ArtCeleration
Project Experience

SAURABH JAGDHANE
ASU ID: 1209572595
(sjagdhan@asu.edu)

SANDESH SHETTY
ASU ID: 1209395990
(ssshett3@asu.edu)

Description:

1. Design a library framework and a service which can be exported to application developers for an Image editing application.
2. The application developer will build apps that can use this library framework to request multiple Image transforms.
3. The library is designed to provide following 5 image transform features:
   {"Gaussian Blur", "Unsharp Mask", "Color Filter", "Sobel Edge Filter", "Motion Blur"}
4. The user can request for multiple image transforms by selecting an option from drop-down menu but the the processed image transform will be delivered in the order of request by the user.

Goals:

1. To design an efficient library framework and service which handles the user's request for any of the image transforms as mentioned in description above.
2. The library will be able to handle multiple requests without waiting for the previous image transform results.
3. The library framework will store user's requests using unique number generated within a library and library will queue all the requests in a FIFO (First in First out) manner
4. Library will offload all the image processing to the service. Service should act as a multithreaded server and can start multiple threads which will include the image processing logic.
5. Threads will inform the library once the processing is done and Shared Memory resources will be used to obtain results.
6. Threads will maintain the unique number and once done processing the respective thread data will be obtained from a queue. Library will check if the thread done processing is at the front of the queue, then only results will be displayed else it will keep waiting adhering to the FIFO mechanism.
7. Queue has to be maintained within a library acting as a FIFO and thread-safe (ConcurrentLinkedQueue).
8. The library service transactions act like a client-server architecture. To achieve modularity service is performed as an isolated process. The library service will communicate over a binder and shared memory.
9. To perform image transform quicker the target is to offload the complete image transform algorithm in C++ code. This can be achieved just by following the same logic as of Java implementation of algorithm and JNI call to native code.
10. To make native implementation run faster ARM NEON (SIMD) instructions to be used so that independent image processing tasks can be run simultaneously.
11. Efficient memory management for each transform logic and optimal use of variables, consideration of garbage values of bitmap image.
12. To perform argument validation for each and every image transform requested by the user without starting any threads. This is to be achieved in library (Artlib.java).

Design:

Application components:-

1. **Artceleration Library (ArtLib.java)**

This is the library which implements all the methods required by the application: getTransformsArray, requestTransform, registerHandler and getTestsArray. The library acts a client and requests the service which acts as a server to process the transform requests made by the library. The library creates a new MemoryFile (equivalent of AshMem in Java) every time the requestTransform method is invoked by the application. The requestTransform method in the library writes the Bitmap image to the MemFile and shares the file descriptor for this MemFile with the Service. The file descriptor is made Parcelable using getParcelableFileDescriptor method of MemoryFileUtil.java which is already provided to us. The library binds to the service and receives the instance of the Messenger object created in the service and we send the ParcelableFileDescriptor, type of transform requested, request number and size of the byte array corresponding to the Bitmap input. A global Messenger object is also created which points to a Handler class in library. Also, this Messenger object is set to the replyTo of the Message object sent to the Messenger of the service which the service uses to reply or communicate with the library. Now, when the library's handler receives a message from Service it reads the output bitmap using the file descriptor of the memory file shared by the service and starts a thread in which it continuously checks whether the request number of the head of the queue matches with the request number of the current one because only then we remove the head of the queue and invoke the onTransformProvessed method using the TransformHandler object registered with library by the application.

2. **Artceleration Service (ArtTransformService.java)**

The Artceleration Service does the work of processing each transform request by acting as a Server and creates a new thread for each new transform request it gets in its handler by the library which acts as a client. The service reads the MemoryFile for the input Bitmap using the file descriptor shared by the library. There are five classes associated with service which are threads corresponding to each of the transform type. The service passes the Messenger object of the library, input bitmap image object, request Number to map exactly to the request number in the queue when the output bitmap image is passed to the library and the memory file object which is created by the service for every new request.

Interface between application components:

For communication between the library and the bounded service, Messenger instance of Service is returned in the onBind method of the Service. This Messenger object points to a handler class in service which has an overridden handleMessage to handle the requests made by the library. In this request message, the replyTo is set by the library to

the Messenger instance created in it pointing to its own handler class which the service uses to reply back to the client (library) so that the library can read the output bitmap image. To write and read the Bitmap images we use a MemoryFile whose file descriptor is created and shared in both the library and service. For every transform request or transaction between the library and service a new MemoryFile is created.

