Revisiting Autofocus for Smartphone Cameras Supplemental Materials (API calls)

Abdullah Abuolaim, Abhijith Punnappurath, and Michael S. Brown

Department of Electrical Engineering and Computer Science
Lassonde School of Engineering, York University, Canada
{abuolaim, pabhijith, mbrown}@eecs.yorku.ca

To prepare the API for receiving user calls, the user needs only to run the API software which operates like a network server. The API will return the string "start" indicating that the API has been started and ready to receive user program calls.

The following API calls are used to set up the API environment. These calls all return `null` and cost 0 cycles:

API call	Description	Return values	Clock cycles
`setScene(int sc)`	Select one of the 10 scenes, `sc`$= 0, ..., 9$	`null`	0
`setRegion(int [] reg)`	Set the region either by selecting one of the predefined regions: Global (`reg`=[0]), 9 Focus Points (`reg`=[1]), 51 Focus Points (`reg`=[2]) or Face Region (`reg`=[3]), or by passing an array of size $r \times 4$ where r is the number of regions. Each region has offset (x,y), width, and height.	`null`	0
`setSharpMeasure(int sh)`	Select one of the two predefined sharpness measures: Sobel (`sh`=0) or Prewitt (`sh`=1).	`null`	0
`setKernelSize(int ker)`	Select one of the three predefined kernel sizes: 3 (`ker`=0), 5 (`ker`=1) or 7 (`ker`=2).	`null`	0
`recordScript()`	Start recording the subsequent API calls in a script.	`null`	0
`endScript()`	Stop recording the subsequent API calls in a script.	`null`	0
`callPD(int` ρ`)`	Compute phase difference and return approximate optimal lens position $p \pm \rho$.	$[C_{\text{loc}}, C_{\text{glob}}, I^j_{C_{\text{glob}}}, j, p]$	1
`callCD(function fun)`	Allow the user to pass custom contrast detection AF implementation as a function. Default Sobel/Prewitt with kernel size as set by user. `fun` is a function written in Python format.	$[C_{\text{loc}}, C_{\text{glob}}, I^j_{C_{\text{glob}}}, j, score]$	1 (if default) or defined by user
`moveLensForward()`	Move the lens a step forward.	$[C_{\text{loc}}, C_{\text{glob}}, I^j_{C_{\text{glob}}}, j]$	1
`moveLensBackward()`	Move the lens a step backward.	$[C_{\text{loc}}, C_{\text{glob}}, I^j_{C_{\text{glob}}}, j]$	1
`noOp()`	No operation. No lens movements. Used to increment C_{loc} in order to move in global time C_{glob}.	$[C_{\text{loc}}, C_{\text{glob}}, I^j_{C_{\text{glob}}}]$	1
`getFaceRegion()`	Detect face(s) and return face region(s) `int face[]` if exists. `face[]` is an array of size $m \times 4$ where m is the number of face regions. Each face region has offset (x,y), width, and height.	$[C_{\text{loc}}, C_{\text{glob}}, I^j_{C_{\text{glob}}}$, `face[]`$]$	0

Table 1: API calls with their parameters and return values. Note that each API call incurs a cost related to the number of internal clock cycles. C_{loc} is used to represent the current clock cycle, C_{glob} represents the current global time, $I^j_{C_{\text{glob}}}$ represents the current image at current C_{glob} and current lens position j. The term p represents the optimal lens position (when a phase-detection API is called), and *score* is the score of sharpness measured (using either a default based on Prewitt or Sobel gradient energy or from a function defined by user).

- `setScene(int sc)`: The user can pass the scene number (`sc`$= 0, ..., 9$) to select one of the ten scenes. We upload the entire temporal sequence at the beginning of this call as a single 4D array. This is loaded into the main memory to speed up image access. Uploading the scene takes from 7 to 15 sec based on scene size ($n \times 50$ images).
- `setRegion(int [] reg)` (see description in Table 1).
- `setSharpMeasure(int sh)`(see description in Table 1).
- `setKernelSize(int ker)`(see description in Table 1).

The `recordScript()` and `endScript()` API calls are used to record other API calls and associated metadata to an output script. This output script can be loaded later for user playback purposes. Both calls return `null` and cost 0 cycles. `recordScript()` creates a output script named by the user and writes user info with all API environment parameters as comments. Then, it starts recording all subsequent API calls. `endScript()` writes a summary and metadata about the performance – for example, total number of lens movements, lens position, API call made at each clock cycle, etc – as comments, too. Additionally, It stops recording API calls and closes the script.

`callPD(int` ρ`)` and `callCD(function fun)` are used to mimic the implementation of phase difference (PD) and contrast detection (CD) AF. The former was discussed in details in the main paper, whereas the latter will be discussed here. The `callCD(function fun)` is added to allow future algorithm development, scalability and give more flexibility to user implementation. By allowing this feature, our API CD AF can be easily extended. The user only needs to develop his own CD implementation, define his own parameters and return values. The CD implementation, thus, can be passed as a function `fun` written in Python. Then, the API is responsible to implement user-custom CD implementation.

The `callCD(function fun)` will take default values if the user passes nothing and return the gradient magnitude energy score *score* of the ROI(s) based on the default objective. For example, if the objective is 9 focus points, *score* will be an array of size 9 where each index represents the gradient energy of a specific region. Both `callPD(int` ρ`)` and `callCD(function fun)` cost 1 clock cycle with default implementation. However, `callCD(function fun)` may cost more if it is computationally expensive and this can be defined by the user.

Regarding `noOp()` API call. This call does nothing except that it advances the 1 clock cycle to increment the internal clock C_{loc}. This is an important call used to fill clock cycles C_{loc} (increment till 10 cycles) in order to move in global clock C_{glob}. For example, suppose that the lens position is optimal at current C_{glob} and the user does not want to move lens but still he needs to access next time point (C_{glob} +1). The only thing possible is to keep calling `noOp()` to increment C_{loc} in order to move time (C_{glob} will be automatically incremented every 10 clock cycles).