Real-time Non-Photorealistic Viewfinder on the Tegra 3 Platform

Tony Hyun Kim*
Stanford University
Department of Electrical Engineering

Irving Lin†
Stanford University
Department of Computer Science

Abstract

Non-photorealistic rendering (NPR), or "stylization," of an image is an important technique in computer graphics that is relevant to a variety of applications, such as effective visual communication, augmented virtual reality, and image compression.

In this report, we present an NPR viewfinder built on top of the FCam framework for mobile computational photography [Adams et al. 2010]. The NPR viewfinder stylizes live, 640×480 video in real-time at over 10 frames per second. We achieve this performance by making use of the tablet's GPU hardware for the relevant image processing steps. Specifically, we perform bilateral filtering, edge detection, and color quantization on the GPU. Our software architecture for GPU integration is a lightweight addition to the FCam framework, and can be used to implement other GPU-based image filters in addition to the NPR filters described here.

In contrast to off-line processing, our on-line NPR viewfinder can leverage additional hardware of the mobile camera for novel effects. As proof-of-principle, we demonstrate the use of FCam's flash hardware to separate the foreground from the background within a scene. We then use this segregation to apply NPR selectively, *e.g.* stylizing only the scene background.

Keywords: Non-photorealistic rendering, image abstraction, computational photography, mobile photography

Links: DL PDF

1 Introduction

Non-photorealistic rendering (NPR), or "stylization," describes a suite of techniques in computer graphics that allows for artistic interpretation of an image at the expense of photorealism. These techniques typically promote simplicity and reduction of unnecessary detail, which can better emphasize the semantic content of the image at hand. As a result, automated algorithms generating NPR images have been investigated in computer vision and related fields that accentuate, for example, perceptually important shapes and boundaries [Winnemöller et al. 2006]. Our current work on a real-time NPR viewfinder on a mobile device is motivated by both the computational usefulness and the aesthetic merits of image stylization. (See Fig. 1 for example images taken by our real-time NPR viewfinder.)

*e-mail:kimth@stanford.edu
†e-mail:irvingl@stanford.edu

When images are stylized at the time of capture on a mobile device, there is the potential to derive additional information from the scene that enables novel and/or efficient NPR effects. Consider the process by which a skilled artist stylizes an image "by hand." Typically, the artist draws upon their visual and physical knowledge of the subject in order to identify and subsequently emphasize the distinguishing elements of the object at focus [DeCarlo and Santella 2002]. In contrast, computational algorithms for image stylization often cannot take for granted the luxury of prior knowledge, and must instead rely on computationally expensive (and potentially less robust) image processing routines to compensate for the lack of scene understanding. On a mobile camera, however, one may acquire multiple captures of the scene at hand under various conditions, and then apply the general technique of image fusion to identify highlights, shadows, textures, depth maps, and other auxiliary information associated with the scene.

In this paper, we present a real-time NPR viewfinder on a Tegra 3 tablet. Our application is built on top of the FCam framework for mobile computational photography [Adams et al. 2010]. The NPR algorithm consists of a sequence of image filters – most importantly bilateral filtering and edge detection – that are applied on live video with GPU acceleration.

The paper is organized as follows. In Section 2, we describe prior work on image stylization that inspired our current project. In Section 3, we describe our lightweight addition to the FCam framework that enables image filters to be applied by GPU hardware. Note that this system is generic and can be used to implement other filters on the GPU. Section 4 provides implementation details for each image filter in our NPR algorithm. In Section 5, we integrate our NPR viewfinder with the flash mechanism of the tablet. Specifically, we use a stream of flash/no-flash video to efficiently segment the image foreground and background. The NPR filter is then applied selectively on the scene background.

2 Prior work

2.1 Mobile computational photography

The NPR viewfinder is built on top of the FCam framework for mobile computational photography. FCam on the Tegra 3 provides the foundational framework for accessing camera hardware, as well as a complete user-interface for camera control (*e.g.* exposure settings, ISO gain, etc.). The provided FCam codebase allowed us to focus immediately on the NPR viewfinder implementation. However, at the time of our development, certain aspects of the FCam specification – such as software control of flash timing – were unsupported on the Tegra 3 prototype, which generally hindered our efforts to integrate more of the tablet's hardware into the NPR routine. (See Section 5 for further discussion.)