Interface between transform threads and Native C++ code:

The thread using JNI calls to process transform include the shared library .so file created after the build process such that it invokes call to the native method implemented in C++**.** This method is passed the input and output bitmap and all the arguments required. This method does not return anything as it directly operates on the references of input and output pixels. So the changes are reflected on the output bitmap directly after input pixels are read and image processing is done on it.

Strategy:

Most of the time was invested in understanding MemoryFile and Binder transaction over remote processes. Along with this FIFO, Multithreading, Messenger IPC needs to be understood. So, Saurabh started off with creating MemoryFile and writing byte array of the image data into the MemoryFile. Also File Descriptor was passed using Messenger IPC and established a 2-way communication channel. Simultaneously Sandesh started off doing some research on thread-safe queue to be maintained with unique request numbers. Initially, Saurabh designed a service to be responsive to library but Sandesh introduced a multithreading for every transform request within a service for every image transform request to be handled in a separate thread.

Checkpoint-1 included the robust implementation of FIFO queue. Hence, we need not focus on it for the final submission instead our goal was to straightaway start off with the implementation of the image transforms algorithms. We decided to finish off Java implementation as soon as possible so that most of the learning and implementation time can be dedicated to NDK and NEON SIMD which worked out quite well as we end up performing all image transforms in native code.

No.	Task	Assignee	Deadline
1.	Library-service setup for a primitive data type	Saurabh	10-27-2016
2.	Writing raw image to Memory File	Saurabh	10-28-2016
3.	Thread safe FIFO queue	Sandesh	10-27-2016
4.	Two way Messenger IPC	Sandesh	10-31-2016
5.	Multithreading for every request to the service	Sandesh	11-03-2016

6.	Creation of classes for multiple image transforms	Saurabh	11-04-2016
7.	Handler in a library for response by multiple threads	Saurabh, Sandesh	11-06-2016
8.	Understanding image processing basics and algorithm logic	Saurabh, Sandesh	11-08-2016
9.	Gaussian blur, Unsharp mask (Java)	Sandesh	11-11-2016
10.	Color filter, Sobel edge filter (Java)	Saurabh	11-13-2016
11.	Motion blur (Java)	Sandesh	11-14-2016
12.	Basic NDK setup	Saurabh	11-15-2016
13.	Motion blur (NDK)	Saurabh	11-18-2016
14.	Sobel edge filter (NDK)	Sandesh	11-19-2016
15.	Study ARM NEON instruction set and sample implementation	Saurabh, Sandesh	11-20-2016
16.	Sobel edge filter (ARM NEON - grayscale image)	Sandesh	11-22-2016
17.	Gaussian blur, Unsharp mask (NDK)	Saurabh	11-24-2016
18.	Color filter (NDK)	Sandesh	11-26-2016
19.	Memory management and timing characterization	Saurabh, Sandesh	11-28-2016
20.	Argument validation	Saurabh, Sandesh	12-01-2016
21.	Documentation and report	Saurabh, Sandesh	12-02-2016

Challenges:

1. Two-way communication between Library and Service.

Solution: Using Messenger objects pointing to a handler in both the library and service. The service's messenger object instance is returned to library when the library binds to the service (server). The library (client) sends its Messenger object instance each time it requests a transform request from the service in the Message object. This is used by the service to reply back to the library. MemoryFile is used to share data between the two isolated processes.

2. Handling multiple requests for image transforms:

Solution: The service acts as a multithreaded server which creates a new thread for every transform request made by the library.

3. FIFO queue to maintain the order of requests

Solution: We have used ConcurrentLinkedQueue data structure as it is thread-safe which reduces the overhead of having to specifically synchronize thread operations on this global data structure. Also, we have request numbers to assign numbers to each request which is a static variable in our library class so that it can be mapped with the request number in queue when the processed image is returned to the library from service.

4. Resolving segmentation faults for native C++ code execution

Solution: Every value of a 2D array should be initialized to some value before updating otherwise it will operate on the garbage value. Also, managing memory as in some transforms we opted to allocated some of the arrays using heap memory to avoid segmentation faults due to the way memory is accessed on stack. Heap memory access increases the processing time a little bit but makes the implementation a lot easier. So, there is a trade-off.

5. To understand the traversing of input and output pixels in Native code using info->stride and setting the output pixels after processing.

