Handling Motion-Blur in 3D Tracking and Rendering for Augmented Reality

Youngmin Park, Vincent Lepetit, and Woontack Woo*, *Member, IEEE*

Abstract—The contribution of this paper is two-fold. First, we show how to extend the ESM algorithm to handle motion blur in 3D object tracking. ESM is a powerful algorithm for template matching-based tracking, but it can fail under motion blur. We introduce an image formation model that explicitly consider the possibility of blur, and show it results in a generalization of the original ESM algorithm. This allows to converge faster, more accurately and more robustly even under large amount of blur. Our second contribution is an efficient method for rendering the virtual objects under the estimated motion blur. It renders two images of the object under 3D perspective, and warps them to create many intermediate images. By fusing these images we obtain a final image for the virtual objects blurred consistently with the captured image. Because warping is much faster than 3D rendering, we can create realistically blurred images at a very low computational cost.

Index Terms—Augmented Reality, Computer Vision, Object Tracking, Object Detection, Motion-Blur, Efficient Second-order Minimization.

1 Introduction

With the rapidly increasing popularity of webcam- and mobile devices-based AR applications, motion blur is one of the main obstacles for vision-based tracking and augmentation. Because it makes salient features almost disappear, motion blur disturbs tracking algorithms based on corners or edges extraction. One solution is to use detection algorithms [1], [2] to restart tracking when the blur stops, but the interruption spoils the user experience.

A method that explicitly handles motion blur is therefore desirable. Recently a few approaches have been introduced but they are either limited to markers [3], [4], or to camera tracking applications [5]. Camera tracking can exploit the whole image, but we mainly focus here on object tracking, which is more challenging since it can rely only on the image portion containing the object. In presence of blur, the amount of image information relevant for tracking can become very low.

Moreover, for Augmented Reality applications, another problem occurs since the virtual objects must be rendered to reflect correctly the motion blur; otherwise they would look unnatural. Previous approaches applied artificial 2D image blurring [6], [7]. However, these do not take perspective into account correctly.

As Figure 1 shows, our contribution is two-fold. We propose a method for object tracking even in presence of large amounts of motion blur, and also a method for rendering blurred virtual objects under general 3D motion. For tracking we explicitly introduce motion blur into the image formation model for template matching. In practice, we chose the Efficient Second-order Minimization (ESM) algorithm because it is one of the most efficient existing algorithm [8], [9]. Our approach yields a very simple to implement generalization of the ESM formula. For rendering we rely on the same image formation model. Our algorithm avoids extensive repetitive rendering required to generate motion blur for 3D motion by combining 2D warping and 3D rendering.

This paper extends our primilary work [10] in both the tracking and rendering aspects. On the tracking side, we relax the need of known exposure time required to simulate the motion blur. We propose a method that automatically adapts to the varying shutter speed, and this results in improved robustness and convergence. We also improved the rendering aspect by adjusting the blending process.

We experimented not only with synthetic images but also with real sequences captured with automatic shutter speed control. In the sequences, the tracked object moves fast and presents heavy motion. We obtain faster, more robust, and more accurate convergence compared to the standard ESM algorithm. In addition, we verified the improvements under varying shutter speed both in synthetic and real images. We also show the generated motion blur reflects successfully the 3D motion.

In the remainder of the paper, we first describe related works about tracking under motion blur and motion blur generation for Augmented Reality. Then, we describe our methods for motion blurred object tracking and motion blur generation and present our results.

- *Y. Park and W. Woo are with U-VR Lab., GIST, Gwangju, S. Korea E-mail: ypark@gist.ac.kr, wwoo@gist.ac.kr*
- *V. Lepetit is with CVLab, EPFL, Lausanne, Switzerland E-mail: vincent.lepetit@epfl.ch*

2 Related Work

Most of the existing tracking methods ignore blur and are therefore prone to fail when the input images are too blurred, but a few methods explicitly consider blur. For marker detection, Okumura et al. first estimate the amount

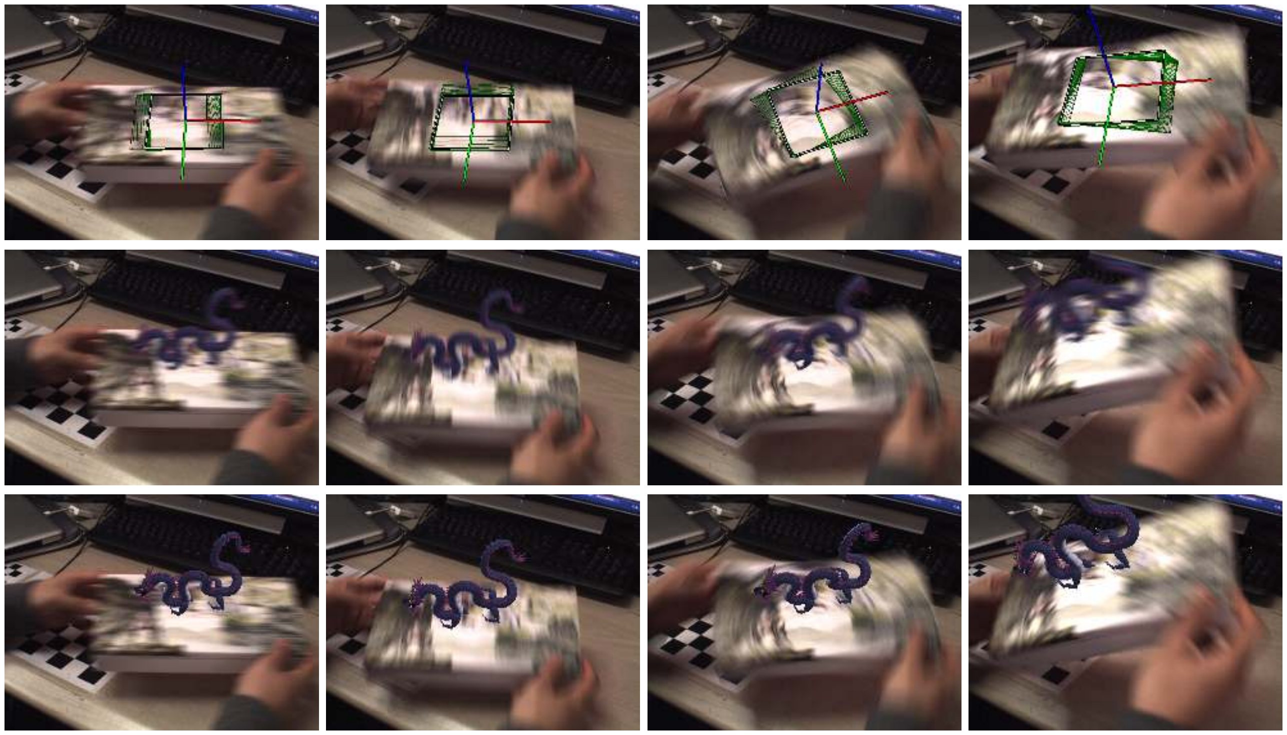


Fig. 1. Tracking and motion blur generation for various 3D motions. **Top row:** Our algorithm can track the selected squared region on the box despite heavy motion blur. The green squares show the successive iteration steps of the motion estimation. **Middle row:** A virtual dragon is rendered with consistent motion blur. Our method properly generates motion blur for general motion, including 3D rotations. **Bottom row:** For comparison, the same object is rendered without motion blur. The insertion is less credible. *Check also the supplementary video.*

of blur to try to remove its effect in the input images and improve the registration [3]. Claus et al. proposed a machine learning approach for a fiducial detection which is robust to motion blur as well as other real world difficulties [4]. However, these are limited to marker and fiducials. Another approach was developed by Klein et al. where an inertial sensor was used to predict the motion blur for more robust edge extraction [11]. In this paper, we rely on the camera and natural features only.