2.2 Bilateral filtering for NPR

Computational algorithms that achieve "cartoon"-like stylization are often based on image filters that achieve: (1) color simplification, in which similar colors are clustered together; (2) identification of object boundaries; and (3) color quantization [Winnemöller

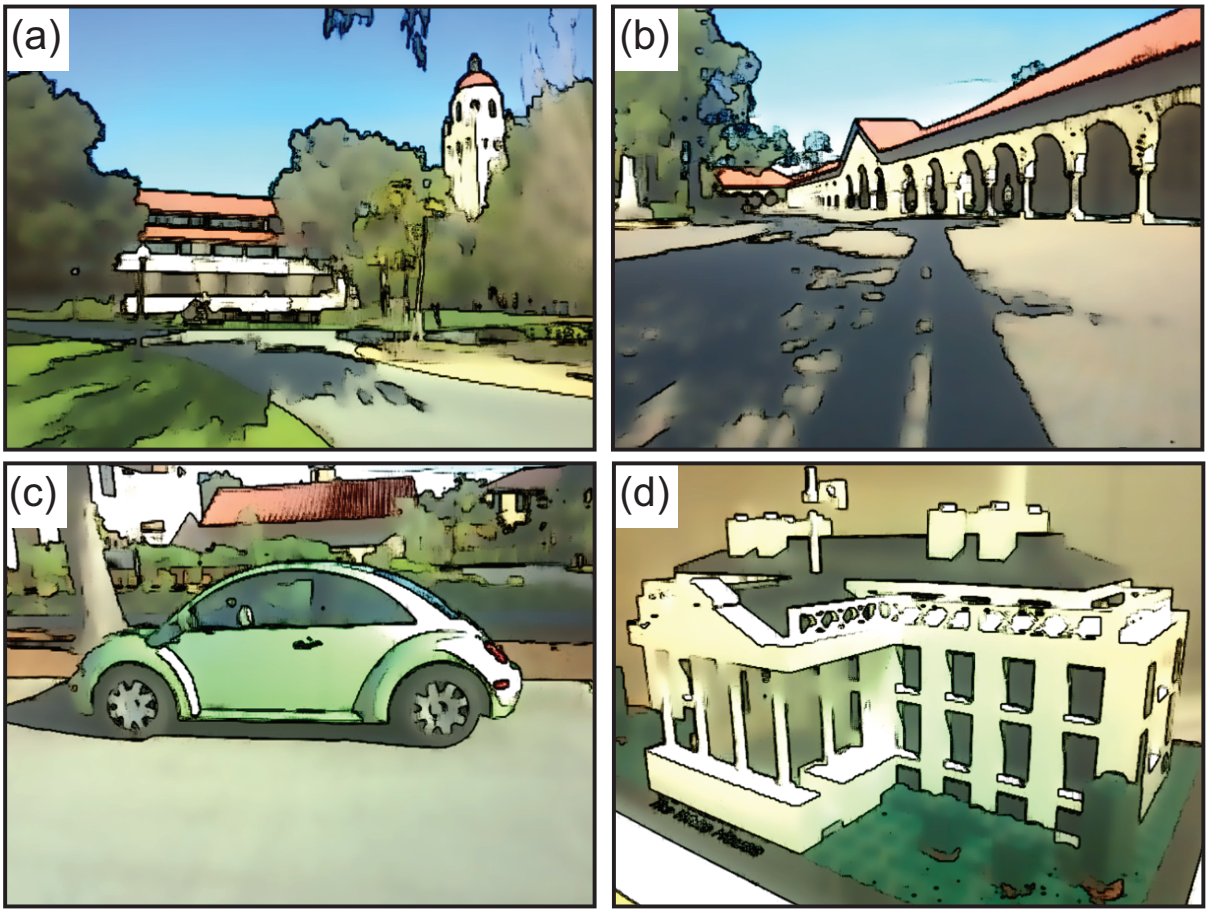

Figure 1: *Typical results from our NPR viewfinder. All examples are generated in real-time on the Tegra 3 tablet. In general, we find that our NPR algorithm, which consists of a sequence of image filters as described in Section 4, performs well with well-lit and colorful scenes. Scenes (a), (b), and (c) are taken outdoors on the Stanford campus, while scene (d) is an indoor shot. Note that panel (d) shows a number of glitches such as missing edges (*e.g. *the columns) and unphysical blurring (*e.g. *the flag). These glitches are common artifacts of bilateral filtering, particularly on noisy images, which is a key component of our NPR algorithm.*

et al. 2006]. In our NPR algorithm, edge detection and color quantization are performed in a very straightforward way (as described in Section 4). On the other hand, we borrow a number of ideas from the literature for the color simplification step.

