

# Enhanced Visual Realism by Incorporating Camera Image Effects

Jan Fischer\*

Dirk Bartz

Wolfgang Straßer

WSI/GRIS - VCM, University of Tübingen

## ABSTRACT

In video see-through augmented reality (AR), virtual objects are overlaid over digital video images. One particular problem of this image mixing process is that the visual appearance of the computer graphics differs strongly from the real background image. The reason for this is that typical AR systems use fast but simple real-time rendering techniques for displaying virtual objects. In this paper, methods for reducing the impact of three effects which make virtual and real objects easily distinguishable are presented. The first effect is camera image noise, which is contained in the data delivered by the image sensor used for capturing the real scene. The second effect considered is edge aliasing, which makes distinguishing virtual objects from real objects simple. Finally, we consider motion blur, which is caused by the temporal integration of color intensities in the image sensor during fast movements of the camera or observed objects.

In this paper, we present a system for generating a realistic simulation of image noise based on a new camera calibration step. Additionally, a rendering algorithm is introduced, which performs a smooth blending between the camera image and virtual objects at their boundary in order to reduce aliasing. Lastly, a rendering method is presented, which produces motion blur according to the current camera movement. The implementation of the new rendering techniques utilizes the programmability of modern graphics processing units (GPUs) and delivers real-time frame rates.

**CR Categories:** H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems—Artificial, augmented, and virtual realities; I.3.3 [Computer Graphics]: Picture/Image Generation—Display algorithms

**Keywords:** augmented reality, photometric registration, photographic imperfections, image noise, antialiasing, motion blur

## 1 INTRODUCTION

In augmented reality (AR), graphical objects are overlaid on top of the view of the real world [2]. Video see-through augmented reality achieves this by continually acquiring digital images from a camera which captures the real environment. The virtual objects are then superimposed on top of these camera images. In order to create a spatially correct overlay of the virtual scene elements, camera tracking techniques are employed (e.g., using the ARToolKit [9]). In most AR systems, standard real-time rendering methods are used for displaying virtual objects. Widespread graphics libraries like OpenGL are often utilized. These standard renderers produce the characteristic artifacts of real-time computer graphics. The generated representation of virtual objects is based on manually defined artificial light sources and simple local lighting models. Moreover, artifacts like aliasing at the outer boundary of virtual objects are generated. Altogether, this leads to significantly different levels of realism of the camera image and the graphical objects. This “realism gap” can be considered one of the main challenges in rendering for AR.

---

\*e-mail: fischer@gris.uni-tuebingen.de

One effect contained in the camera image, which is not present in the virtual objects, is image noise. It is generated by the image sensor of the camera capturing the real environment. Depending on the quality of the image sensor and environmental conditions, the impact of the noise can vary from being a subtle phenomenon to a clearly discernible variation in the video stream. In this paper, a novel approach to rendering virtual objects, which mimics the noise contained in the camera video stream, is proposed. A simple calibration step, which measures image variations relative to averaged color intensities in a static scene, is described. The characteristics of the camera image noise are then estimated. These characteristics are used in a modified AR rendering pipeline, which simulates noise when displaying virtual objects.

A second effect which makes virtual objects easily distinguishable is aliasing. At the outer boundary of graphical models, the typical hard edges generated by a polygonal renderer are visible. Here, we introduce an algorithm which performs a smooth blending between the boundary pixels of virtual objects and the adjacent camera pixels, leading to significantly reduced aliasing.

Another peculiarity in the camera images is motion blur. It results from the temporal integration of color intensities in the image sensor. If there is fast motion in the captured scene, due to fast movement of the camera or observed objects, color intensities are averaged over time. This leads to a blurred reproduction of the real scene. A method for recreating motion blur in the virtual objects is presented. The size and direction of the blur is approximated based on the change of the estimated camera pose. An iterative rendering algorithm then draws overlapping transparent copies of the virtual object, resulting in a blur effect.

The new AR rendering methods described here utilize the programmability of modern graphics processing units (GPUs) and generate output images at real-time frame rates.

