Compositing for Small Cameras

Georg Klein* David Murray†

Active Vision Laboratory
Department of Engineering Science
University of Oxford

ABSTRACT
To achieve a realistic integration of virtual and real imagery in video see-through augmented reality, the rendered graphics should have a similar appearance and quality to those captured by the video camera. This paper describes a compositing method which models the artefacts produced by a small low-cost camera, and adds these effects to an ideal pinhole image produced by conventional rendering methods. We attempt to model and simulate each step of the imaging process, including distortions, chromatic aberrations, blur, Bayer masking, noise and colour-space compression, all while requiring only an RGBA image and an estimate of camera velocity as inputs.

1 INTRODUCTION
Video see-through Augmented Reality attempts to insert virtual graphics into the real world, typically by blending some rendered image onto a video feed from a small hand-held or head-mounted camera. In some applications it may be desirable to have the virtual graphics appear as if they were part of the real world, and to create this illusion requires surmounting a number of challenges: tracking should be accurate and jitter-free, so that the graphics appear glued in place in the real world; occlusions between real and virtual objects should be correct; the lighting of the virtual objects should match that of the real world; and the quality and texture of the rendered pixels should match that seen in the video feed.

In this paper we address the last problem. Virtual graphics are usually rendered assuming that the camera is a perfect pin-hole device, when in reality the web-cams or other small devices often used for AR add many distortions and imperfections to the image. To convincingly blend the two images there are then two options: One is to somehow remove the distortions and imperfections from the captured image, but this is unrealistically difficult. The other is to artificially introduce imperfections into the rendered images, and this is the approach taken here. More specifically, this paper seeks to emulate the imaging process which occurs in small cameras with wide-angle lenses, such as the Unibrain Fire-i. There has been some previous work on this topic: Watson and Hodges [7] have shown that lens distortion can be emulated or corrected using graphics hardware; more recently Fischer et al [1] have shown that the integration of rendered graphics can be improved by adding synthetic noise and motion blur, and by anti-aliasing the blending seams between graphics and video. Both of these methods exploit ever-increasing GPU bandwidth to achieve their aims, and we follow the same approach.

To enable easy integration with existing rendering methods, we cast our compositing method as a post-rendering process: We operate on an ideal rendered image of the virtual graphics such as is typically produced by OpenGL. This image is warped and blurred according to lens, motion and sensor characteristics, and then resampled and degraded in accordance with sensor behaviour. We then blend the degraded image with the captured video data to produce the composited image. We attempt a principled simulation of camera effects based on data either learned (preferentially) from device specifications, off-line calibration or guesswork (when necessary). We show that lens distortions, chromatic aberrations, anti-aliasing filters, motion blur, Bayer interpolation, noise, quantization and colour-space conversion can all be modeled on today’s commodity graphics hardware.

The next section of this paper discusses previous related work. Section 4 discusses the real imaging pipeline of a small camera. In Section 5 we show how this pipeline can be simulated on the computer. Results and conclusions are presented in Section 6.

2 RELATED WORK
Grain matching (a simulation of the texture of film stock) is long-established in the off-line world of the movie industry, and the imaging process of digital cameras has been investigated in off-line contexts (e.g. [2]). In AR such sensor effects were only recently considered: Fischer et al [1] add noise and motion blur to the rendered image, and anti-alias the edges where real and virtual images meet. This paper can be seen as an extension of that work, in that we consider more sensor effects. However our method is also more general, in that it requires no special treatment of seams and transparently handles transparent areas; further, our more principled simulation of the imaging process means that some behaviours, such as the `splotchy’ nature of image noise, emerge naturally from our method and do not have to be simulated by hand.