6. Linking the invocation from Java to native method implemented using C++.

7. Writing the Android.mk (Makefile) and understanding its interpretaion during the build process of the application.

Improvements/ Security Flaws:

1. We will improve the processing time for multiple transform request by using multithreading.
2. Improving the multiple threads handling for performing image transforms in parallel.
3. Reducing the busy waiting for the threads before popping out of the queue.
4. Validation of intArgs and floatArgs arguments done in ArtLib.java for each type of transform requested to prevent undesired behaviour.
5. A button can be added on the application screen which allows us to view and select our gallery images or take picture from camera, which can be edited using the application at runtime to view the effects.
6. Executed all the transforms using Native NDK instead of Java which resulted in significant time improvements as shown in the below table:

Time improvements:

Sr. No.	Image transform	Java (secs)	C++ (secs)
1.	Gaussian blur	25	1.27
2.	Unsharp mask	40	3.24
3.	Color filter	13	0.045
4.	Sobel edge filter	27	0.170
5.	Motion blur	19	0.253

Conclusion:

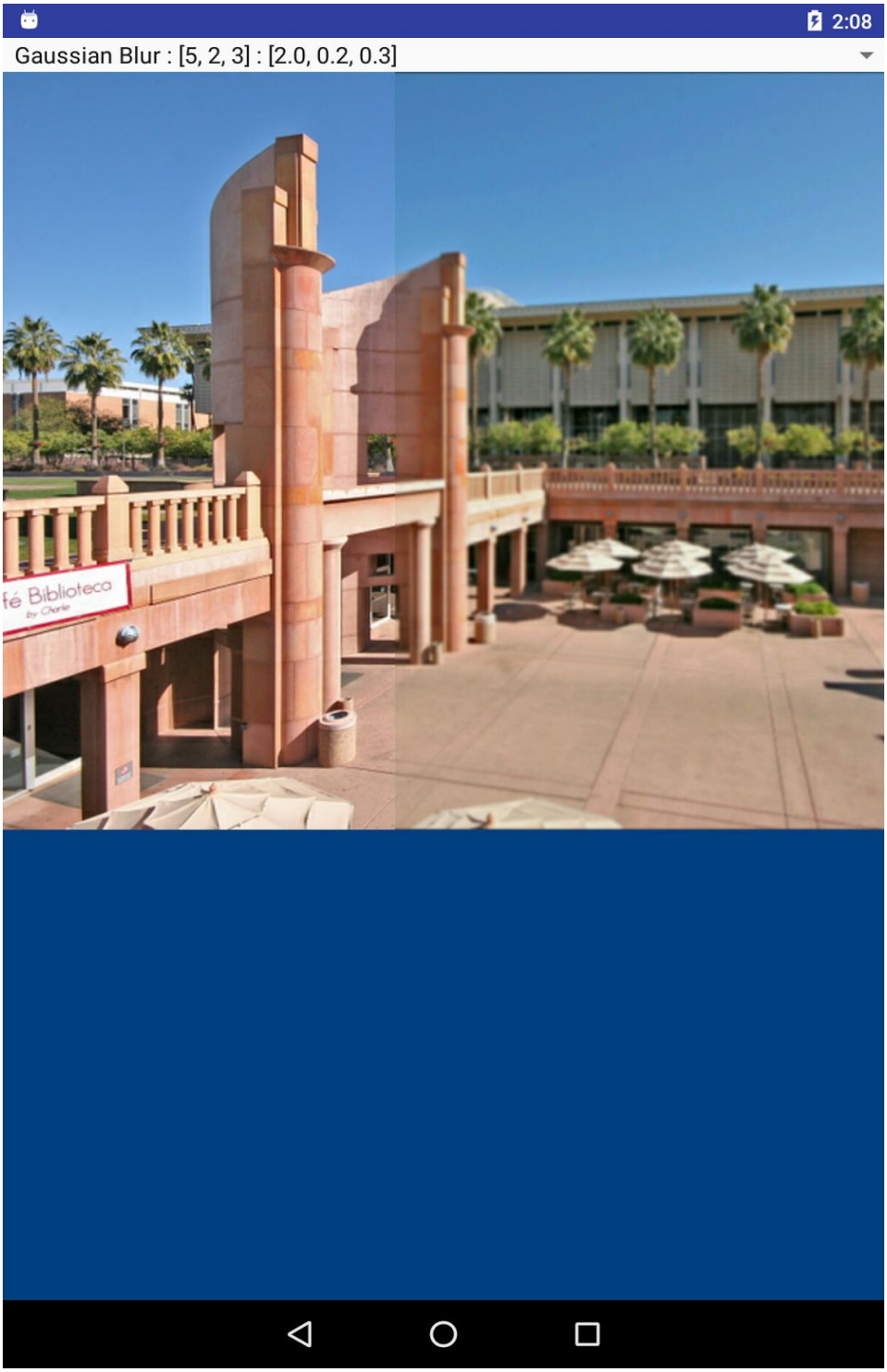
1. We have tried to implement NEON for Sobel Edge Filter where we have to create a grayscale image as the first step before moving ahead with keeping only x-gradient, y-gradient and overall. We were able to get a little distorted form of grayscale image using ARM Neon instructions as our image format was restricted to ARGB_8888. This limited our capacity to exploit arm neon instructions completely as only R, G, B values are to be updated for a set of pixels in a register but the alpha channel value has to be retained as it is in the input bitmap.