## 2 RELATED WORK

The methods proposed in this paper aim at adapting the visual realism of virtual objects to the appearance of the camera image. They can therefore be considered techniques for *photometric registration*, which is defined as the task of adapting the illumination conditions and overall visual appearance of two images. An early method for the correct automatic illumination of virtual objects added to real images was described by Debevec [3]. An example of a method which analyzes the illumination conditions in AR is the work of Kanbara and Yokoya [8]. Their system measures the distribution of real light sources, which is then used for adapting the representation of graphical objects. A similar technique, which also utilizes an acquired environment illumination map, was proposed by Agusanto et al. [1]. In their system, a mirror ball and special camera are used in a preparatory procedure for determining the lighting conditions.

Recently, a different approach to equalizing the visual realism of real and virtual scene elements was proposed. By applying similar non-photorealistic stylization filters to both the camera image and the computer-generated graphics, *stylized augmented reality* generates a homogeneous visual appearance in the output video stream [4]. It was shown that the application of a stylization filter in augmented reality significantly reduced the discernability of virtual objects in a psychophysical study [6]. The drawback of stylized augmented reality techniques is that they alter the camera image, typically by removing details from the input image. While this

is acceptable for some applications, e.g. entertainment and art installations, it is not appropriate in other scenarios. The techniques presented in this paper adapt the virtual objects to the visual appearance of the camera image, which is preserved in its original, unprocessed form.

### 3 IMAGE NOISE

We assume that a given digital camera delivers a distinctive type of image noise. In the literature, precise theoretical descriptions of image sensor noise have been discussed (e.g., see [10]). However, such an exact noise model can only be established using elaborate measurement procedures under controlled conditions. We propose a simplified noise model, which can be estimated based on an un-complicated calibration procedure.

In this simplified noise model, the variation of intensity in each color channel is assumed to be normally distributed. (This assumption is supported by the experimental results mentioned in Sec. 6.) We denote the difference between a captured color channel value and the ideal, correct value as channel variation  $v_{\{r,g,b\}}$ . The probability density function of  $v_{\{r,g,b\}}$  can thus be expressed as a Gaussian function. The characteristics of this normal distribution are described by the mean,  $\mu_i$ , and the standard deviation,  $\sigma_i$ . Each color channel is treated separately, which leads to three different mean values and standard deviations. As a refinement of the basic noise model, we use different normal distributions depending on the overall pixel intensity, since some effects contributing to image noise are proportional to the amount of incoming light (e.g., shot noise [10]). This is modeled by estimating several sets of normal distribution parameters. The average color channel intensity of each pixel is quantized into one of  $N$  intensity bins, for each of which a separate normal distribution is estimated.

An easy-to-use calibration step for determining the noise distribution parameters ( $\mu_i^j, \sigma_i^j$ ) for all channels  $i \in \{r, g, b\}$  and all intensity bins  $j \in \{0, \dots, N-1\}$  is proposed. This calibration step is based on the principle of capturing arbitrary but static scenes with the camera over a duration of several frames. We refer to these static scenes as *reference scenes*, and the number of reference scenes is denoted as  $S$ . For each of the reference scenes, several frames must be recorded. At the beginning of the noise distribution parameter estimation,  $S$  average images are computed. For each reference scene, the average image  $a_s$  is calculated by averaging all associated reference frames. The average images  $a_s$  are assumed to be the “ground truth”, i.e., the ideal pixel values for the actual observed static scene. It is now possible to determine the noise in each pixel of each reference frame relative to its associated average image. For each color channel and each intensity bin, one channel variation histogram is constructed. This is achieved by iteratively processing and counting the color channel variations in all pixels of all reference frames. The result of the histogram construction process are  $3 \cdot N$  histograms,  $histogram_i^j(v)$ . The content of each histogram entry represents the number of occurrences of the corresponding color channel variation. Finally, the noise distribution parameters are estimated from the computed channel variation histograms. We use maximum likelihood estimation for determining the mean values  $\mu_i^j$  and standard deviations  $\sigma_i^j$  of the measured noise data for all color channels  $i$  and all intensity bins  $j$ .