Recently, Klein et al. developed a SLAM method robust to motion blur [5]. It first estimates the amount of blur and its direction by template matching between small-scaled camera images of the previous and current frames. This then helps extracting edges for model-based tracking [11]. In this paper, we focus on object tracking, which is more challenging since the object can represent only a portion of the captured image. We use a template matching-based approach because it does not rely on feature extraction [12], [13], which can be dangerous when motion blur erases most of the image features, and because it usually yields very accurate results.

Rendering virtual objects with a consistent motion blur was already done [6], [3]. Okumura et al. only consider the blur due to an out-of-focus lens and estimate the Point Spread Function from the image to apply it to the rendered image [3]. Fischer et al. consider 2D translational motion blur and simulate it by translating the object image several times and summing the created images [6]. A more recent approach by Klein et al. consider the motion blur due to the rotation of the camera [7]. In the tracking process, they estimate the image motion as a composition of a 2D translation and a 2D rotation, and use this motion to transfer the motion blur on the virtual objects.

These approaches are limited to uniform 2D motion blur, and cannot render effects such as the ones shown in the first three columns of Figure 1, where the amount of blur is not uniform because of rotation motions. Mei et al. proposed an algorithm that takes perspective into account correctly and showed it can efficiently be implemented on the GPU [14]; unfortunately it can only consider planar objects. By contrast, our method renders complex effects on 3D objects while remaining efficient by rendering the virtual objects twice under different viewpoints and interpolating and merging these two views.

3 The ESM-Blur and ESM-Blur-SE Algorithms

In this section, we first introduce our image formation model that considers blur explicitly, and we then show how the ESM algorithm can be adapted to this model. In the next section we will show how to efficiently render blurred virtual objects under the same image formation model.

3.1 Image Formation Model

The usual approach of template matching-based tracking techniques is to assume a simple image formation model $\mathcal{I}(\Theta)$ where the captured image $\mathcal{I}_c$ can be generated by warping a known template $\mathcal{T}$:

$$\mathcal{I}(\Theta) = w(\mathcal{T}, \Theta)\,, \tag{1}$$

where $w(.,.)$ is a warping function, applied to the template $\mathcal{T}$ with parameters Θ. The goal is to estimate motion $\widehat{\Theta}$ by maximizing the correlation between the captured image $\mathcal{I}_c$ and the warped template as:

$$\widehat{\Theta} = \underset{\Theta}{\operatorname{argmin}} \|\mathcal{I}_c - \mathcal{I}(\Theta)\|^2\,. \tag{2}$$

However such standard model ignores the possibility of motion blur, and we generalize this model as:

$$\mathcal{I}(m, t_0) = \frac{1}{1-t_0}\int_{t_0}^{1} w(\mathcal{T}, m(t))dt\,, \tag{3}$$

that is, the captured image is assumed to be the average of the template warped under a motion defined by $m(t)$. t_0 denotes the time when the shutter opens, and can be normalized to be between 0 and 1. We now have to estimate the full motion of the template between t_0 and 1.

For simplicity, we assume this motion is linear so that we can write:

$$m(t) = t \cdot \Theta\,, \tag{4}$$

where Θ is the template motion on shutter-close, and our image formation model of Eq. (3) is simplified as:

$$\mathcal{I}(\Theta, t_0) = \frac{1}{1-t_0}\int_{t_0}^{1} w(\mathcal{T}, t\Theta)dt\,. \tag{5}$$

This model is valid even if there is no motion blur—in cases when neither the object nor the camera move. Moreover when $t_0 \to 1$, that is when the capture is assumed to be instantaneous, this model tends to the more standard model of Eq. (1). Figure 2 validates the reliability of our image formation model for general motion.

One issue is parameterization, which needs to satisfy the linear interpolation assumed in Eq. (4). We use exponential maps for rotations together with simple 3-vectors for translations. We show in the following that optimizing Eq. (2) can be done for such a model.

3.2 Optimization

Thanks to the Inverse Compositional algorithm [12], we can always assume that the template motion Θ is small. More precisely, the input image is first warped with the inverse of the transformation estimated for the previous image, and we only have to estimate the motion from the template in a reference position to the warped input image. This motion is small if the object moves slowly enough. To make the derivations simpler, we consider here the simple case where $\Theta = \mathbf{0}$ corresponds to the template reference position.

Since the template motion Θ is assumed to be small, we can consider as was done in the formulation of the ESM algorithm, the second-order Taylor expansion of the warping function as:

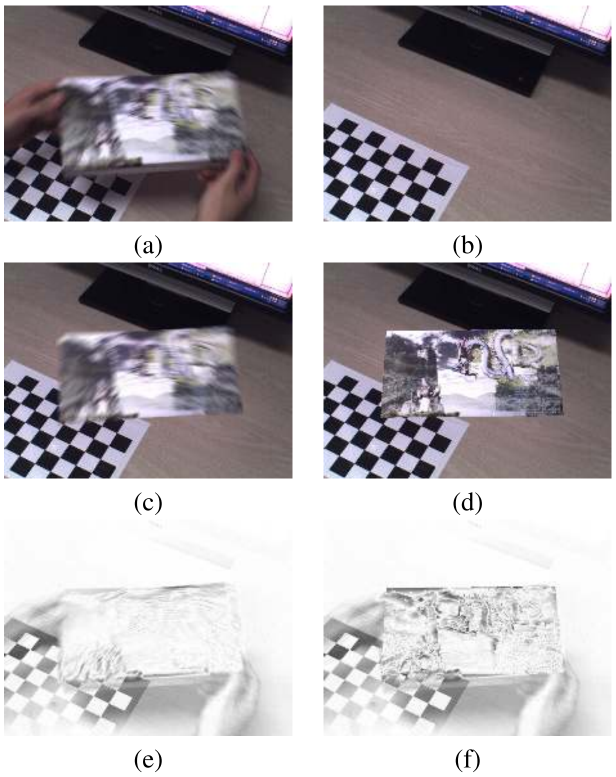


Fig. 2. Validation of the proposed image formation model. (a) is the capture of the box with motion blur. (b) is the background image which is used for overlay of (c) the rendered template with motion blur by Eq. (5) and (d) the rendered template without motion blur. (e) and (f) show the absolute difference of (c) and (d) with the captured image. The proposed image formation model produces much less difference. Note that the proposed tracking does not require template blurring.

$$w(\mathcal{T}, \Theta) \approx \mathcal{T} + \mathbf{J}_{\mathcal{T}}\Theta + \Theta^\top \mathbf{H}_{\mathcal{T}}\Theta\,, \tag{6}$$

where $\mathbf{J}_{\mathcal{T}}$ is the Jacobian of $w(\mathcal{T}, \Theta)$ and $\mathbf{H}_{\mathcal{T}}$ is its Hessian, computed at $\Theta = 0$. By plugging this into Eq. (5), we obtain the following second-order approximation of our image formation model:

$$\begin{aligned}
& \mathcal{I}(\Theta, t_0) \\
\approx\ & \frac{1}{1-t_0}\int_{t_0}^{1} [\mathcal{T} + \mathbf{J}_{\mathcal{T}}(t\Theta) + (t\Theta)^\top \mathbf{H}_{\mathcal{T}}(t\Theta)]dt \\
=\ & \mathcal{T} + \frac{1}{1-t_0}(\int_{t_0}^{1} t dt)\mathbf{J}_{\mathcal{T}}\Theta + \frac{1}{1-t_0}(\int_{t_0}^{1} t^2 dt)\Theta^\top \mathbf{H}_{\mathcal{T}}\Theta \\
=\ & \mathcal{T} + a(t_0)\mathbf{J}_{\mathcal{T}}\Theta + b(t_0)\Theta^\top \mathbf{H}_{\mathcal{T}}\Theta
\end{aligned} \tag{7}$$