The basic mechanism for color simplification is the bilateral filter [Tomasi and Manduchi 1998] which, in its general form, can be prohibitively expensive for real-time applications. Hence, the bilateral filter in our NPR viewfinder makes use of several approximations that were previously proposed by Refs. [Pham and van Vliet 2005] and [Fischer 2006].

Pham *et. al.* introduce the idea of a separated bilateral filter, which is a two-dimensional filter that is formed by the convolution of two one-dimensional bilateral kernels. Recall that the true two-dimensional bilateral filter is not spatially separable. Denoting by K the spatial extent of the kernel, only $O(K)$ texture fetches are required to compute each output pixel under the separable approximation, rather than $O(K^2)$ fetches in the true bilateral filter.

In addition, following the example of [Fischer 2006], we omit the spatial part of the bilateral filter entirely. With this simplification, the "bilateral" kernel weight between two pixels is given by

$$w = \exp -\frac{|u-v|^2}{2\sigma_C^2}, \tag{1}$$

where u and v are the pixel values (*e.g.* YUV), and σ_C is a parameter of the filter. Note that Eq. 1 does not take into account the spatial separation between pixels u and v. The primary advantage of this approximation is that the filter computation requests fewer evaluations of the exponential function. On the other hand, to compensate for the lack of spatial decay, we are typically limited to small kernel sizes ($K \approx 3$ pixels). It thus follows that multiple passes of the approximate "bilateral" filter is required to achieve a visual effect comparable to the true bilateral filter.

2.3 Hardware integration for NPR

A key inspiration for our work on mobile NPR were the beautiful NPR images computed with the aid of depth-edge maps in Ref. [Raskar et al. 2004]. By equipping an off-the-shelf camera with a custom array of flash sources, Raskar *et. al.* demonstrate an elegant and computationally efficient method for obtaining depth-edge maps associated with a scene. The depth-edges are then used as inputs to their NPR algorithm.

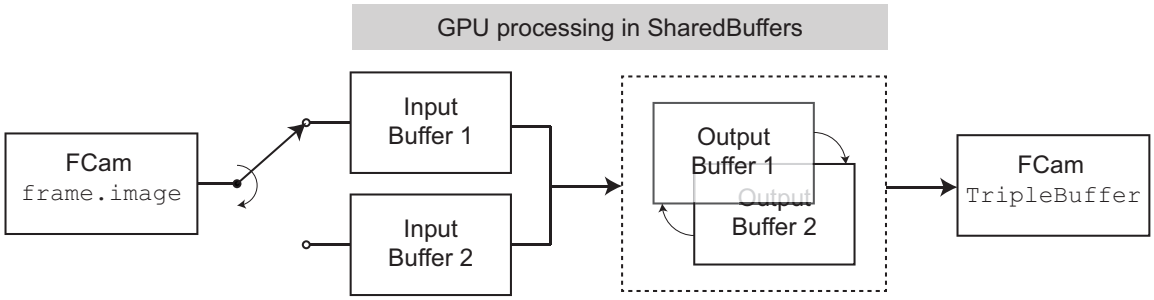

Figure 2: *Conceptual illustration of our GPU-based workflow for image processing, which resides within the main "FCamAppThread" loop of FCam. At each iteration of the FCam loop, a single* 640×480 *viewfinder image is retrieved (shown as "frame.image"). This image is (deep) copied into one of possibly many SharedBuffer containers, which can be accessed by both the CPU and GPU. A previously specified sequence of GPU shaders is then applied to the input image, with intermediate results stored in alternating fashion between two temporary SharedBuffers. The final result is transferred to FCam's "TripleBuffer," where the image will be polled by FCam's rendering routine.*

As the Tegra 3 tablet includes a stereo camera pair and a single flash, we had originally believed (incorrectly) that the method of Raskar *et. al.* could be directed ported to our device. While this notion turned out to be false, we continued to explore the use of flash in the stylization process. Ultimately, we developed an NPR viewfinder that uses a stream of flash/no-flash images to perform selective stylization of only the scene background (or foreground).

3 GPU workflow

Our workflow for GPU-based image processing is implemented in the main "worker" loop of FCam (*i.e.* `FCamAppThread` in `FCamInterface.cpp`), and is described below.

