CS 478: Hello ImageProcessing

Tony Hyun Kim

February 5, 2012

1 Face-based autofocus

1.1 Autofocus algorithm

My `MyAutofocus` algorithm is ported literally from my HelloCamera assignment. It performs a linear search over the space of image focus. The current face-recognition extension integrated well with my previous autofocus implementation since the latter had already incorporated a region-of-interest (ROI) `Rect` internally. Now, I simply hook the candidate ROI from face-classification to my `MyAutofocus` object.

1.2 How to handle multiple faces

If multiple ROI candidates are returned by `MyFaceDetector`, then I increase the threshold `minNeighbors` in the classifier routine (`detectMultiScale`) until all but one, presumably-best candidate remains. With my scenes, this strategy worked robustly.

1.3 False positives

My strategy of always seeking the best face candidate provides protection against intermittent false positives. If the routine does happen to select a wrong target, I can simply re-toggle face-based autofocus.

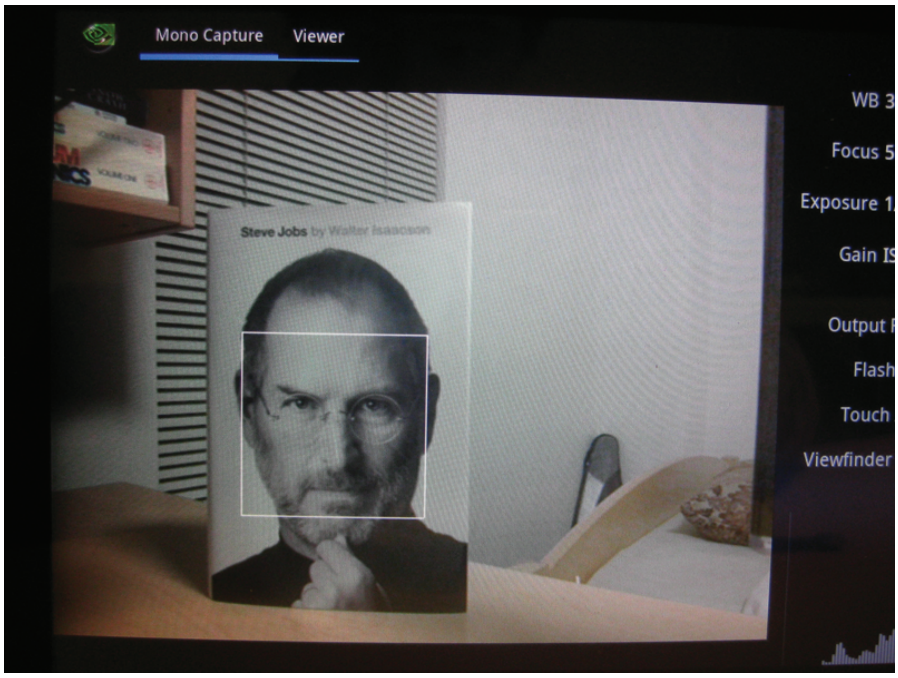
1.4 Face too out-of-focus

I have found that, if there is a face in the viewfinder, my implementation is quite robust. However, in the chance that `MyFaceDetector` does not suggest any candidates – because the face is too out-of-focus or for any other reason – I decrement `minNeighbors` until a candidate ROI is generated. I have observed that, when `minNeighbors` is set to 0, plenty of candidates are generated even from a scene containing no faces.

1.5 Gallery

I thought the idea of face-censoring was neat, so I hacked an implementation. This (and basic face-based autofocus) is shown in Fig. 1.

(a)



(b)