We can avoid the term $b(t_0)\Theta^\top \mathbf{H}_{\mathcal{T}}\Theta$ as it was done in the ESM derivations by considering the expression of the Jacobian of $w(\mathcal{T}, .)$ for the captured image. We express the Jacobian $\mathbf{J}_{\mathcal{I}}$ of our image formation function $\mathcal{I}(\Theta)$ at Θ by computing the derivative of Eq. (7). Then, we get:

$$\begin{aligned}\mathbf{J}_{\mathcal{I}}(\Theta) &= \frac{\partial \mathcal{I}(\Theta, t_0)}{\partial \Theta^\top} \\ &= a(t_0)\mathbf{J}_{\mathcal{T}} + 2b(t_0)\Theta^\top \mathbf{H}_{\mathcal{T}}\,. \end{aligned} \tag{8}$$

By plugging Eq. (8) into Eq. (7), our image formation model becomes

$$\begin{aligned}\mathcal{I}(\Theta, t_0) &\approx \mathcal{T} + a(t_0)\mathbf{J}_{\mathcal{T}}\Theta + \frac{1}{2}(\mathbf{J}_{\mathcal{I}}(\Theta) - a(t_0)\mathbf{J}_{\mathcal{T}})\Theta \\ &= \mathcal{T} + \frac{1}{2}(\mathbf{J}_{\mathcal{I}}(\Theta) + a(t_0)\mathbf{J}_{\mathcal{T}})\Theta\,,\end{aligned}$$

which we can use to compute the motion $\widehat{\Theta}$ as:

$$\widehat{\Theta} = \mathbf{J}^{+}_{\text{blur}}(\mathcal{I}_c - \mathcal{T})\,, \tag{9}$$

where $\mathbf{J}^{+}_{\text{blur}}$ is the pseudo-inverse of $\mathbf{J}_{\text{blur}}$, with

$$\mathbf{J}_{\text{blur}} = \frac{1}{2}(\mathbf{J}_{\mathcal{I}}(\Theta) + a(t_0)\mathbf{J}_{\mathcal{T}})\,. \tag{10}$$

$a(t_0)$ is a function dependent on the shutter opening time. From Eq. (7),

$$a(t_0) = \frac{1}{1-t_0}\int_{t_0}^{1} t dt = \frac{t_0+1}{2}\,.$$

$\widehat{\Theta}$ is added to the current pose. This process is performed iteratively until the Euclidean norm of the motion update becomes smaller than a threshold.

Eq. (10) generalizes the standard ESM formula. When we assume as it is usually done that the exposure time is infinitesimal, then $t_0 = t_1$ and $a = 1$, and Eq. (10) simplifies to the standard ESM formula. On the other extreme, when the shutter remains opened, $t_0 = 0$ and then $a = \frac{1}{2}$, reducing the influence of the Jacobian computed at the template.

It is therefore very easy to modify an existing implementation of the original ESM to make it more robust to motion blur. We call the resulting algorithm ESM-Blur.

In practice, the correct value for a can be directly computed if one knows the camera shutter opening and closing times. In our previous work, we assumed t_0 was equal to 0 [10]. In the following section, we present our solution to relax this assumption and estimate the value of t_0 automatically.

3.3 Shutter Speed Estimation

The method described in the previous section assumed t_0 to be known. However, this is not necessarily the case in practice, for example the camera automatically adapts the shutter speed to the current lighting conditions. Even when the lighting condition is mostly static, the shutter speed still changes according to the change of principal brightness of the scene. This can be caused by moving objects and shadow cast from them.

We now show that we can estimate both the motion parameters and shutter opening time simultaneously. For this, the parameter vector we want to estimate is now augmented with the increment of t_0 as $[\Theta \mid \Delta t_0]^\top$. This requires derivatives of the input image and template with respect to t_0, which can be computed analytically:

$$\begin{aligned}\mathbf{J}_{\mathcal{I}}(\Theta, t_0) &= \left[\mathbf{J}_{\mathcal{I}}(\Theta) \mid \frac{\partial \mathcal{I}(\Theta, t_0)}{\partial t_0}\right] \\ &= \left[\mathbf{J}_{\mathcal{I}}(\Theta) \mid \frac{1}{2}\dot{a}(t_0)\mathbf{J}_{\mathcal{T}}\Theta\right]\,, \text{ and} \qquad (11) \\ \mathbf{J}_{\mathcal{T}}(\Theta, t_0) &= [\mathbf{J}_{\mathcal{T}}(\Theta) \mid \mathbf{0}]\,, \qquad (12)\end{aligned}$$

where the augmented column of $\mathbf{J}_{\mathcal{T}}(\Theta, t_0)$ is simply the null vector because we can assume the template is captured without any motion.

By substituting the Jacobian matrices in Eq. (10) with the augmented matrices in Eqs. (11) and (12), we get the final augmented Jacobian matrix of $\mathbf{J}_{\text{blur}}$ as:

$$\begin{aligned}\mathbf{J}_{\text{blurSE}} &= \frac{1}{2}(\mathbf{J}_{\mathcal{I}}(\Theta, t_0) + a(t_0)\mathbf{J}_{\mathcal{T}}(\Theta, t_0)) \\ &= \frac{1}{2}\left[\mathbf{J}_{\mathcal{I}}(\Theta) + a(t_0)\mathbf{J}_{\mathcal{T}} \mid \frac{1}{2}\dot{a}(t_0)\mathbf{J}_{\mathcal{T}}\Theta\right]\,, \end{aligned} \tag{13}$$

which can be used to estimate the motion and t_0 parameters iteratively at the same time in the same way as Eq. (9) by replacing $\mathbf{J}_{\text{blur}}$ with Eq. (13) as:

$$\begin{bmatrix} \widehat{\Theta} \\ \Delta t_0 \end{bmatrix} = \mathbf{J}^{+}_{\text{blurSE}}(\mathcal{I}_c - \mathcal{T})\,. \tag{14}$$

At each iteration, the shutter opening time parameter is updated by addition. However, we force t_0 to remain positive, and lower to 1 as:

$$t_0 = \begin{cases} 0 & t_0 < -\Delta t_0 \\ 1 & t_0 > 1 - \Delta t_0 \\ t_0 + \Delta t_0 & \textit{otherwise} \end{cases} \tag{15}$$

because it is assumed to be normalized between 0 and 1 in our image formation model. Without this, t_0 could become out of its proper range because of numerical errors and model imperfection. We call the resulting algorithm ESM-Blur-SE, for Shutter speed Evaluation.

3.4 Initialization and Re-Initialization

We use the Ferns method [2] to automatically initialize our tracking method described above. It also happens that tracking fails when the blur is too important or when there is a complete occlusion, and we use the Ferns for re-initialization when a failure is detected. To detect a failure, we compute the Normalized Cross-Correlation (NCC) between the template and the image patch warped back from the input image with the estimated pose. In practice, we estimate the tracking has failed if the NCC is less than 0.8.

4 3D Motion Blur Generation

This section describes our method to render the virtual objects under a motion blur consistent with the blur present in the captured image. It is based on the image formation model already introduced in Eq. (3) for tracking. To stick with real-time constraints, we propose a method that combines 3D rendering and 2D warping efficiently.