3.1 Initialization of shader sequence

The NPR algorithm consists of a sequence of image filters that are applied sequentially on each frame of the viewfinder video. For example, the results shown in Fig. 1 are obtained by two iterations of (approximate) bilateral filtering, followed by gradient-based edge detection. For performance reasons, all of our image processing filters are implemented as GPU shaders.

Upon initialization of FCam, we load and compile all GPU shaders, and then specify the sequence of filters that will be applied on each frame. For example, the following code snippet from `FCamAppThread` initialization

```
char shaderPaths[NUM_SHADERS][] = {
  "/data/gpgpu/bilateral_x.frag",
  "/data/gpgpu/bilateral_y.frag",
  "/data/gpgpu/edge.frag",
};
GLuint programs[NUM_SHADERS]
for(int i=0; i<NUM_SHADERS; i++)
  setup_shader(shaderPaths[i], programs[i])
```

requests `bilateral_x`, `bilateral_y`, and `edge` shaders to be loaded from disk. Each shader is then compiled and assigned an OpenGL program index, which is stored in the corresponding entry of the `programs` array.

Subsequently, we specify the exact sequence of shaders to run on each viewfinder frame. For example,

```
GLuint shaderSequence[] = {
  programs[0], programs[1],
  programs[0], programs[1],
  programs[2]};
```

corresponds to two rounds of bilateral filtering followed by edge detection.

The previous two steps – (1) loading/compiling of shaders, and (2) specification of `shaderSequence` – are the only requirements to "hook-up" a sequence of GPU shaders to the viewfinder. No further changes to the GPU workflow code is necessary! So, note that this mechanism can implement arbitrary image filters on the GPU, in addition to the NPR-related filters discussed in this paper.

3.2 Shader mechanism

At each iteration of the `FCamAppThread` loop, a single 640×480 viewfinder frame is retrieved from the camera sensor in YUV colorspace. The image data associated with this single frame is shown as `frame.image` in Fig. 2.

The capture data in `frame.image` is transferred to one of the `SharedBuffer` containers (denoted as "Input Buffer" in Fig. 2), which are buffers that can be accessed by both the CPU and the GPU. In Fig. 2 we show two input `SharedBuffer` buffers that are designed to store a flash and a non-flash image of the scene respectively. (The two captures are obtained through two separate iterations of the `FCamAppThread` loop. Slight control overhead is required to steer the incoming frame towards the appropriate input buffer.) For a 640×480 image, the "copy-to-GPU" (`frame.image`→`SharedBuffer`) time is about 5 ms. See Table 1 for additional timing details.

Once `frame.image` has been loaded in a `SharedBuffer`, we run the requested sequence of GPU shaders (as specified in `shaderSequence`), with the intermediate results being ping-pong'ed between two temporary `SharedBuffers`. When the shader sequence is complete, the final result is deep-copied back into FCam's `TripleBuffer`, where the image will be accessed by the viewfinder rendering routine of FCam. The "copy-to-CPU" (`SharedBuffer`→`TripleBuffer`) time is typically 30 ms.

4 NPR shaders

As discussed in Section 2, cartoon-like stylization of a photograph can be achieved by promoting: (1) color simplification through bilateral filtering, (2) enhancement of object boundaries by edge detection, and (3) color quantization. In this section, we provide implementation details for each of our image processing shaders.

Figure 3: *A viewfinder image processed with two iterations of the approximate bilateral filter with parameters ($K = 3$, $\sigma_C = 0.02$).*

4.1 Bilateral filtering

Our implementation of the bilateral filter makes use of the approximations in Refs. [Pham and van Vliet 2005] and [Fischer 2006]. Specifically, we use a separable approximation to the two-dimensional bilateral filter, and altogether omit the spatial dependence of the bilateral filter weights (as shown in Eq. 1).

The bilateral filtering step has three parameters:

- The number of passes of the approximate bilateral filter,
- The radius K of the filter kernel,
- The selectivity of the filter to a particular color: σ_C.

Consider the front facade of the White House model in Fig. 3. We observe notable flattening of colors and textures already with two passes of a relatively small ($K = 3$) bilateral kernel.

4.2 Edge detection

The edge detection step assigns a sharpness score to each pixel by computing the YUV differential against the pixel's up, down, left, and right neighbors. The resulting score is passed through OpenGL's built-in `smoothstep` function, which mainly acts to filter out weak, spurious edges. We replace the luminance value Y of the candidate pixel by

$$Y' = 1 - \text{smoothstep}(\text{sharpness score}) \qquad (2)$$

which introduces "soft" (*i.e.* varying-strength) edges to the color-simplified image.