The computed noise distribution parameters are used in an adapted rendering pipeline that displays virtual objects with overlaid noise. Noise textures are generated during program initialization as a preparation step for the actual rendering algorithm. The noise textures are generated as RGB textures, in which each texel holds random channel variation values. These channel variations are generated such that they have a distribution corresponding to the measured noise mean values and standard deviations. A special random number generator is used so that random values with

a normal distribution of a given mean and standard deviation are generated (see [7]). This is not possible with the standard C++ random number generator, which produces an uniform distribution (i.e., each value in the random number range has the same probability). A user-defined parameter determines the size of the generated noise textures. For each of the intensity bins defined in the calibration step, a noise texture is generated.

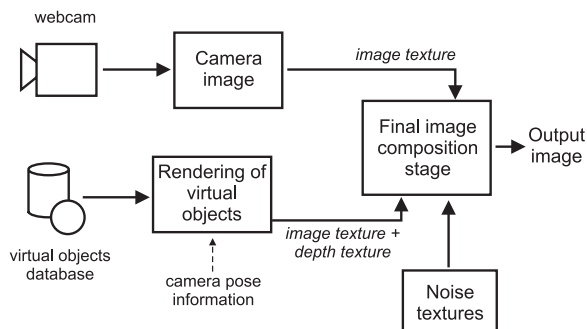


Figure 1: Diagram of the augmented reality rendering pipeline for overlaying noise over the virtual objects.

The noise textures are used in an adapted multipass AR rendering pipeline (see Fig. 1). In the first pass, the camera image is rendered. This is typically done by a special function of the AR framework. The camera image is then read back into texture memory. Before the second pass of the rendering pipeline is executed, the frame buffer is cleared to black. Next, the virtual objects of the augmented environment are rendered. The contents of the frame buffer are then again read back into texture memory. This time, however, both the contents of the color buffer and the depth buffer are copied into separate textures. The image data acquired in the first two rendering passes are then combined into the final output image using a specialized shader program, which was implemented in the OpenGL Shading Language.

For each pixel in the output image, this noise shader looks up the color and depth values at the corresponding location of the image textures from the previous rendering passes. It decides whether a virtual object pixel is to be rendered at the current coordinates by comparing the obtained depth value with the far limit of the OpenGL depth range. If the depth buffer value is less than the value of the depth range limit, a virtual object pixel is present. The color values of virtual object pixels are modified based on the pre-computed noise textures. Random variations are applied to the texture coordinates in order to create varying, animated noise. First, a random 2D translation is added to the texture coordinates. As the second variation, the texture coordinates are rotated by a random angle. The resulting, randomly modified texture coordinates are then used for looking up the channel variations in the noise texture. The average intensity of the virtual object pixel is used for selecting the noise texture corresponding to the associated intensity bin. Finally, the channel variations stored at the determined texture coordinates are added to the color of the output pixel.

After initial experiments, several refinements were introduced into the described noise rendering pipeline based on an empirical assessment of the visual appearance of the generated noise overlays. The first improvement makes it possible to adapt the size of the noise texels. In our experience, the color variations in the camera image often affect patches of several pixels. Therefore, a noise texel magnification factor was introduced, which can be selected by the user. As a second refinement, a user-definable noise scaling factor was added. The color channel variations obtained from the noise textures are multiplied with this factor. This way, the impact of the noise renderer can be adapted in order to achieve a stronger

effect. Finally, experiments with the original system revealed that the changes in the noise overlay happened too often, resulting in a strong flickering effect. Therefore, a frame counter with a user-definable update delay was introduced. This way, the random variations of the texture coordinates (i.e., random translation and rotation angle) are updated only after several frames. This results in a more slowly varying noise, which corresponds better to the noise in the camera image.