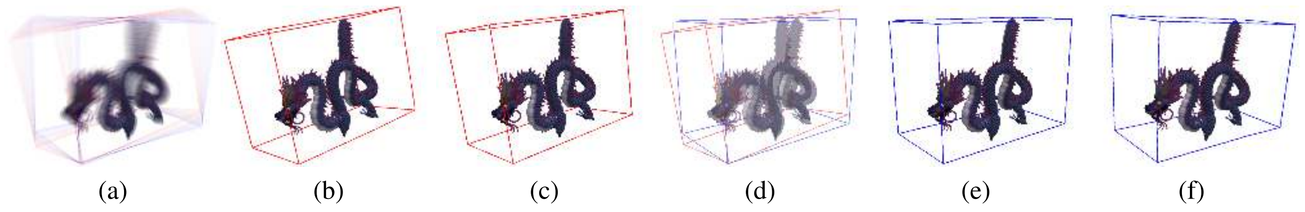


Fig. 3. Overview of our approach to efficient blur rendering. To render the virtual object consistently with the retrieved motion such as in (a), we first generate two intermediate views (c) and (e) we call *intraframes* using OpenGL. To create the other intermediate views, we warp the intraframes using affine transformations to approximate perspective projections. The contributions of the two intraframes are weighted and summed to obtain intermediate views such as (d). (b) and (f) are warped respectively from (c) and (e) only. Finally, by summing all the intermediate views, we obtain a blurred view as in (a).

4.1 Rendering Blur

One general way to simulate motion blur is to render the virtual object many times along its path during the image exposure and blend the rendered images all together. However, several tens of such images are needed in practice to create realistic smooth blur, and doing the rendering in 3D would be prohibitive for real-time applications. Rendering a model many times for blur effect is not attractive as the system would have to dedicate more computation times to rendering. Furthermore, Computer Vision algorithms now tend to use the GPU, leaving less time for rendering.

Therefore, as depicted by Figure 3, our strategy is to render only a few images of the virtual object, and generate intermediate images by warping these rendered images. We call such sparsely rendered images *intraframes* by quoting the term from the MPEG format specifications. Because warping is much faster than 3D rendering, this allows us to generate correctly blurred images within our real-time constraints.

More formally, the image $\mathcal{I}_{\text{virt}}$ we want to create can be written as:

$$\mathcal{I}_{\text{virt}} = \frac{1}{1-t_0}\int_{t_0}^{1} r(\mathcal{O}, m(t))dt\,, \tag{16}$$

where $r(\mathcal{O}, m(t))$ is the image of the virtual object $\mathcal{O}$ rendered under pose $m(t)$. As in Section 3, t_0 denotes the time when the camera shutter opens, which is normalized to be between 0 and 1. Also as in Section 3, we assume that $m(t)$ evolves linearly between t_0 and 1. Our image formation model for rendering is therefore similar to the one we used for tracking as defined in Eq. (3). But contrary to Section 3, we cannot assume here the object pose is always small, and we write $m(t)$ as $m(t) = \Theta_p + t\Theta$, where Θ_p is the object pose estimated for the previous captured image, and Θ is the object motion for the current image computed as in the previous section.

In practice, we have to replace the integral in Eq. (16) by a finite sum:

$$\mathcal{I}_{\text{virt}} \approx \frac{1}{n+1}\sum_{i=0}^{n} r(\mathcal{O}, \Theta_p + \frac{i}{n}\Theta)\,. \tag{17}$$

The blur realism increases with n, but of course the rendering time increases as well.

As discussed above, we actually render in 3D only two intraframes. For a good balance, we choose to take them at $i = \frac{n}{4}$ and $i = \frac{3n}{4}$. We therefore approximate $\mathcal{I}_{\text{virt}}$ as

$$\mathcal{I}_{\text{virt}} \approx \frac{1}{n+1}\big(\mathbf{I}_{\frac{n}{4}} + \mathbf{I}_{\frac{3n}{4}} + \sum_{\substack{i=0 \\ i\neq\frac{n}{4},\, i\neq\frac{3n}{4}}}^{n} \mathbf{B}_i\big)\,, \tag{18}$$

where $\mathbf{I}_{\frac{n}{4}} = r(\mathcal{O}, \Theta_p + \frac{\Theta}{4})$ and $\mathbf{I}_{\frac{3n}{4}} = r(\mathcal{O}, \Theta_p + \frac{3\Theta}{4})$. The images $\mathbf{B}_i$ for the other values of i are intermediate images created from $\mathbf{I}_{\frac{n}{4}}$ and $\mathbf{I}_{\frac{3n}{4}}$ as explained in the next section.

4.2 Generating the Intermediate Images

From the two intraframes, we generate n intermediate frames using affine warping. One strategy could be to use only the first intraframe to generate the first part of the intermediate frames, and the second intraframe to generate the second part. However that would result in a discontinuity at the junction of the two parts since the affine warping is only an approximation of a real 3D rendering. Near this junction, we therefore blend the warped images from the two intraframes together to avoid this discontinuity.

More exactly an intermediate frame $\mathbf{B}_i$ is computed as

$$\mathbf{B}_i = \begin{cases} w(\mathbf{I}_{\frac{n}{4}}, \mathbf{A}_{\frac{n}{4}\to i}) & \text{if } i < \frac{n}{4} \\ \alpha(i)w(\mathbf{I}_{\frac{n}{4}}, \mathbf{A}_{\frac{n}{4}\to i}) + \beta(i)w(\mathbf{I}_{\frac{3n}{4}}, \mathbf{A}_{\frac{3n}{4}\to i}) & \text{if } \frac{n}{4} < i < \frac{3n}{4} \\ w(\mathbf{I}_{\frac{3n}{4}}, \mathbf{A}_{\frac{3n}{4}\to i}) & \text{if } i > \frac{3n}{4} \end{cases} \tag{19}$$

where $\alpha(i)$ varies linearly from 1 to 0 when i varies from $\frac{n}{4}$ to $\frac{3n}{4}$, and $\beta(i)$ varies linearly from 0 to 1 when i varies from $\frac{n}{4}$ to $\frac{3n}{4}$. $w(.,.)$ is the warping function and the $\mathbf{A}_{i\to j}$ are the affine transformations between image i and image j.

To compute the affine transformations $\mathbf{A}_{i\to j}$, we rely on the projections of the corners of the bounding box to the virtual object. We project these corners in the intraframes and the intermediate frames using a pose interpolated in 3D from t_0 to 1. The transformation $\mathbf{A}_{i\to j}$ is then computed as the affine transformation that transforms the corners projections in frame i to their projections in frame j as best as possible, in the least-squares sense. Thus the transformation is approximately valid for any pixel lying on the virtual object.

The approximations involved by the affine warping are validated in Figure 4 in which we compare renderings with and without these approximations, showing the approximations yield almost no visual artefacts.

4.3 Determining the Number of Intermediate Images

This section presents an adaptive scheme for determining the number of intermediate frames used for rendering. In our previous paper [10], this number was systematically set to a fixed value (we used 48 intermediate images for a total of 50 images when including the intraframes). It was a simple but not optimal solution as a small motion could require much less images for a satisfying rendering, and a large motion more images.

Therefore we estimate the required number of intermediate images according to the motion amplitude in the image. This is done by first projecting the corners of the bounding box using the estimated pose and computing their distances to their locations in the previous image. We then use the average distance in pixels as the number of intermediate frames. In practice, we also limit the maximum number to 48 in our implementation since we observed it is visually acceptable for inter-frame motion.