The parameters of edge detection are two floating-point values `edge0` and `edge1`, which are provided to `smoothstep`. Intuitively, their roles are:

- `edge0` controls the sensitivity of edge detection. Namely, any sharpness score below `edge0` is clamped to 0 by `smoothstep`.
- `edge1` (in conjunction with `edge0`) then determines the mapping of the sharpness score to the edge luminance. Any score larger than `edge1` is clamped at 1 by `smoothstep`.

4.3 Color quantization

Following color simplification by bilateral filtering and edge detection, we can (optionally) obtain a "cell-shaded" appearance by quantizing the YUV values taken by the pixel. (Note: quantization is not necessary for the basic cartoon effect. The results of Fig. 1 were not passed through the color quantizer.)

Task	Elapsed time (ms)
Copy-to-GPU (`frame.image`→`SharedBuffer`)	4.9 ± 1.7
Bilateral filtering (Single xy-pass; Kernel radius $K = 3$)	30.3 ± 9.9
Edge detection	8.2 ± 5.3
Color quantization	15.9 ± 3.6
Image mux	29.9 ± 8.7
Copy-to-CPU (`SharedBuffer`→`TripleBuffer`)	31.9 ± 8.6

Table 1: *Typical elapsed times for various sub-steps of the NPR algorithm on a single* 640×480 *viewfinder frame. Error bars are standard deviations over* 100 *measurements.*

The quantization filter takes two parameters:

- The number of quantization buckets for Y,
- The number of quantization buckets for U and V.

Note that the pixel values are quantized after the edge detection step (rather than vice versa). This ordering is to prevent the formation of false edges at the boundaries between quantized levels.

4.4 Image multiplexing

This is a simple shader that takes two image sources, and renders an output texture where each pixel is derived from one of the two input images based on some per-pixel condition.

When combined with a flash/no-flash video stream, the multiplexer mechanism can be used for foreground/background-selective stylization. The basic idea is as follows. The mux is provided both the original image and the stylized image. For each pixel, the mux determines whether the pixel belongs to foreground or background with the aid of a flash/no-flash pair. If the pixel belongs to background, then its value is sampled from the stylized image. Otherwise, the mux selects the original (non-stylized) source.

4.5 User interface

We provide additional controls to the FCam GUI so that the parameters relevant to all image filters can be adjusted on-the-fly and adapted as necessary to the scene conditions. We also provide the ability to store and load "presets" (*i.e.* a complete set of filter parameters) which allows convenient comparisons between different NPR viewfinder settings.

4.6 Typical performance

Table 1 lists typical times required for the individual steps of the GPU workflow. The elapsed time for the image processing steps can be reduced beyond the values cited here, by hard-coding the filter parameters into the shader (instead of setting them at run-time through user interface hooks).

The basic NPR viewfinder (corresponding to Fig. 1) consists of the two CPU/GPU copies, two passes of bilateral filtering, and edge detection. The aggregate latency is on the order of 100 ms, which corresponds to 10 frames per second.