Lens effects have been considered by a number of researchers. Radial distortion was discussed by Watson and Hodges [7] for counteracting the effects of HMD distortions, and their method has
been widely adopted for AR compositing: either to un-distort the camera image and then render graphics on top, or to distort the rendered graphics to match the distorted video feed. (It is notable that the latter operation is still frequently performed poorly, with dark or grey outlines appearing on the outside edges of rendered graphics. This is due to improper blending, as discussed in section 4.1.) Okumura et al. [3] demonstrate the real-time estimation and application of motion and defocus blur, resulting in noticeably more natural-looking graphics. Defocus blur is an important effect for cameras with large apertures and varying focus; it is less significant for small fixed-focus lenses such as used here. We do not consider it in this paper because it is difficult to simulate in post-processing, unless the rendered graphics are known to all be at the same depth.

3 The imaging pipeline of a Unibrain Fire-i

This section describes in practical terms the image formation process which takes place on small cameras, with emphasis on those steps which cause visible distortions or non-ideal effects to appear in the image. We use the Unibrain Fire-i camera fitted with a 2.1mm wide-angle lens as an example. Not only is this a popular setup widely used for real-time tracking work, it performs a large amount of image processing on-board and so produces complex image effects: This makes it a good example to use. Figure 1 illustrates a 640×480 YUV-411 frame (converted to RGB) captured by a stationary camera pointed at a printed piece of paper. The image shows a number of effects which have been introduced by the camera, and these arise from the image formation process, which is described in five steps:

1. **Lens effects:** Incoming light is focused onto the image sensor by the 2.1mm lens. This produces large quantities of barrel distortion, as well as some image softness (particularly in the image corners) and subtle vignetting (darkening of the corners and edges). Further, different wavelengths are refracted to varying degrees, resulting in chromatic aberrations, visible as the purple/yellow fringing in Figure 1. Bright lights would cause flare and very near objects would be blurred by defocus.

2. **Bayer mask:** The lens projects an image of the world onto a Bayer mask (or colour filter array, or CFA) which forms part of the camera’s sensor. Each of the sensor’s photo-sites are masked by a colour filter allowing either red, green or blue light to pass (on the Fire-i, this array is patterned as alternating rows of GGBG and RGGR). The colour filters are not perfect band-pass filters, so substantial colour crosstalk takes place, i.e. a red photo-site will also capture some blue and green light. Many Bayer filters also employ an anti-aliasing filter (a blurring light spreader) to avoid the formation of colour moiré patterns when observing high-frequency structure.

3. **Image sensor:** The Fire-i employs a CCD which converts to electrical charge the colour-filtered light incident on each photo-site during a finite exposure time. The integration time can be substantial (of the order of 30ms indoors) and this gives rise to motion blur. Each photo-site is further subject to thermal and shot noise. On the Fire-i, each photo-site’s integration period is simultaneous (global shutter); on some cameras with CMOS sensors, different rows integrate at different time periods (rolling shutter), which introduces image warping when motion is observed. The charge from each photo-site is converted to a digital signal in an analog-to-digital converter, introducing quantization noise.

4. **In-camera processing:** On the Fire-i, the 10-bit ADC output is fed to a video processing chip which performs a variety of image operations on the raw Bayer signal. Unfortunately documentation of these operations is not available, but they include sharpening, exposure control, Bayer interpolation (to reconstruct a full-colour image from the Bayer signal), colour processing (to counteract the colour channel mixing described earlier), further quantization, and colour-space conversion to YUV-411. The converted stream is sent to the host computer via an IEEE-1394 link.

5. **Colour-space conversion:** The YUV-411 stream contains independent luminance information (Y) for each pixel, but groups of four horizontal pixels share the same colour (U,V) information. The host computer converts this stream for processing and display, often producing a black-and-white image for tracking purposes and an RGB image for display.