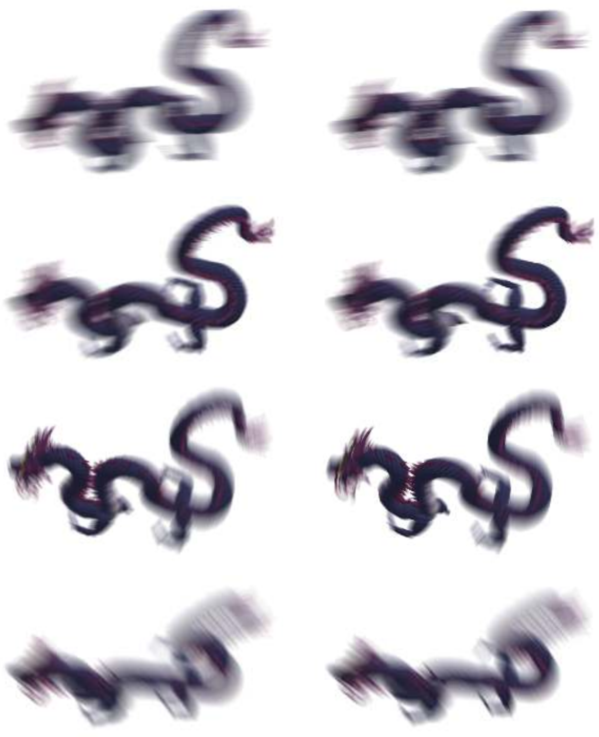


Fig. 4. Validating our blur rendering method. In order to validate the approximations done in our blur rendering method, we compared for four different motions renderings using our method with 48 intermediate images (on the left column) against renderings obtained by using 1000 intermediate images rendered without approximations with OpenGL (on the right column). The differences are barely visible while our method is significantly faster.

```
// Intermediate image:
uniform sampler2D tex_src;

// Accumulated and blended image:
uniform sampler2D tex_acc;

varying float va; // alpha for an image
varying vec2  uv; // texture coordinate

void main()
{
  vec4 src =
    texture2D(tex_src, gl_TexCoord[0].st);

  if (color.a == 0.0)
  {
    discard;
  }
  else
  {
    vec4 dst = texture2D(tex_acc, uv.st);
    gl_FragColor  = src * va + dst;
  }
}
```

(a)

```
// Accumulated and blended image:
uniform sampler2D tex_acc;

// Camera image:
uniform sampler2D tex_dsp;

// # of intermediate images:
uniform int   N;

varying vec2  uv; // texture coordinate

void main()
{
  vec4 src =
    texture2D(tex_acc, gl_TexCoord[0].st);
  vec4 dst = texture2D(tex_dsp, uv.st);

  if (src.a > 0.0)
  {
    if (color.a > 1.0-1.0/N) color.a = 1.0;
    gl_FragColor = src + dst * (1.0-src.a);
  }
  else
  {
    gl_FragColor = dst;
  }
}
```

(b)

Fig. 5. GLSL source codes for (a) accumulating intermediate images and (b) final overlaying on the camera image.

4.4 Implementation

For efficiency, we warp the intraframes and blend the resulting intermediate images using the graphic card only. Not only the graphic card can perform these two operations very fast, but this avoids reading back the intraframes from the video memory into the main memory. The rendering of the intraframes is done using render-to-texture in OpenGL. The textures are set to fit on a virtual image plane, which is then transformed by linear interpolation to create the intermediate images. The final motion-blurred image is generated by simply accumulating each image with proper alpha values. The whole process is very fast and enough images can be generated to create a realistic blur effect while keeping the real-time constraint.

The textures used to store the intraframes and the accumulation image require 4 channels, for the three color channels plus one for alpha blending. Specifically, the accumulation image uses the 32-bit floating type pixel format to avoid discretization issues which can become important when we consider alpha blending of many images. However, if the processing time must be more shorter while giving up fine details, 8-bit format textures can be still used with some more care.

The warping and accumulation process is done as follows. The intraframes are rendered over a transparent, black background (R=G=B=A=0.0). The alpha channels of the pixels that lie on the object are then set to 1. Warping this intraframe to the intermediate images is done by rendering a rectangle under the pose estimated for the intermediate image with the intraframe as the texture. This rendering is directly accumulated to the texture by the fragment shader provided in Figure 5(a).

Note that this accumulated image is not yet overlaid on the camera image. The camera image is filled in another texture with alpha values of 0. If we can neglect discretization issues, this blending can be implemented with naive OpenGL operations. However it is not always the case if the number of intermediate images is too large. The second fragment shader presented in Figure 5(b) completes the alpha values of the accumulation image before blending with the camera image.

5 Experimental Results

We evaluated our tracking and motion blur generation algorithms on a desktop PC with a 3.2 GHz CPU and an NVIDIA GeForce GTX 580 graphics card. The camera used for the tracking captured 640×480 resolution images up to 30Hz. Its shutter speed can be fixed with a specified exposure time or adjusted automatically. Rendering is done at the same resolution with the camera image.

5.1 Tracking

5.1.1 Synthetic Images

We first performed synthetic experiments to measure and compare the performance of ESM, ESM-Blur, and ESM-Blur-SE (ESM-Blur with shutter speed estimation). We selected the three templates shown in Figure 6 from the benchmark dataset [9]. We generated synthetic images with controlled motions and exposure times. Some of them are shown in Figure 7. The performance evaluation focuses only on the tracking performance since the shutter speed estimation is intended to help the pose estimation.

We experimented with three different types of motion: 3D translation only, 3D rotation only, and a combination of 3D translation and rotation. For each type, we increased the motion gradually. Table 1 gives the correspondence between the 3D motion and the average 2D motion of the template corners in the image.

We measured at each step the success ratio, the number of iterations to convergence, Normalized Cross-Correlation (NCC) between the template and the image patch at the converged pose. If the NCC is higher than some threshold, it was counted as a success. For each step of motion, 1000 images were generated with random exposure time. Note that the images do not constitute a consecutive sequence. For each image, we ran the three algorithms with the same initial pose used for blur generation. The maximum number of iterations were limited to 100 to prevent too much computation time.

TABLE 1
Average amplitude (in pixels) of the template corners 2D motion as a function of the displacement in 3D space in the synthetic images. The 2D motion grows almost linearly with the 3D motion.

Translation (cm)	**5**	**10**	**15**	**20**	**25**	**30**
2D motion (pixels)	12.1	24.0	35.9	48.0	60.0	72.0
Rotation (deg)	**5**	**10**	**15**	**20**	**25**	**30**
Pixel	12.6	25.0	37.5	49.9	62.3	74.8
Transl. & Rot.	**5**	**10**	**15**	**20**	**25**	**30**
Pixel	16.2	31.8	47.7	63.8	80.0	95.7

The results for the template in Figure 6(a) are shown in Figure 8. We observed very similar trends from the results for other templates, therefore we omit them to save space. The success rates are higher for ESM-Blur and ESM-Blur-SE than the standard ESM algorithm. The proposed methods also converge faster than ESM. Except maybe for the rotational motion for which the difference is small, the improvements are consistent over the different experiments. Note that even though ESM-Blur-SE converges faster than ESM-Blur, it requires more computation per iteration. The average processing time is presented in the following section.

Figure 8(j)-(l) show the difference of the estimated pose by the proposed methods and the ground truth pose. We performed more experiments on other targets, and they are consistent with the results shown in Figures 8(g)-(i).