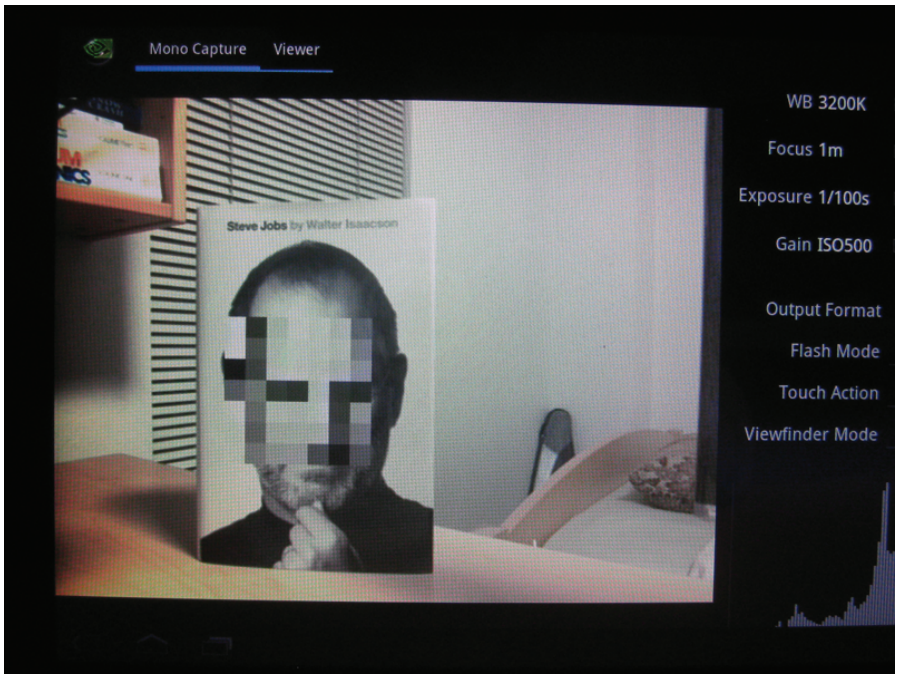


Figure 1: Demonstration of face detection. (a) The region of interest (ROI) corresponding to the acquired face is marked in the viewfinder with white borders. (b) For fun: a hackish implementation of face censoring.

2 Flash/No-flash fusion

2.1 Flash/No-flash fusion algorithm

I used a joint bilateral filter implemented in ImageStack with the no-flash image as the target, and the flash image as the reference. I requested the permutohedral implementation. An example of a flash/no-flash fusion result is shown in Fig. 2.

2.2 Determination of parameters

The relevant parameters are the Gaussian filter sizes σ_x and σ_y, as well as the "color filter" σ_c for the reference image. In addition, I had to tweak the sensor gain for the flash image (i.e. the gain reduction hack to avoid actual metering).

As usual I determined the parameters by experimentation. Firstly, I had to reduce the sensor gain for the flash image to 10% of the no-flash gain to avoid total saturation of the flash image. Even then, my flash scene came out rather exposed and flat, which led me to reduce σ_c accordingly.

Once I could tell that my chosen value of σ_c was having an edge-preserving effect in the filtered image, I began tweaking σ_x and σ_y in the vicinity of $1-5$ pixels. Over this range, I didn't feel that there was a clear optimum in the quality of the filtered image. I didn't enjoy having to wait some 20 seconds between each calculation; so I chose a final value of $\sigma_x = \sigma_y = 3$ pixels and moved onto the Zebra viewfinder task.

2.3 Assumptions of the fusion algorithm

The idea of bilateral filtering makes a certain number of assumptions. Firstly, we are assuming that the pixels in the flash and no-flash images are registered. That means that there should be minimal scene/camera motion between the two shots. In my case, I had placed the tablet on a flat surface – I did not attempt to take a flash/no-flash pair by holding the camera.

Ideally, the flash image would be an accurate and sharp capture of the edges in the no-flash image. Looking at the flash image in Fig. 2(a), it is clear that there are plenty of shortcomings. For instance, the columns of the White House model cast shadows on the building's facade that will be mis-interpreted as an edge. (The model's flag also casts a shadow on the wall.) Furthermore, even though I had greatly reduced the sensor gain for the flash shot, the model remains overexposed, and it becomes difficult to distinguish the front four columns of the White House entrance. This is a mechanism by which the edges present in the actual scene may be missed by the reference (flash) image.

3 Zebra viewfinder

3.1 Zebra shader program

Of all the tasks in this assignment, I enjoyed Zebra viewfinder the most. Mainly because the end result is "actually" real time, and the viewfinder is smooth as silk. The Zebra mode is demonstrated in Fig. 3, along with an additional "experimental mode" that I've played with.