4 Implementation

This section describes a method by which some elements of the above image formation process can be emulated on a computer. We start with a high-resolution image of virtual graphics rendered in OpenGL, and progressively down-sample, blur and degrade the image to produce the data which the camera would have measured at each Bayer photo-site, and subsequently blend and colour-space-convert the image together with the video input feed to produce a final 640×480-pixel blended image. We implement everything using OpenGL and the its shading language, GLSL. The input image needs to have an alpha (transparency) channel: at this point it is helpful to briefly review alpha blending and compositing in general.

4.1 A note on colour representation

We adopt the usual convention of an alpha value with a range from zero to unity indicating fractional pixel coverage: that is, an RGBA pixel encodes both an RGB colour as well as a fraction of the pixel’s area which is covered by that colour. When performing calculations such as convolution and interpolation on RGBA data it is however also important to store this data in an appropriate format; here, this means storing pixels using pre-multiplied alpha.

The use of pre-multiplied alpha has been commonplace in computer graphics since the compositing paper of Porter and Duff [8]. In the pre-multiplied representation, each pixel stores not the usual quadruplet of \( c = [r, g, b, \alpha] \) but rather the values \( \hat{c} = [\alpha r, \alpha g, \alpha b, \alpha] \), with all values in the range 0–1. This has the advantage that interpolation and summation over pixels becomes trivial: The average of pixels \( \hat{c}_1 \) and \( \hat{c}_2 \) is simply \( \hat{c} = \frac{1}{2} (\hat{c}_1 + \hat{c}_2) \), whereas \( \hat{c} (c_1 + c_2) \) would yield incorrect results when \( \alpha_1 \neq \alpha_2 \). Correct summation over pixels of different alpha values is crucial for the ability to handle transparent objects, and avoids the sometimes seen black or grey outlines around virtual graphics.

4.2 Processing

Our method requires three inputs per frame: a rendered pin-hole image of the virtual graphics to be displayed, the input video image, and an estimate of the camera’s rotational displacement during the frame’s exposure. The rendered image should be stored in pre-multiplied alpha: This either requires a trivial modification of blending modes when rendering, or a pre-processing step to multiply each colour by its alpha component. The image should also be sufficiently high-resolution to provide good detail across the distorted frame; with the wide-angle lens used here, this means we use a 3000×2250 pixel rendering. The transparent background should be of colour \( \hat{c} = [0, 0, 0, 0] \).

The rendered image is processed and blended with the video frame in eight distinct steps, which are shown in a flow-chart in Figure 2. These steps are now described in detail.

1. **Radial distortion:** The pinhole image is warped by rendering into a 2048×1536-pixel texture as a 24×18-cell grid. The coverage of both the input image and the new distorted image is slightly larger than the video frame: this provides the margin of pixels required for later blur and aberration steps, and is illustrated in the
(a) Anti-aliasing blur - a small quantity of blur is applied to emulate limited lens resolution and any anti-aliasing filter the sensor might have.

(b) Corner softness - we calculate image-varying softness by assuming that it varies with separation of the red and blur portions of the image as calibrated for chromatic aberration. That is, the higher the colour separation, the higher the blur.

(c) Motion blur - we use a 16-tap motion blur kernel in the next step, and for large blur lengths these taps can be more than one pixel apart. In such cases, we apply extra Gaussian blur to prevent visible banding.

4. Motion blur: On the same image grid as above, we estimate the local direction and magnitude of motion blur based on a tracking system’s estimate of the rotational motion during exposure. We consider only camera rotation and a static scene. Anything else is very difficult to achieve as a post-process, and the general case would require multiple geometry rendering passes. We apply motion blur as a 16-tap filter.

5. Bayer sampling: The blurred image is subsampled with a simulated Bayer mask. Instead of using a single 640 × 480-pixel Bayer image, we find it convenient to sample to three intensity+alpha images, since this produces shader code with no branches for this and later stages. We use two half-size images for the red and blue channels, and a partially-filled, 45° rotated green image. This allows exact pixel coverage and trivial Bayer interpolation in the next stage.