The gain of the tracking performance does not sacrifice robustness to image noise. In the generated images, we added Gaussian noise of mean 0 and increasing standard deviation to see how the performance varies. The noise is added independently to each pixel of the 3 channels. The results are shown in Figure 9. This validates that the performance degradation rate of the proposed methods is similar, thus the proposed methods have no specific drawback to the image noise.

5.1.2 Real Sequences

Our two test sequences are 930 and 1500 frames long, and are available as supplementary material. The second sequence contains much faster motion, thus tracking fails more frequently. During capture of both sequences, the camera was set on auto-shutter control. Some frames are presented in Figure 1. For ESM-Blur, we set $t_0 = 0$ and $t_1 = 33.33$ms. Table 2 compares our ESM-Blur and ESM-Blur-SE with the standard ESM.

Over the first sequence, ESM fails to converge correctly 11 times, while ESM-Blur and ESM-Blur-SE fail only 3 and 2 times, respectively. Both methods are reinitialized

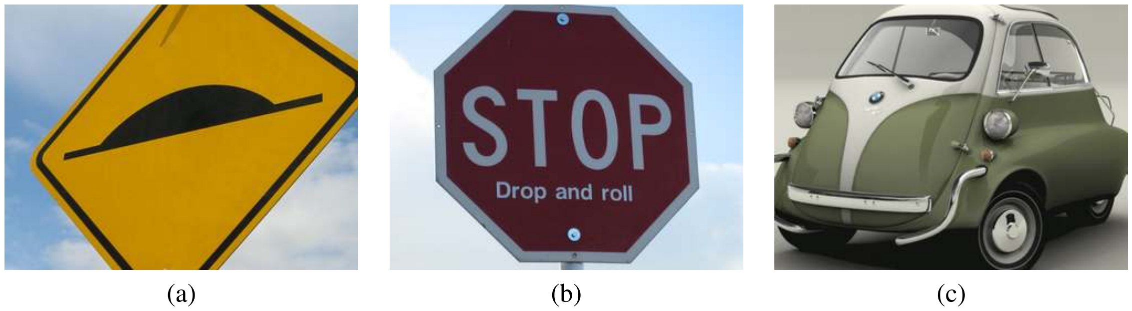


Fig. 6. The three templates from the dataset introduced by Lieberknecht et al. [9] are used in our synthetic experiments.

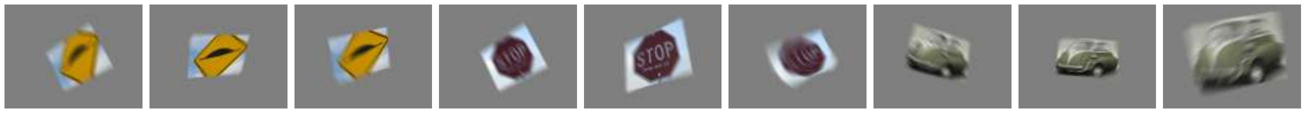


Fig. 7. Examples of blurred images generated from the templates in Figure 6.

TABLE 2
Comparison of tracking performances for ESM, ESM-Blur, and ESM-Blur-SE on two real image sequences composed of 930 and 1500 frames each. The percentages in parentheses report the improvements with respect to the original ESM algorithms.

Sequence #1	**ESM**	**ESM-Blur**	**ESM-Blur-SE**
Number of Correctly Tracked Frames	767	859 (+11.99%)	910 (+18.64%)
Average Number of Iterations	24.33	20.04 (-17.63%)	16.81 (-30.9%)
Sequence #2	**ESM**	**ESM-Blur**	**ESM-Blur-SE**
Number of Correctly Tracked Frames	1004	1209 (+20.41%)	1282 (+27.68%)
Average Number of Iterations	27.62	23.48 (-14.98%)	19.01 (-31.17%)

TABLE 3
Average times for tracking and rendering evaluated on the sequence of Figure 1. Rendering was performed using 48 intermediate images, and the naive solution with 50 views takes about 12.30 ms.

	Step		**Average time (ms)**
Tracking	Conversion to grayscale		0.89
	Motion estimation	ESM	0.17 / iter (4.69 / frame)
		ESM-Blur	0.17 / iter (3.77 / frame)
		ESM-Blur-SE	0.24 / iter (4.36 / frame)
Rendering	Intraframe rendering		0.65
	Inter. image & alpha blending		0.60
	Final rendering		0.72

automatically in case of failure using a Fern-based detector when the amount of blur becomes small enough, which means that the following frames are also lost until the reinitialization succeeds. Avoiding failures is of course still desirable to avoid augmentation interruption. The proposed methods respectively tracked 92 and 143 more frames, which are 11.99% and 18.64% more than ESM.

The iterations were stopped when the Euclidean norm of the motion update was less than 0.1. The frames with more motion blur typically need more iterations and yield lower NCC. The average iterations of ESM-Blur of ESM-Blur-SE are respectively 17.63% and 30.9% lower than that of ESM, and it shows faster convergence.

The second sequence exhibits more challenging conditions, therefore the tracking fails more often and takes more iterations. ESM fails to converge correctly 51 times, while the proposed methods fail only 28 and 23 times. The proposed methods tracked 205 and 278 more frames, which are 20.41% and 27.68% more than ESM. Iterations were 14.98% and 31.17% lower than ESM.

Figure 10 shows the convergence of the three methods on some images of the second sequence. In these images ESM failed to converge correctly, while ESM-Blur and ESM-Blur-SE estimated the motion correctly. It can also be seen that ESM-Blur-SE converged within fewer iterations. Figure 11 gives quantitative results, and is consistent with the performances evaluated on the synthetic images. The proposed methods outperform ESM both in the number of iterations and the final NCC.

Average computation times for the most consuming steps are given in Table 3. Because the computations involved in ESM and ESM-Blur are very similar but ESM-Blur requires less iterations, ESM-Blur is faster than ESM. Although ESM-Blur-SE converges within even fewer iterations, the overall time is slightly more because one iteration takes more computation due to the additional parameter estimation. From this result, we recommend ESM-Blur-SE for more robustness, and ESM-Blur for faster performance, for example on mobile devices.

In our experiments, we took 32×32 pixels for the template size because it was a good compromise between the performance and computation time. Usually, the op-