#### 4 ANTIALIASING

By using the data gathered in the described noise rendering pipeline, it also becomes possible to perform a smooth blending between virtual objects and the camera image. In order to perform the blending, a 3x3 neighbourhood of color and depth texels is looked up for each output pixel. For each of the texels in this area it is then decided whether a virtual object or the camera image is visible at that location. As described in Section 3, this can be determined with a simple comparison of the obtained depth value with the far depth range limit. This comparison yields a decision variable  $D(x, y)$ , which has a value of zero if a virtual object pixel has been detected and one if a camera pixel is present.

If a camera image pixel was detected at the currently regarded location, the corresponding camera image texel is used as output. Otherwise, the result of the blending operation is used. This blending is computed as shown in Equation 1. The blending operation is designed so that an averaging of color values is only done for boundary pixels, i.e., those virtual object pixels which have adjacent camera pixels. For such a virtual object boundary pixel, only the neighbouring camera image pixels are taken into account for the blending, while adjacent virtual object pixels are ignored.

$$b(x, y) = \frac{1}{n} \left( v(x, y) + \sum_{i=-1}^1 \sum_{j=-1}^1 D(x+i, y+j) \cdot c(x+i, y+j) \right) \quad (1)$$

A sum of camera image pixels  $c(x+i, y+j)$  is computed over the 3x3 neighbourhood in Equation 1. Only pixels with decision variable  $D(x+i, y+j)$  of one (i.e., camera pixels), are taken into account. The color of the central virtual object pixel,  $v(x, y)$ , is also added. The resulting summed up color data is then divided by factor  $n$ , which is equal to the number of pixels which were included in the summation (i.e., the number of adjacent camera image pixels plus one). The final result is the antialiased virtual object boundary pixel color  $b(x, y)$ . Due to this averaging of color values between virtual object boundary pixels and adjacent camera image pixels, a smooth blending is performed at the object boundaries. This effect is illustrated in Figure 2.

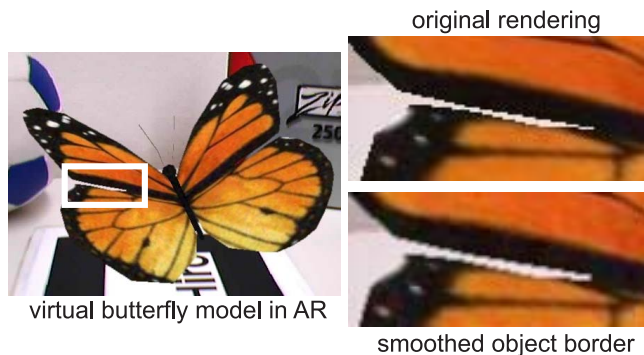


Figure 2: Smooth blending at the boundary of a virtual butterfly model. As shown on the right hand side, the aliasing between the camera image and the virtual object is reduced significantly by averaging the color values. (Here, a smoothing factor  $\beta = 0.8$  was used.)

As a refinement of the described blending procedure, a smoothing factor  $\beta \in [0; 1]$  was introduced for mixing the original virtual object color into the output pixel with a percentage of  $(1 - \beta)$ . In this way, the impact of the smoothing effect can be adapted.

#### 5 MOTION BLUR

A third effect which makes the camera image and virtual objects look different is motion blur. As explained in Section 1, motion blur results from the temporal integration of light intensity in the image sensor. If there is fast movement in the observed scene, a blurred camera image is captured. Here, we present a new display technique for virtual objects in AR, which takes motion blur into account. In order to be able to mimic the effects of motion blur in the camera image, the magnitude and direction of the blurring have to be known. We approximate the motion blur vector using the available camera pose information.

In a preprocessing step, the geometric center of each virtual model is computed. This is achieved by finding the bounding box of the model and calculating its center. During the runtime of the AR system, the blur vector is estimated in every frame. This approximated blur vector is defined as the two-dimensional motion of the center of the virtual object in image space. It is computed by projecting the object center into image space according to the current camera pose. The difference between the projected object centers in two consecutive frames is used as the blur vector.