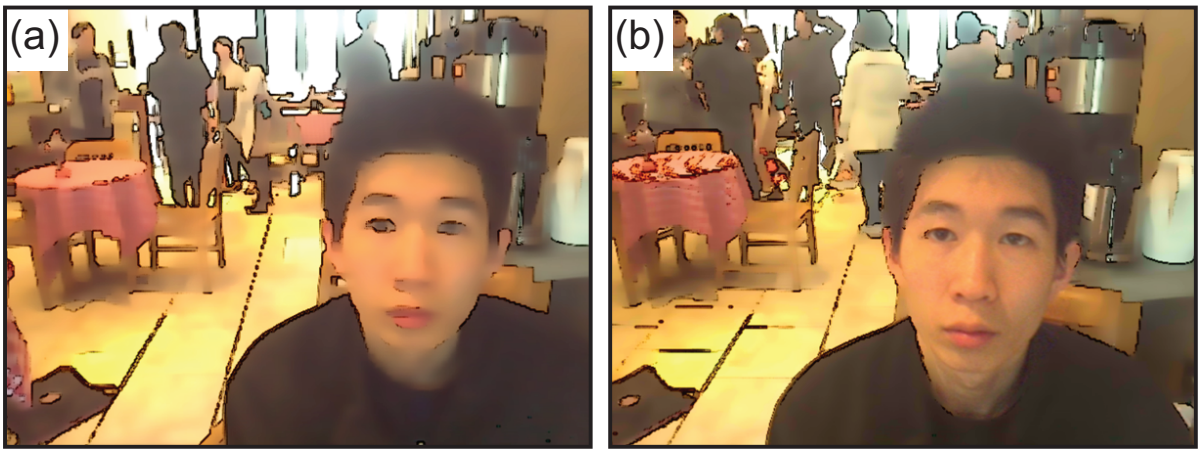

Figure 4: *Demonstration of foreground/background-selective stylization by using a flash/no-flash video stream. (a) The standard NPR viewfinder that stylzes the complete frame. (b) Selective stylization of only scene background.*

5 Hardware (flash) integration

One of the most exciting aspects of mobile computational photography is the convergence of powerful computational capability (for image processing) with a suite of sensors (imaging and otherwise) on a single platform. In this spirit, we discuss some of our experimental results with the flash hardware on the Tegra 3 tablet.

5.1 Acquisition of a flash/no-flash pair

As illustrated in Fig. 2, the underlying GPU workflow provides two input `SharedBuffers` that are intended to store a pair of flash and no-flash images. Recall that a single iteration of the `FCamAppThread` loop pulls exactly one image from the sensor. Hence, two iterations of the loop are required to obtain a completely new flash/no-flash pair.

(Note that we do not perform any explicit registration between the flash and no-flash images. Thus, glitches are expected if the scene is significantly altered between the flash and no-flash captures.)

5.2 Joint bilateral filtering

As an initial sanity check, we modified the bilateral filter (Section 4.1) to perform joint bilateral filtering on the flash/no-flash pair [Petschnigg et al. 2004]. We found image fusion results that were comparable to those in the "Hello ImageProcessing" assignment earlier in CS 478.

5.3 Flash-based, background-selective NPR

The basic principle of operation underlying flash-based foreground-background (FG-BG) segmentation is the observation that the flash effect is most pronounced for objects that lie directly ahead of the camera, *i.e.* in the foreground. This is especially true for a direct, head-on flash such as the one present on the Tegra 3 tablet.

Hence, an extremely naïve method for foreground-background segmentation is as follows. First, a no-flash and a flash image pair is taken in rapid succession. Next, we compute the pixel-wise difference between the image pair. Finally, we threshold the difference image in order to classify each pixel as belonging to the foreground or the background. Here, we are assuming that the flash will cause the foreground pixels to be significantly brighter than in the no-flash case, while the background pixels are assumed to be invariant to flash action.

The segmentation of the image into foreground and background can then be used to perform, for example, background-selective NPR. We provide the FG-BG classification as the selector to the multiplexer shader (Section 4.4), which samples from the original or the stylized image accordingly. Fig. 4 shows a typical result where the foreground object is sampled from the original image, while the background scene is derived from the stylized image.

5.4 FCam limitations

Despite the basic result shown in Fig. 4, we experienced a number of difficulties in integrating the flash hardware for background-selective NPR. The difficulties were due to the incomplete implementation of FCam on the Tegra 3 tablet.

Firstly, the provided FCam implementation did not provide software control over the flash duration, and we observed significant variations in the flash duration from shot to shot. Empirically, it was not unusual to see the effects of a "lingering flash" in the no-flash reference, which completely breaks the working assumption underlying flash-based FG-BG segmentation.

Secondly, the FCam implementation did not correctly support "tagging" of frames with the flash parameters. Given the asynchronous shot requests in the FCam architecture, the ideal implementation of flash/no-flash acquisition would inspect each retrieved frame for its flash tags, and steer the image towards the "flash" `SharedBuffer` or the "no-flash" `SharedBuffer` accordingly. As it turns out, the tagging of viewfinder frames was incomplete, and we had to rely on an open-loop steering method that was not actively synchronized with respect to the shot requests.

6 Conclusion and Future work

We have demonstrated a real-time NPR viewfinder for FCam on the Tegra 3 tablet. With hard-coded filter parameters, our system generates cartoon-like renderings of live video at over 10 frames per second. The NPR results are particularly convincing on outdoor scenes on a nice day. In addition to the results shown in this paper, many more NPR images are available on our development blog:

`http://cs478.blogspot.com`

Underlying the NPR viewfinder is a general framework for performing image processing with GPU hardware on the Tegra 3. Our GPU-based workflow is general and can be used to implement algorithms other than the NPR filters described in this paper.