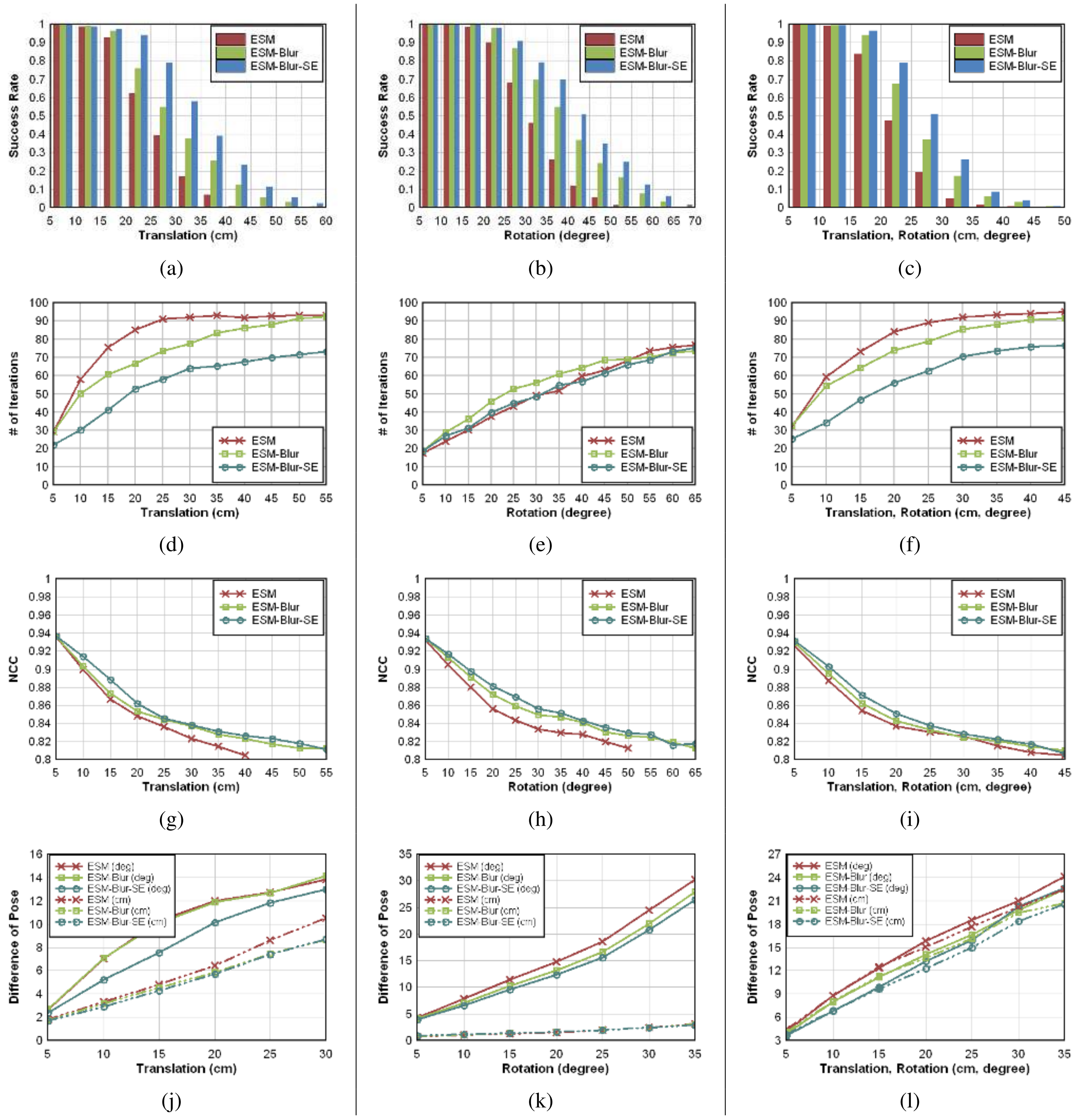


Fig. 8. Comparison of tracking performances of ESM, ESM-Blur, and ESM-Blur-SE on synthetically generated images for the template of Figure 6(a). From left to right: translation only, rotation only, and translation and rotation. From top to bottom: Correct convergence rate, number of iterations, Normalized Cross-Correlation at convergence, and differences between the estimated pose and the ground truth pose. ESM-Blur-SE systematically outperforms ESM-Blur, which systematically outperforms ESM.

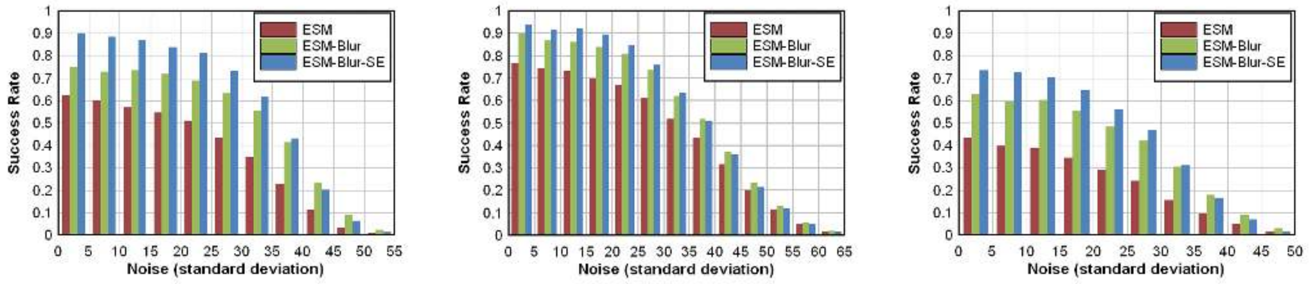
timization converges within fewer iterations if the template size is larger, however each iteration spends more time. The computation time for an iteration of our unoptimized implementation takes time proportional to the template size.

5.2 Motion Blur Generation

The dragon model used in the experiment is made of 16,583 vertices and 31,144 polygons. The model was rendered as a compiled display list, which is very fast since all vertices and pixel data are stored in the graphics card.

Typical results of the proposed motion-blur generation are shown in Figure 4. Even though we incorporated 2D image warping instead of several 3D renderings, the rendered image illustrates little artefacts, which is almost invisible in real-time augmentation.

Average rendering times are given in Table 3. The most



(a) (b) (c)

Fig. 9. Success rates for increasing level of image noise for (a) translation-only motion, (b) rotation-only motion, and (c) translation and rotation. The average motion was 20 cm (with standard deviation 10 cm) and 20 degrees (with standard deviation 10 degree) rotation along every axis. The proposed methods exhibit similar performance degradation to the original ESM.