Motion blur rendering is only activated if the length of the blur vector is greater than a threshold (typically between 5 and 10 pixels). Moreover, excessively long motion blur vectors resulting from an erratic pose estimation are ignored. If motion blur rendering is to be applied in a time step, an adapted display method is used. The simulated motion blur effect is created by repeatedly blending the virtual object image over the camera image at different positions. These positions are generated along the computed blur vector, which is centered at the current projection of the object center. Alpha blending is used during the rendering of the individual copies of the virtual object image. This way, the virtual object appears blurred from its current position along the blur vector.

When rendering the virtual object image, care has to be taken that the black background pixels are not included, which would lead to a darkened camera image. We solve this problem with a custom fragment shader program, which only outputs true object pixels based on a comparison of depth buffer values (similar to the decision variable  $D(x, y)$  in Sec. 4).

#### 6 RESULTS

We have tested the noise calibration method with a Sony Eye Toy USB webcam and a Unibrain Fire-i Firewire webcam. Figure 3 shows the standard deviations of the noise distributions for the red channel of the Sony Eye Toy camera. In these experiments, four intensity bins were used ( $N=4$ ). The mean values of the distributions are not depicted in the diagram. They are almost zero, as is to be expected from the differences between images in a video sequence and their average image. The computation of the distribution parameters in the noise calibration step typically takes between several seconds and some minutes, depending on the number and resolution of the reference frames. The estimated noise parameters are stored in a file and loaded during the initialization of the AR application.

The new rendering methods for AR were tested with several virtual models, one of them is the virtual model of a hamburger. Figure 4 shows the virtual hamburger model and the effect of the simulated noise overlay. Benchmark measurements have shown that the noise overlay method has a negligible impact on the frame rate. In a typical benchmark, the frame rate was only reduced from 29.8 fps to 29.3 fps.

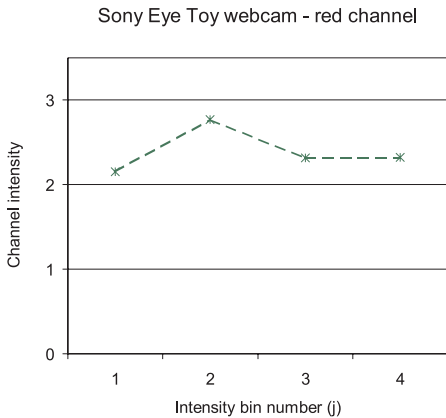


Figure 3: Standard deviations of the noise distributions for the red channel of the Sony Eye Toy webcam.

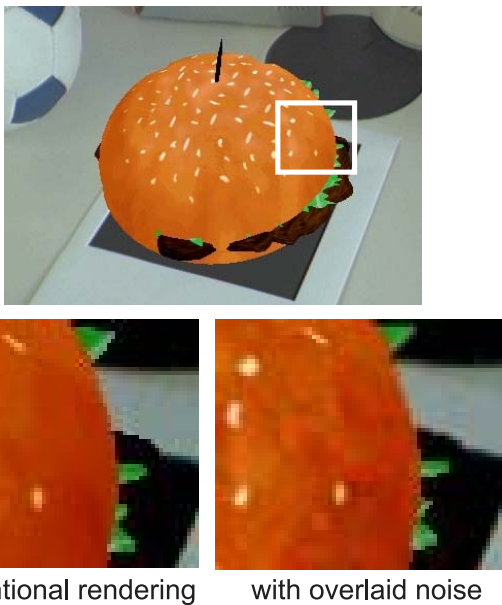


Figure 4: Virtual hamburger model in AR with simulated noise. (The contrast of the right image detail was enhanced to improve visibility.)

The effect of the new antialiasing method for AR was illustrated in Figure 2. In our implementation, the noise overlay and antialiasing techniques are combined into a single shader program. Therefore, the aforementioned noise shader benchmarks include the time required for the antialiasing method.

The effect of the motion blur rendering method on the virtual hamburger is depicted in Figure 5. Note that the simulated motion blur corresponds well to the real motion blur in the camera image. The motion blur technique has a measurable impact on the rendering performance, but can still deliver real-time frame rates. We measured an average frame rate of 24.4 fps for typical fast motions over a sequence of 520 frames.