2. ARM NEON results mismatching (commented code included in Sobel Edge Transform cpp file), argument validation done, FIFO queue done and tested, all 5 image transforms are successfully implemented in native (C++) as well as Java.

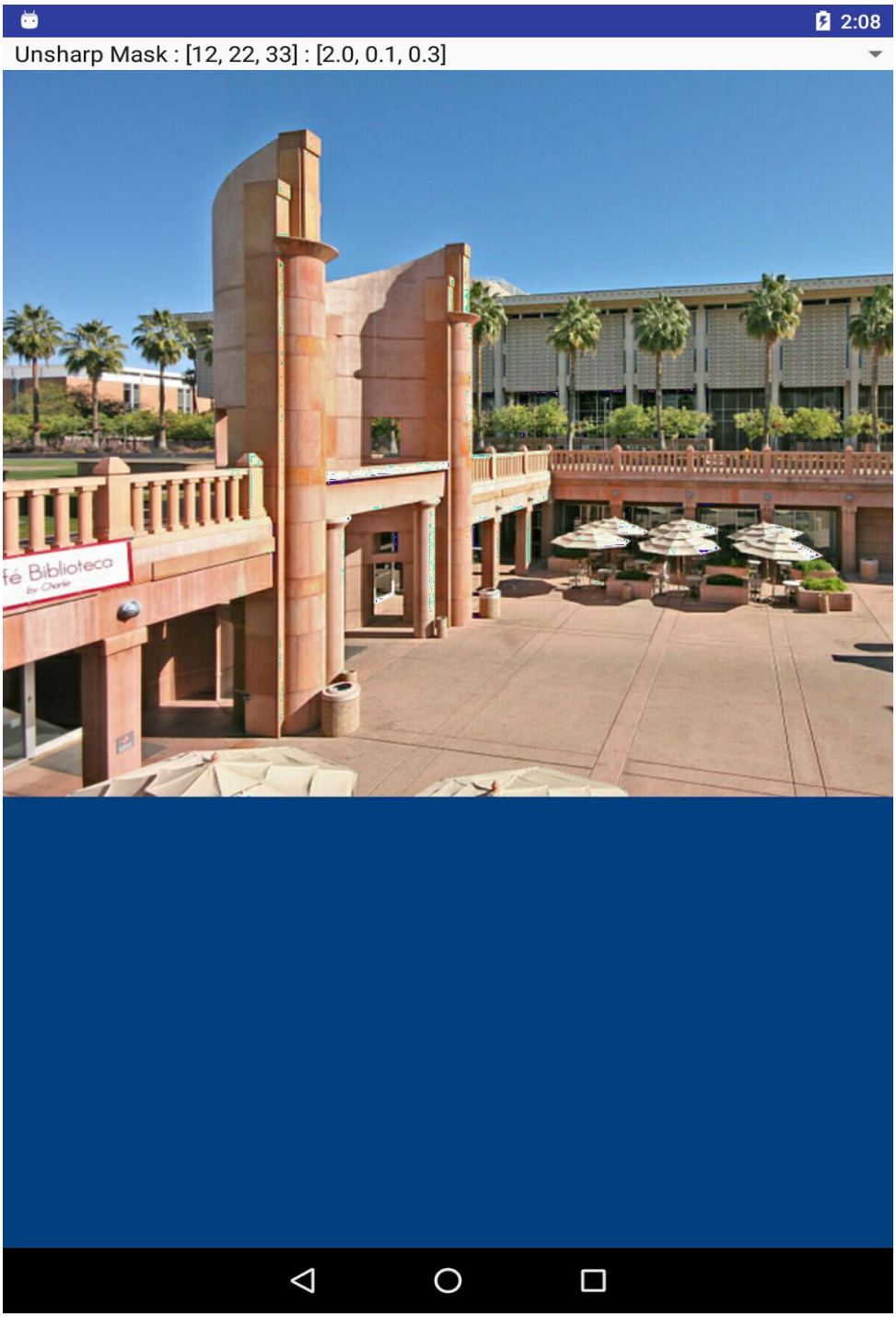
3. We got to work on and learn new interesting things like implementing native code in Android environment using Android NDK, implement image transform algorithms using pseudo codes which were previously known to us only theoretically, MemoryFile for large data transfer and Messengers for RPCs. Also, maintaining consistency with respect to the order of transform requested using FIFO (First In First Out) queue and understanding the library and service framework.