We also showed how the flash hardware on the Tegra 3 tablet can be used for foreground/background-aware NPR. We expect that flash-based selective stylization can be made more robust with future (*i.e.* more complete) iterations of FCam.

Below, we list some items for future work:

6.1 Performance

We have demonstrated an NPR viewfinder that renders live video at roughly 10 frames per second. Clearly, it would be desirable to increase the frame rate to > 30 frames per second. Some ideas for optimization include:

- Performance tweaks to the GPU shaders. Namely, all of our shaders are currently implemented in the most literal interpretation of each filter. With additional knowledge of the GPU hardware, it may be possible to map the filter to the GPU more efficiently. For instance, Ref. [Fischer 2006] suggests implementing the exponential function (Eq. 1) as a 2D texture lookup in (u, v) rather than a call to the `exp` function.
- Downsampling the NPR image. Since the stylized image is typically less detailed than the original source, it may be possible to maintain visual quality even if the NPR image were downsampled.

6.2 Hardware

Previously, we remarked that the flash-based selective-NPR suffered from an incomplete implementation of FCam on the Tegra 3 tablet. With future support, we would be able to acquire alternating flash/no-flash streams much more robustly. In such a case, we would be inclined to attempt even more advanced methods for FG/BG classification, *e.g.* the use of pairwise potentials, etc.

On the other hand, while flash/no-flash pairings provide a rough classification of the scene into foreground and background, the strobing flash is particularly grating on the eyes, and generally not very practical in public situations. Instead, the potential use of the stereo cameras for depth estimation is a very enticing alternative.

6.3 Applications

Supposing that the NPR frame rate can be improved to > 30 frames per second, it would be extremely fun to devise a user interface that allows one to alter the parameters of the NPR filters fluidly in real-time by directly touching the scene on the viewfinder.

We are also interested in applications that build on top of real-time NPR. For instance, the work of Ref. [Fischer 2006] is motivated by the use of NPR to achieve a convincing augmented reality, *i.e.* to use NPR stylization to blur the distinction between real video and computer-generated insets. This technique could, for instance, give rise to mobile games that take visual cues from the tablet's environment without incurring the jarring effect from the juxtaposition of real video and computer renderings.

(We actually went driving once with the NPR viewfinder... The view out the front windshield was very reminiscent of old-school driving games!)

Acknowledgements

We thank Nvidia for providing the Tegra 3 loaner tablet and for their technical support. We thank the CS 478 staff for introducing the subject of mobile computational photography. THK thanks Timo Stich of Nvidia for providing starter code for GPU integration on the Tegra 3.

References

ADAMS, A., TALVALA, E.-V., PARK, S.-H., JACOBS, D. E., AJDIN, B., GELFAND, N., DOLSON, J., VAQUERO, D., BAEK, J., TICO, M., LENSCH, H. P. A., MATUSIK, W., PULLI, K., HOROWITZ, M., AND LEVOY, M. 2010. The frankencamera: An experimental platform for computational photography. In *SIGGRAPH*.

DECARLO, D., AND SANTELLA, A. 2002. Stylization and abstraction of photographs. In *SIGGRAPH*.

FISCHER, J. 2006. *Rendering Methods for Augmented Reality*. PhD thesis, Universität Tübingen.

PETSCHNIGG, G., AGRAWALA, M., HOPPE, H., SZELISKI, R., COHEN, M., AND TOYAMA., K. 2004. Digital photography with flash and no-flash image pairs. *ACM Transactions on Graphics 23*.

PHAM, T. Q., AND VAN VLIET, L. J. 2005. Separable bilateral filtering for fast video preprocessing. In *IEEE International Conference on Multimedia & Expo*.

RASKAR, R., HAR TAN, K., FERIS, R., YU, J., AND TURK, M. 2004. Non-photorealistic camera: Depth edge detection and stylized rendering using multi-flash imaging. *ACM SIGGRAPH*.

TOMASI, C., AND MANDUCHI, R. 1998. Bilateral filtering for gray and color images. *Computer Vision*, 839–846.

WINNEMÖLLER, H., OLSEN, S. C., AND GOOCH, B. 2006. Real-time video abstraction. *ACM Transactions on Graphics*.