Depth Calibration 
The RGB-D camera we used consists of two cameras: an RGB 
the texture and shape of a 3D object for robust detection We com-
pute 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 appear-
ances 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 1D and 2D 
location in both images are retrieved, and a reference template with 
the greatest similarity is chosen as a match. Then, the detected tar-
get’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 expen-
ensive 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 captur-
ning 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 val-
ues 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 im-
age. We first calibrated the RGB and depth cameras through a typi-
cal camera calibration method [19] using a chessboard pattern, and 
as a result, intrinsic parameter matrices of RGB and depth cameras, 

\[ K_r, K_d \], 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, chang-
ing the distance between the pattern and the camera from 50 cen-
timeters 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.

RGB-depth alignment: Let us denote by \( M_{d \rightarrow c} \) 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, \( p_d \), in the depth image is back-projected based on its depth 
value, \( z_p \), to calculate a corresponding 3D point, \( P_d \).

\[
P_d = z_p K_d^{-1} p_d \quad (1)
\]

Then, \( P_d \) is transformed to the RGB camera’s coordinate frame 
and projected onto the RGB image plane.

\[
P_c = M_{d \rightarrow c} P_d \quad (2)
\]

\[
p_c = K_c P_c \quad (3)
\]

Finally, the depth of a pixel at \( p_c \) in the RGB image is determined 
as the depth of \( P_c \). If two or more \( P_d \)s correspond to the same 
\( p_c \), the closest one is chosen. As a result of the RGB-depth align-
ment, 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 \( O_I \) and \( O_D \) as a target’s reference patches taken from 
an RGB image and a depth image, respectively 2. We denote by \( T_I \) 
and \( T_D \) the templates computed from \( O_I \) and \( O_D \), respectively. 
We call \( T_I \) ‘image template’ and \( T_D \) ‘depth template’.

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

\[
T = \{ T_I, T_D, \mathcal{H}, O_D \} \quad (4)
\]

1In the remainder of the paper, ‘depth image’ means an aligned depth 
image instead of a raw depth image.

2Although we use grayscale images instead of color images, we keep 
calling it ‘RGB image’ to distinguish it from a depth image.
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 $O_I$ and $O_D$, 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 $O_I$ is divided into $m \times n$ sub-regions, $R_i$, and the orientations of strong gradients, whose magnitude is larger than a threshold, are collected from each $R_i$. When gathering orientations, $R_i$ is translated in a certain range to improve the robustness of detection to small deformations [11]. The collected orientations in $R_i$ are denoted by $DO(O_I, R_i)$. The orientations of gradients are discretized to 7 bins to represent $DO(O_I, R_i)$ 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 $R_i$. Finally, an image template, $T_i$, is represented as:

$$T_i = \{DO(O_I, R_i) | R_i \in O_I\}$$  \hspace{1cm} (5)

A depth template $T_D$ is computed in a similar way as $T_i$, but we consider the orientations that appear frequently in $O_D$, 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, $O_D$, and its subregions, $M_i$, we build a histogram by accumulating orientations of gradients in $M_i$. Orientations are discretized to 7 bins, and $M_i$ 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 $\delta_{D}$. Typically, $\delta_{D}$ is set to 30% of the number of subregions in $O_D$. $T_D$ is represented as an 8-dimensional vector from the resulting binary values. Each element indicates that a specific orientation appears frequently in $M_i$. In case of the depth template, the remaining 8-th element represents the existence of a strong gradient in $M_i$, where depth information is unreliable. A depth template, $T_D$, is defined as:

$$T_D = \{FO(O_D, M_i) | M_i \in O_D\}$$ \hspace{1cm} (6)

where $FO(O_D, M_i)$ represents orientations that frequently appear in $M_i$.

3.3 Template matching

When there is an incoming RGB image patch, $P_I$, the similarity between an image template, $T_i$, and $P_I$ is defined as

$$Sim(P_I, T_i) = \sum_{R_i} \delta(\text{do}(P_I, R_i), DO(O_I, R_i))$$ \hspace{1cm} (7)

where $\text{do}(P_I, R_i)$ computes only the orientation of the strongest gradient in $R_i$. The function $\delta(\cdot)$ is an element-wise AND operation between two 8-dimensional vectors.