Screenshots:

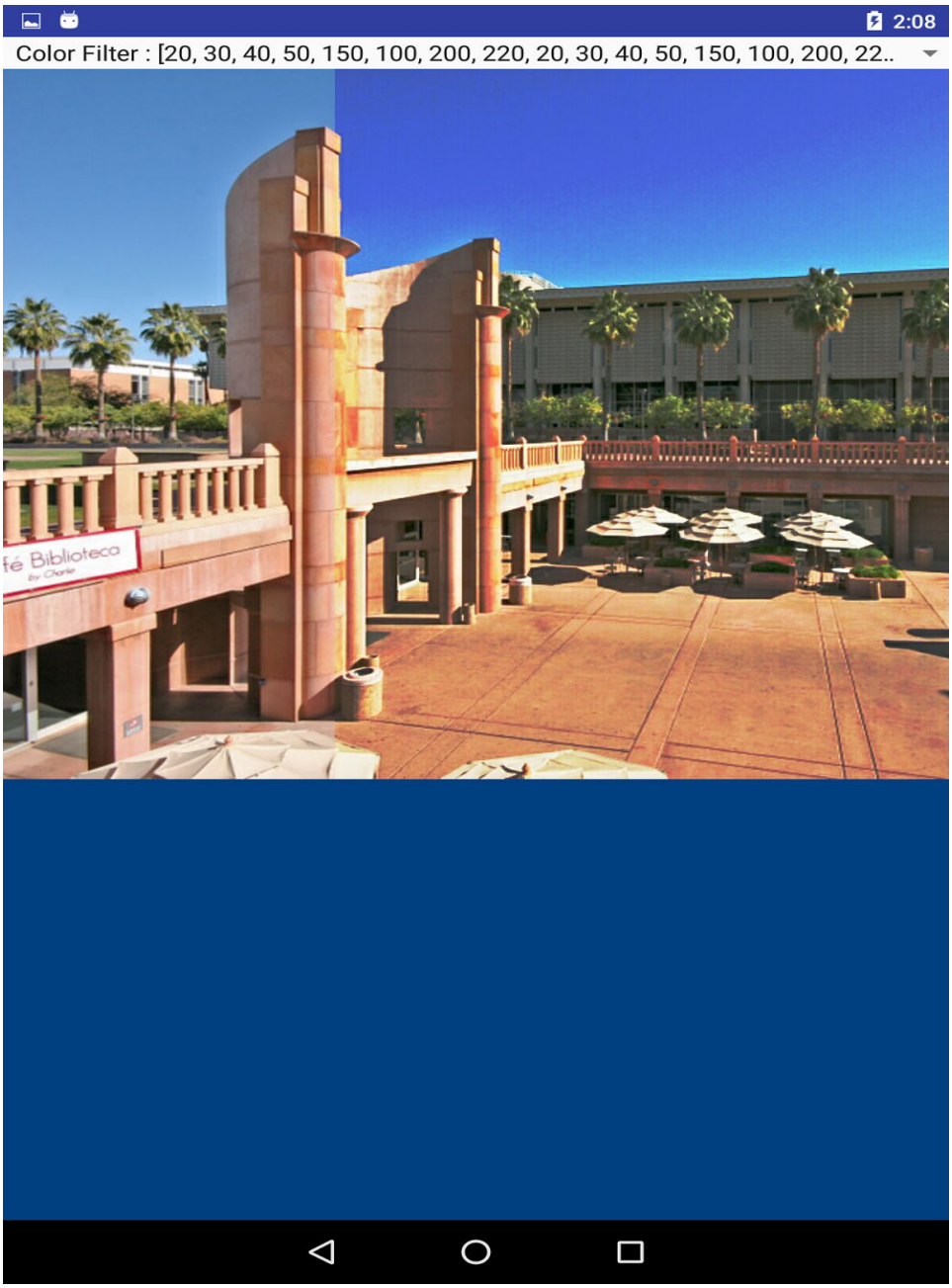
1. Gaussian Blur



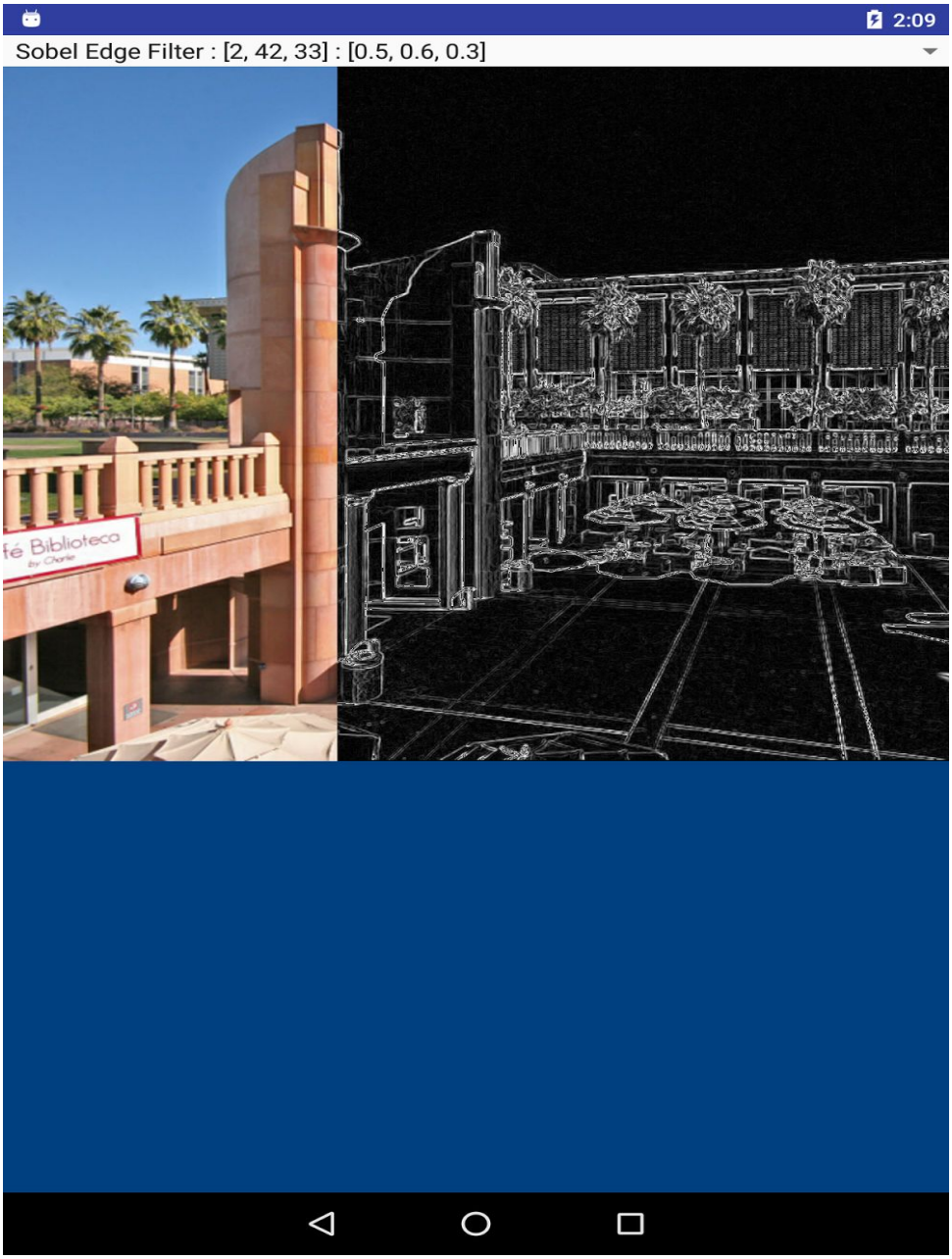
2. Unsharp mask



3. Color filter



3. Sobel Edge filter



5. Motion blur