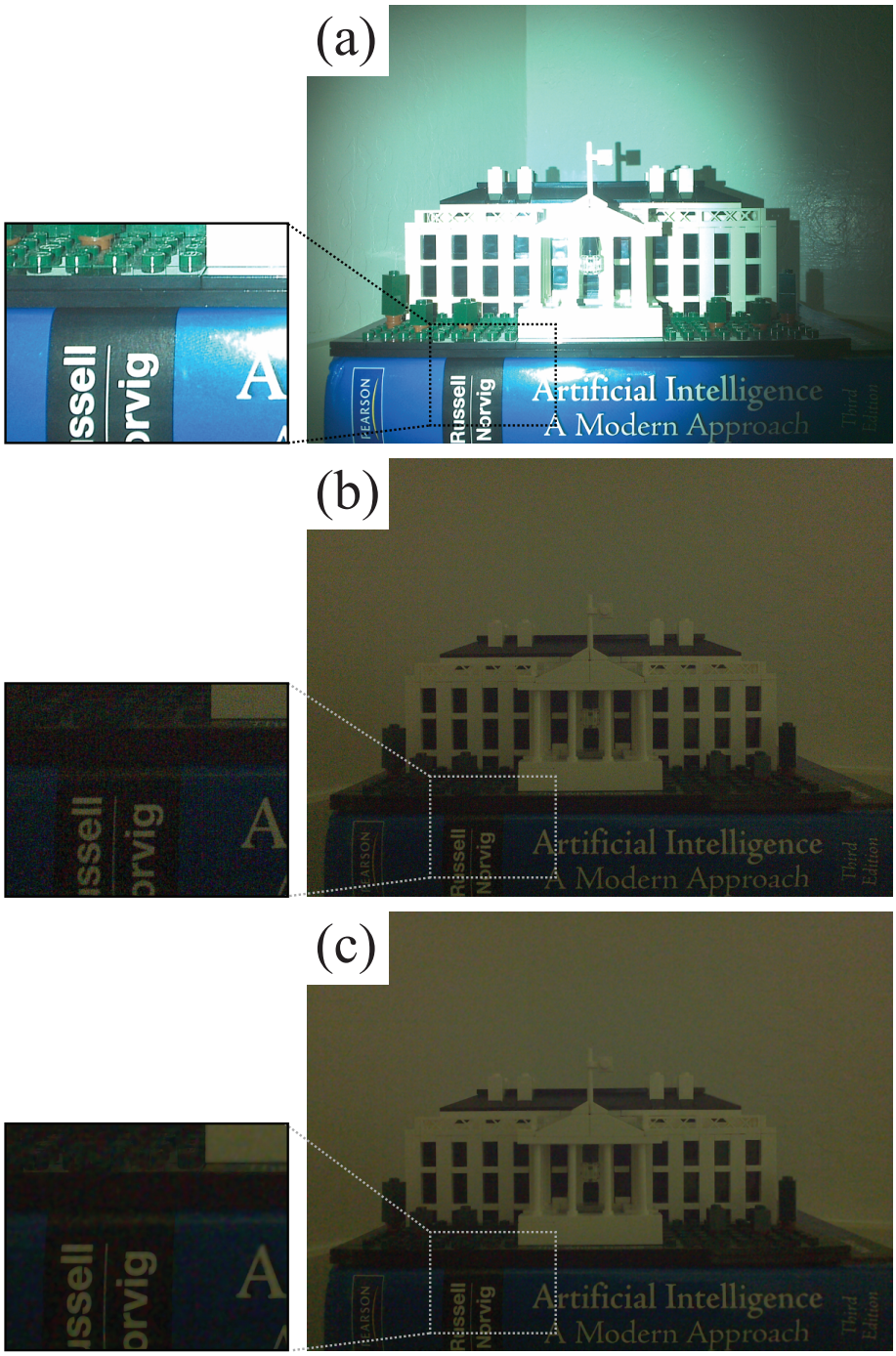


Figure 2: Flash/no-flash fusion using joint bilateral filtering. (a) The original flash image. (b) The original no-flash image. (c) Result of joint bilateral filtering, with $\sigma_x = \sigma_y = 3$ pixels and $\sigma_c = 0.25$.

Within the Zebra fragment shader, I check whether the pixel is overexposed (*i.e.* color values all greater than 0.95). If not, I simply pass on the original color. On the other hand, if the pixel is overexposed, I overwrite it to be either white or black corresponding to the zebra stripes. The assignment of the final stripe color – black or white – can easily be performed by taking the appropriate modulus of the Manhattan distance of the pixel to the image origin.