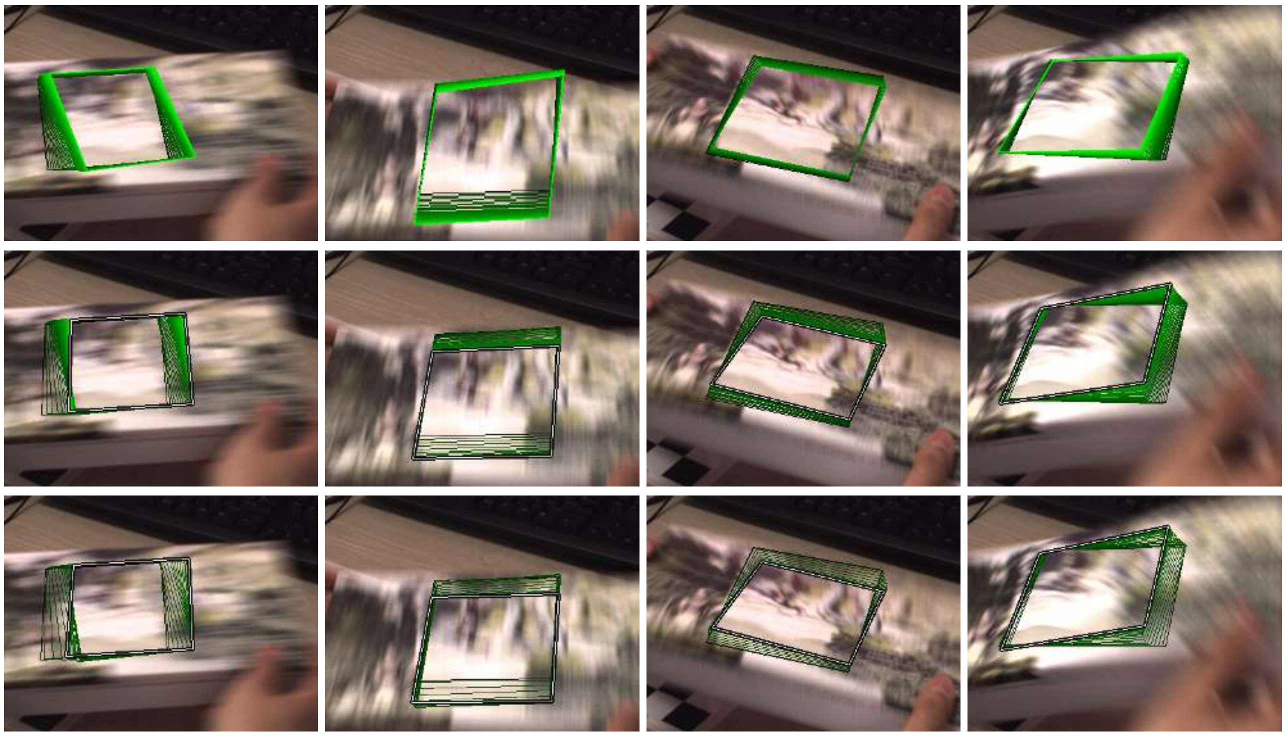


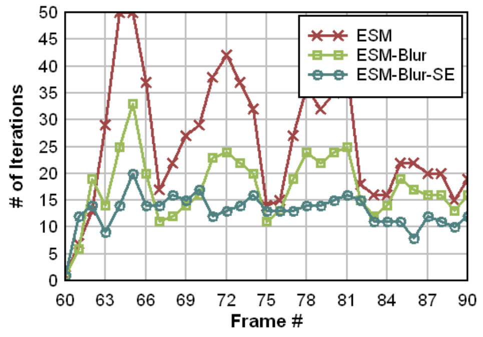
Fig. 10. Comparison of the convergence between ESM (top row), and the proposed ESM-Blur (middle row) and ESM-Blur-SE (bottom row) when ESM fails in the sequence shown in Figure 1. The quadrangles denote the intermediate positions for the template as retrieved by the successive iterations. ESM-Blur-SE converges to the correct pose even faster than ESM-Blur. The images are cropped around the center for visibility.

object dependent part is the intraframe rendering, which needs to draw the object twice. By contrast, rendering intermediate frames is very cheap and independent from the complexity of the object. Both operations are performed on the graphics card, letting the CPU acquiring and tracking the next frame while the GPU is rendering the augmented image.

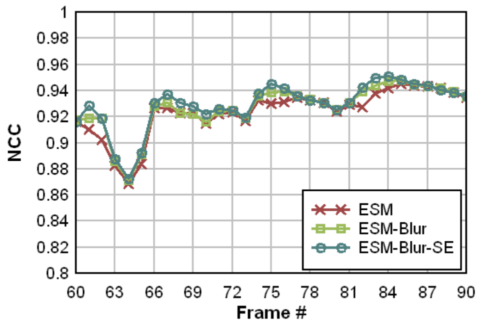
6 Conclusion & Future work

We proposed two complementary methods for Augmented Reality under motion-blur, one for tracking, and the other one for rendering. The core is a generalized image formation model which simulates more realistic image capture. The model is then used in a very efficient optimization based on ESM, and also applied to motion-blur generation with a simple and efficient implementation. We also presented how to eliminate the need of manual exposure time setting, and proved the improvements.

Our implementations of the tracking methods exploited only the original input resolution images for performance measure. To enlarge further the convergence range, pyramidal implementations could also be used in conjunction



(a)



(b)

Fig. 11. Comparison of tracking performances over a section of the real sequence for which all methods always converge. (a) The number of iterations to convergence. (b) The NCC between the warped template and the current image. The performance are consistent with the synthetic experiments. The proposed methods converge faster and NCC is slightly better.

with our algorithms.

An interesting future research direction is to investigate other image formation models to better handle complex imaging effects, for example image distortions due to rolling shutters often used in webcams and cameras in mobile phones. Extensions that estimate another useful parameters such as intrinsic camera parameters would be also useful.

Acknowledgments

This work was supported 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)

References

[1] D. G. Lowe, “Distinctive Image Features from Scale-Invariant Keypoints,” *International Journal of Computer Vision (IJCV)*, vol. 60, no. 2, pp. 91–110, Nov. 2004.
[2] M. Ozuysal, P. Fua, and V. Lepetit, “Fast Keypoint Recognition in Ten Lines of Code,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.
[3] B. Okumura, M. Kanbara, and N. Yokoya, “Augmented Reality Based on Estimation of Defocusing and Motion Blurring from Captured Images,” in *International Symposium on Mixed and Augmented Reality (ISMAR)*, 2006.
[4] D. Claus and A. Fitzgibbon, “Reliable Fiducial Detection in Natural Scenes,” in *European Conference on Computer Vision (ECCV)*, 2004.
[5] G. Klein and D. Murray, “Improving the Agility of Keyframe-Based SLAM,” in *European Conference on Computer Vision (ECCV)*, 2008.
[6] J. Fischer, D. Bartz, and W. Strasser, “Enhanced Visual Realism by Incorporating Camera Image Effects,” in *International Symposium on Mixed and Augmented Reality (ISMAR)*, 2006.
[7] G. Klein and D. Murray, “Simulating Low-Cost Cameras for Augmented Reality Compositing,” *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, vol. 16, no. 3, pp. 369–380, 2010.
[8] E. Malis, “Improving Vision-based Control using Efficient Second-order Minimization Techniques,” in *Proceedings of the International Conference in Robotic and Automation*, 2004.
[9] S. Lieberknecht, S. Benhimane, P. Meier, and N. Navab, “A Dataset and Evaluation Methodology for Template-based Tracking Algorithms,” in *International Symposium on Mixed and Augmented Reality (ISMAR)*, 2009.
[10] Y. Park, V. Lepetit, and W. Woo, “ESM-Blur: Handling & Rendering Blur in 3D Tracking and Augmentation,” in *International Symposium on Mixed and Augmented Reality (ISMAR)*, 2009.
[11] G. Klein and T. Drummond, “Tightly integrated sensor fusion for robust visual tracking,” *Image and Vision Computing*, vol. 22, no. 10, pp. 769–776, September 2004.
[12] S. Baker and I. Matthews, “Lucas-Kanade 20 Years On: A Unifying Framework,” *International Journal of Computer Vision (IJCV)*, vol. 56, no. 3, pp. 221–255, Mar. 2004.
[13] S. Benhimane and E. Malis, “Homography-based 2D Visual Tracking and Servoing,” *IJRR (Special Issue on Vision and Robotics joint with the IJCV)*, 2007.
[14] C. Mei and I. Reid, “Modeling and generating complex motion blur for real-time tracking,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

Youngmin Park received the BS degree in computer engineering from the Kangwon National University (KNU), Gangwon-do, Korea, in 2004, and the MS degree from the Department of Information and Communications (DIC), Gwangju Institute of Science and Technology (GIST), Gwangju, Korea, in 2006, where he is currently working toward the PhD degree. His research interests include Augmented Reality, 3D vision-based tracking, object detection, and mobile computing.

Vincent Lepetit is a Senior Researcher at the CVLab, EPFL. He received the engineering and master degrees in Computer Science from the ESIAL in 1996. He received the PhD degree in Computer Vision in 2001 from the University of Nancy, France, after working in the ISA INRIA team. He then joined the Virtual Reality Lab at EPFL (Swiss Federal Institute of Technology) as a post-doctoral fellow and became a founding member of the Computer Vision Laboratory. His research interests include vision-based Augmented Reality, 3D camera tracking, object recognition and 3D reconstruction.

Woontack Woo Since 2001, he has been with the Gwangju Institute of Science and Technology (GIST), where he is a Professor in the School of Information and Communications (SIC) and the Director of Culture Technology Institute (CTI). His research interests include 3D Computer Vision for ubiComp, HCI, DigiLog X, Context-Aware Mixed and Augmented Reality (CAMAR), Culture Technology (CT), etc.