Each channel samples individually from the blurred image using a 12 × 9-cell grid, which slightly distorts the sampling process to account for chromatic aberration. (Aberration can be easily calibrated by pointing the camera at a white noise print-out, and performing Lucas-Kanade tracking across colour channels, taking the green channel as the baseline.) Each pixel is further corrupted with Gaussian noise: We use a noise texture to obtain normally distributed noise samples, and scale these according to a calibrated linear function of each pixel’s intensity. Pixel dynamic range is then slightly reduced to match the camera’s own output range, and finally each pixel is quantized to 6-bit resolution.

6. YUV Blending: The rendered image is blended into the input video frame. At this stage, each rendered colour channel has its own alpha channel, which is interpolated from the CFA images alongside the colour component. We employ bilinear Bayer interpolation, since this is very fast. Ideally we would like to use the same method as used by the camera, but this method is unknown.

The blending procedure effectively converts the YUV frame to RGB, blends it with each rendered colour channel in turn, and then converts back to YUV. However we rearrange these operations to ensure that source video pixels not covered by graphics (α=0) remain completely unchanged. The output is a composited 640 × 480-pixel image with a YUV triplet per pixel.

7. Chroma split and squash: To match the colour resolution of the video input, the UV-component of the blended image is horizontally subsampled by a factor of four using a uniform box filter.

8. Chroma recombine: The blended image is converted to RGB, using the full-resolution luminance (Y) information and the low-resolution chrominance (UV).

If one were to use a camera with raw Bayer output, blending could of course be performed directly on the Bayer signal, before interpolation, for optimal results.
5 RESULTS

We have integrated the above compositing method into a markerless tracking system [3]. On a desktop computer with a fast graphics card (nVidia GeForce 9800 GTX) the new compositing method adds 4ms of rendering overhead per frame compared to our previous method, which undistorted the video frame, drew graphics into it, and re-distorted the composite. The extra overhead can rise to 5.5ms in scenes with large motion blur. On more modest mobile hardware (nVidia GeForce 8600M/GT) the overhead is substantial at 14ms extra rendering cost per frame. However, this does not necessarily impact frame-rate, since compositing takes place on the GPU while the CPU already tracks the next frame.

Figure 3 shows some side-by-side comparisons of the proposed compositing method with our previous approach. Full-resolution versions of these images, as well as video results, are included in the accompanying material. As well as some simple inserted objects, the results show a texture of some test patterns superimposed next to a printout of the same texture. In most instances the new method provides superior integration of the real and virtual images.

However, the method is not without drawbacks. Some problems arise from the decision to apply sensor effects as a post-process step: for example, objects which should appear stationary in the image are still blurred, which looks unnatural. Further, items which are very bright because they are user interface components (which should stand out from the background) are treated the same way as any other graphics, and the resulting look is dull. Solving these drawbacks would require modifications to the 3D rendering stage.

Further problems include mis-estimation of camera velocity by the tracking system (this is likely due to the assumption of zero acceleration, which is often wrong) which causes a motion blur mismatch between real and virtual objects. Beyond this, many sensor effects are not simulated quite correctly: for example, the real chromatic aberrations are more purple than the simulated versions, and the Bayer effects on the text do not match. This is partially due to a lack of understanding of the processes in the camera, and partially due to implementation and calibration constraints.

6 CONCLUSIONS

This paper has presented an attempt to simulate the behaviour of small web-cams by post-processing the ideal images produced by the standard OpenGL pipeline. Results show that the integration of real and virtual graphics can be improved by simulating some of the various artefacts that give the video image its characteristic look. That said, the method as presented here is not currently flexible enough to allow the fine-tuning of the simulated effects on a per-object basis. Further, the method requires powerful graphics hardware. A more lightweight approach, perhaps implementing only a subset of the steps described here, would currently be more appropriate for mobile hardware.

ACKNOWLEDGEMENTS

This work was supported by EPSRC grant GR/S97774/01.

REFERENCES