3.2 Other tasks for the GPU

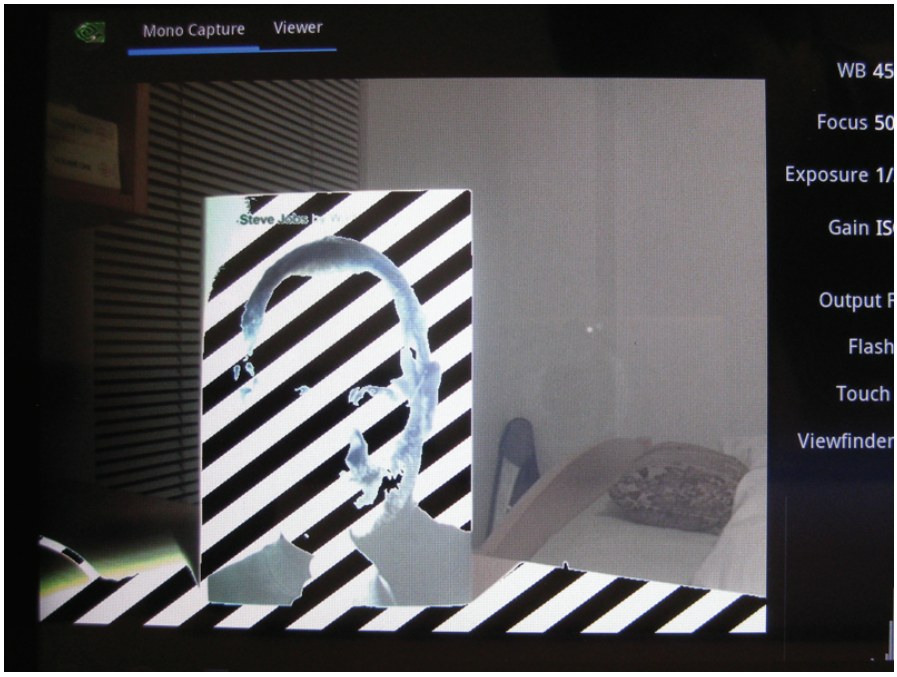
Following the zebra viewfinder, I continued to play with GLSL and found out that each fragment shader can request an arbitrary number of textures at arbitrary locations (within computational reason, of course). This implies that we can implement a variety of image filtering operations in GLSL. As a proof of principle, I wrote an "experimental" viewfinder mode that replicates the sharpness map calculation that I had used in `MyAutofocus`. The result is shown in Fig. 3(b). In contrast to the `MyAutofocus` implementation, the GLSL version does not skip any pixels and actually provides true real-time performance.

In this spirit, it would also be interesting to perform flash/no-flash filtering (or general joint bilateral filtering) on the viewfinder in real-time by going through GLSL. I very much like the end result of flash/no-flash fusion; I just don't want to wait 20-seconds to see how the shot turned out.

The significant compute capabilities of the GPU can be leveraged to calculate and display other filters that may help a photographer compose and capture his or her scene. For instance, an optical flow map could be computed whose properties can trigger a shot (perhaps when the subject is still, or a rapidly moving object enters the scene).

While working on `MyAutofocus` of the HelloCamera assignment, I found it interesting that one could in principle estimate a rough depth map by noting the position of the focusing lens that yields the highest sharpness score for a sub-region of the image. Computationally, the depth map calculation (via focusing) seems much more feasible on the GPU.

(a)



(b)

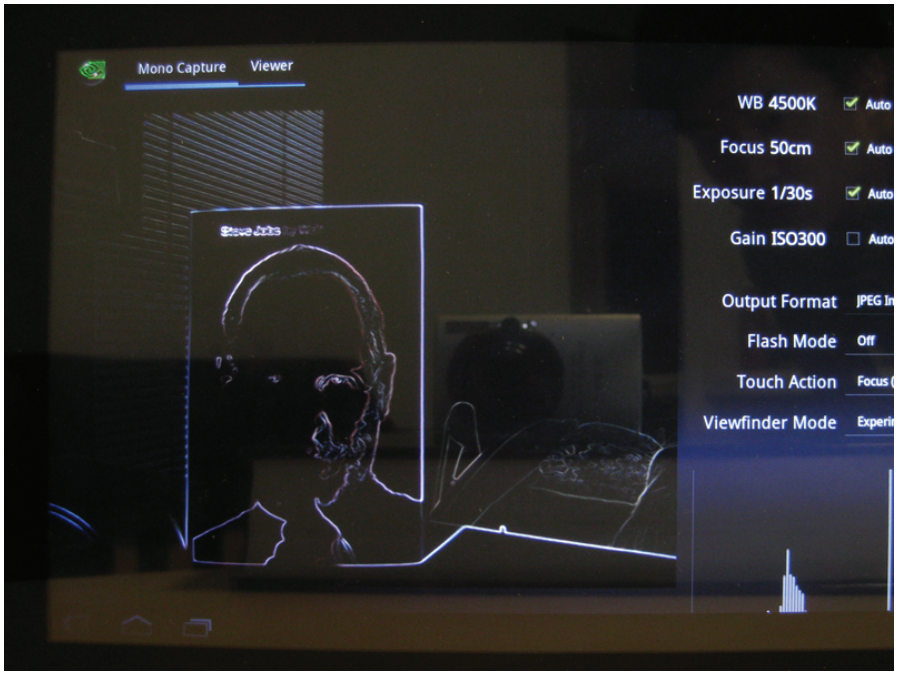


Figure 3: Alternate viewfinder modes implemented through GLSL. (a) Zebra mode identifies the pixels in the scene that are overexposed. (b) GLSL-implementation of up-down-left-right gradients for sharpness calculation. Such an edge-image could serve as the reference in joint bilateral filtering.