The similarity between a depth image patch, $P_D$, and a depth template, $T_D$, is defined as

$$Sim(P_D, T_D) = \sum_{M_i} \delta(\text{do}(P_D, M_i), FO(O_D, M_i))$$ \hspace{1cm} (8)

where $\text{do}(P_D, M_i)$ computes the most frequent orientation in a region $M_i$.

From both similarity measures, the similarity function between a 2-tuple patch $P = (P_I, P_D)$ and a template, $T_i$, is defined as

$$Sim(P, T_i) = (1-\alpha) Sim(P_I, T_i) + \alpha Sim(P_D, T_D)$$ \hspace{1cm} (9)

where $\alpha$ 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 $P$, and the most similar template is chosen as a match:

$$(P^*, T^*) = \argmax_{i,j} Sim(P^i, T^j)$$ \hspace{1cm} (10)

where $P^i$ and $T^j$ 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, $P^* = (P^*_I, P^*_D)$, and a matching reference template $T^*$ are obtained.

3.4 Pose Estimation of a 3D Object

We assume the target’s pose is initially identical to the pose of $T^*$. The initial pose is refined by aligning 3D points computed from the current depth region, $P_D$, with those of the reference template, $T^*$. 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 $S$ the 3D points on a source surface, $D$ 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, $S$ and $D$ correspond to the 3D
points of $T^*$ and those of $P^*_D$, respectively. The initial pose is $H^*$ of $T^*$ and $\Delta W$ is initialized as $[I | 0]$.

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

A correlation matrix, $H$, relating $S$ and $D'$ is defined as

$$H = \sum_k (d'_k - c_{d'}) (s_k - c_s)^T,$$

where $c_s$ and $c_{d'}$ represent the centroids of $S$ and $D'$, respectively. $\Delta W$ between $S$ and $D'$ is computed from $H$:

$$\Delta R = VU^T$$

$$\Delta t = c_s - Rc_{d'},$$

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, $\Delta W$, the refined pose after the $i$-th iteration, $W_i$, is computed as

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

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 $S$ and $D'$. 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 $3 \times 3$ 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, $s_k$, and the matches to $s_k$'s are computed in parallel. The resulting matches, $s_k$'s and $d'_k$'s, are kept in the GPU for computing the correlation matrix, $H$. Summation is a major operation in computing the centroids, $c_{d'}$ and $c_s$, 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 $640 \times 480$ at 30 Hz). The size of a reference template was $154 \times 154$ and the number of total templates was 256. The size of a subregion was $7 \times 7$ 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.
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 (α = 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 different 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 gradient 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 (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
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)

<table>
<thead>
<tr>
<th>Procedure</th>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradient computation</td>
<td>10.7</td>
<td>0.5</td>
</tr>
<tr>
<td>Template comparison</td>
<td>39</td>
<td>10.2</td>
</tr>
<tr>
<td>Total</td>
<td>49.7</td>
<td>10.7</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Procedure</th>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point match</td>
<td>422.8</td>
<td>6.3</td>
</tr>
<tr>
<td>Correlation matrix computation</td>
<td>160</td>
<td>15.6</td>
</tr>
<tr>
<td>([R</td>
<td>t]) computation</td>
<td>0.9</td>
</tr>
<tr>
<td>Total</td>
<td>583.7</td>
<td>22.8</td>
</tr>
</tbody>
</table>

Table 3: Point registration error (unit: millimeters)

<table>
<thead>
<tr>
<th>Object</th>
<th>Mean</th>
<th>Stdev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speaker</td>
<td>3.75</td>
<td>0.43</td>
</tr>
<tr>
<td>Building</td>
<td>3.13</td>
<td>0.44</td>
</tr>
<tr>
<td>Cart</td>
<td>2.15</td>
<td>0.54</td>
</tr>
<tr>
<td>Bunny</td>
<td>4.49</td>
<td>0.55</td>
</tr>
<tr>
<td>Harubang</td>
<td>3.89</td>
<td>0.32</td>
</tr>
<tr>
<td>Bear</td>
<td>4.21</td>
<td>0.29</td>
</tr>
</tbody>
</table>

References

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.