## 7 CONCLUSIONS

In this paper, we have presented new techniques for dealing with three types of effects which make camera images and virtual objects easily distinguishable; image noise, aliasing, and motion blur. These new AR rendering techniques have been implemented on the GPU and achieve real-time frame rates.

The problem of specialized rendering methods is one of the main

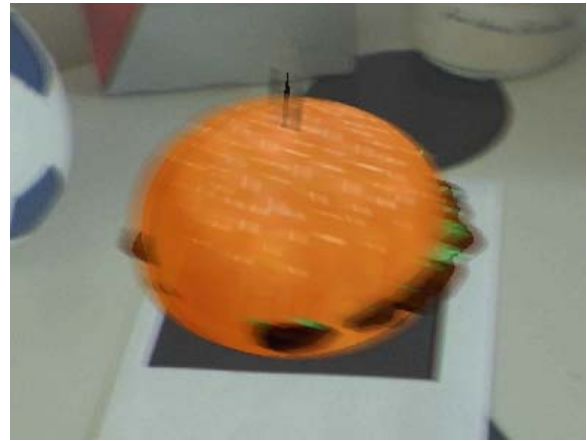


Figure 5: Motion blur applied to the hamburger model.

challenges in augmented reality. In this area, the task of equalizing the visual realism in the camera image and the virtual objects is very important. Nonetheless, the topic of adapted rendering algorithms for AR has received relatively little attention in the past compared to other main research directions. With this paper, we hope to contribute to the ongoing research in this area and inspire further investigations into the problem.

Please note that a detailed discussion of the algorithms presented in this paper is available in [5].

This work has been supported by project VIRTUE in the focus program on "Medical Navigation and Robotics" (SPP 1124) of the German Research Foundation (DFG).

## REFERENCES

- [1] K. Agusanto, L. Li, Z. Chuangui, and N.W. Sing. Photorealistic Rendering for Augmented Reality using Environment Illumination. In *Proc. of IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 208–216, October 2003.
- [2] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, and B. MacIntyre. Recent Advances in Augmented Reality. *IEEE Computer Graphics and Applications*, 21(6):34–47, November/December 2001.
- [3] P. Debevec. Rendering Synthetic Objects Into Real Scenes: Bridging Traditional and Image-Based Graphics With Global Illumination and High Dynamic Range Photography. In *Proc. of ACM SIGGRAPH*, pages 189–198, July 1998.
- [4] J. Fischer. *Rendering Methods for Augmented Reality*. Ph.D. thesis, Universität Tübingen, 2006.
- [5] J. Fischer and D. Bartz. Handling Photographic Imperfections and Aliasing in Augmented Reality. Technical Report WSI-2006-03, Wilhelm Schickard Institute for Computer Science, Graphical-Interactive Systems (WSI/GRIS), University of Tübingen, July 2006.
- [6] J. Fischer, D. Cunningham, D. Bartz, C. Wallraven, H. Bühlhoff, and W. Straßer. Measuring the Discernability of Virtual Objects in Conventional and Stylized Augmented Reality. In *Proc. of Eurographics Symposium on Virtual Environments*, pages 53–61, May 2006.
- [7] A. Fog. Non-uniform Random Number Generators. <http://www.agner.org/random/stoc.htm>, 2005.
- [8] M. Kanbara and N. Yokoya. Geometric and Photometric Registration for Real-Time Augmented Reality. In *IEEE and ACM International Symposium on Mixed and Augmented Reality Poster Proceedings*, pages 279–280, September 2002.
- [9] H. Kato and M. Billinghurst. Marker Tracking and HMD Calibration for a video-based Augmented Reality Conferencing System. In *Proc. of IEEE and ACM International Workshop on Augmented Reality*, pages 85–94, October 1999.
- [10] P.J. Withagen, F.C.A. Groen, and K. Schutte. CCD Characterization for a Range of Color Cameras. In *Proc. of Instrumentation and Measurement Technology Conference*, May 2005.