# EEE 598 Fall 2016 Assignment-2
# Final Report
# ArtCeleration -
# An Artistic Transformation Library

**Project Members:**
**Omkar Salvi (1209536104)**
**Abhishek Patil (1209417518)**

## Goals

**High Level Objectives:**

- Getting hands on experience of using NDK, NEON in Android application
- Understand and implement library framework and threads
- Implement and communicate with service
- Communication between application, library and service
- Implement different image transform algorithms
- Understand and design non-blocking FIFO queues
- Getting hands on experience of android application development

**Assignment Description:**

ArtCeleration is library framework which is used for converting image into various artistic transforms. The actual transforms will be implemented in a service. The application will send image to library. Library will send the image to service for processing. Library should support multiple requests in non-blocking fashion. That means library should not block the following requests till 1st request is being served completely. Library should allow multiple requests to service. Service may execute these requests in any fashion and send them to library. But library should send the transforms of images back to application in order. I.e. in the same order as the requests were made.

## Design

**Application Components:**

1) **Library :**

   a) **Strategy for queueing requests/responses:**
   We are assigning a unique request id to each request made by application. To implement non-blocking functionality for requests, we have implemented a Queue which will store the request id of each request made by application to the library. When a request is sent to service for processing its request id is pushed into the queue. After service returns the image when processing is finished, we are checking whether the request id associated with this image is at the head of the queue. If it is not at head of queue then it will wait for its turn. When head of the queue is same as waiting processed image's request id, It will be sent back to application. Also element at head of the queue will be popped out. This way we are implementing non-blocking FIFO queue to service multiple requests to application.

   b) **Strategy for accepting requests:**
   In requestTransform function, we are converting bitmap into byte array. This byte array is written to a memory file. Then by using getParcelFileDescriptor method of

MemoryFileUtil we created a parcelable file descriptor for this memory file which has the byte array. Then we created a bundle and put the parcelable file descriptor inside the bundle. Finally we created a Message and put the bundle, bytearray size and request id in it. This message is sent to the service.

c) **Strategy for responding to responses from service:**
The handler class 'ArtLibIncomingHandler' is created inside library, which will handle the result obtained from service. In it's handleMessage method we are processing the message obtained from service. But to provide support for multiple concurrent responses we are implementing the threads. So for each response a thread will be created. Inside this thread we are processing the data obtained from service. First it will wait for its turn as per request id. Then we extract the bundle and get file descriptor from it. From this we read the byte array, convert it into a bitmap and send this bitmap to application via method 'onTransformProcessed' of the 'TransformHandler'.

d) **Input validations:**
The input arguments for different transforms are different. Null parameters are handled by putting null checks for integer and float arguments in each switch case. In addition to these, only valid arguments such as positive integers or restricted values are predefined in the application. These checks are essential so that user should not face any error in during the use of application.

2) **Service:** Image transforms are performed in an isolated service process. Inside the service to service multiple requests, we are creating a thread for each request. Inside this thread image transform algorithm will be implemented. At this stage we have implemented only 1 class for image transform i.e. 'GaussianBlurTransform.java' . So all the requests to transforms will result in threads executing this class. We are using switch-case structure to select the requested transform functionality. Inside service we created a method onBind which will return the communication channel from the service.

3) **GaussianBlurTransform:**
Gaussian Blur transform is nothing but blurring the given image using the Gaussian functions. For this transform we are first reading the bitmap from the file which we sent to service , into the byte array. Then creating a bitmap from this byte array. Then we are calling the method "doGaussianBlur" which takes bitmap, radius and sigma values as input arguments.The parameters used in gaussian blur are as follows:
**Standard Deviation(f0)** - This defines intensity of blur. A larger number is a higher amount of blur.
**Radius(a0) -** Represents the "radius" r of neighboring pixels being averaged.

The function will first calculate the Gaussian weight vector. To make this faster we are leveraging the symmetry property of this vector. Thus we are only calculating the 1st half of the values in vector and copying them into remaining half of the vector. This makes the computation of G(k) faster. Then we are extracting the different color values of each pixel and storing them into different matrices. So we will get 4 such matrices for alpha, red, green and blue channel. Then we calculated the q and P matrices respectively for red, green and blue input matrices. Then combined the values of alpha, red, green, blue to form a single pixel value for each of the location in input bitmap. We stored each of these calculated values into different output bitmap at respective position. The computation time of gaussian blur transform is reduced by calculating it as two independent transforms. Generation of gaussian weight vector G(k), where k belongs to [-r,r] is done as specified in assignment description:

$$G(k) = \frac{1}{\sqrt{2\pi\sigma^2}} exp(-\frac{k^2}{2\sigma^2})$$

Then $q(x,y) = G(-r)*p(x-r,y), + \ldots + G(0)*p(x,y),+ \ldots + G(r)*p(x+r,y)$ and

$$P(x,y) = G(-r)*q(x,y-r), + \ldots + G(0)*q(x,y),+ \ldots + G(r)*q(x,y+r)$$

Finally, converted output bitmap into byte array, store this array into file, created parcelable file descriptor and put it in bundle. This bundle is sent to library via message.

4) **UnsharpMaskTransform:**

For generation of Unsharp Mask transform we need to do the Gaussian Blur transform onto the input bitmap. Thus, we create an object of GaussianBlurTransform class and send it to constructor of UnsharpMaskTransform class. In this class we are saving this object as a member of the class. In constructor we are assigning this object to member. Then using this object we call the “doGaussianBlur” method of GaussianBlurTransform class. This method returns the bitmap which is gaussian blur transform of input bitmap. This transform is obtained using below parameters:

**Sigma (f0) - Floating argument passed to function.**

**Radius - 6 * f0**

Then for each pixel in input, we are calculating the difference between input and gaussian blur transformed bitmap pixel for all color channels. Then scaled it by second float argument. Finally to generate the output bitmap, we added the values for each color channel with its corresponding original values, combined them together to form one value representing the pixel. Then these values are stored into output bitmap at respective positions. Computation of parameters is done as specified in the project description.

**s = (p-q) * f1**

**P = s + q**

5) **<u>ColorFilterTransform:</u>**

We used NDK for implementing this image transform. We have written a native code in C++ for this transform. We defined a static native function “colorfilterndk” inside our native library. It takes bitmap and int array as the arguments. We have declared this method in a class “NativeClass”. We call this function by class. To use this function which is inside our native library, we are loading the library into our code in the static section at the beginning of the class ColorFilterTransform. This function stores the bitmap information into object of AndroidBitmapInfo. We check if the bitmap is in ARGB_8888 format. Then we use a method “AndroidBitmap_lockPixels” which will attempt to lock the pixel address. Locking will ensure that the memory for the pixels will not move until the “AndroidBitmap_unlockPixels” call, and ensure that, if the pixels had been previously purged, they will have been restored. After this, we extract the int argument array into array of type jint. We then call “transformcolors” method. This method will extract the color channels from each pixel in input bitmap. Then it will send these values to another method called “convertRange”. This method will map the input value of the color to the desired output value with the help of int argument array. We are basically formulating the equations of line between a pair of points from the int argument array. Then computing the output value by giving color channel value as input to this equation. It returns the final value for that color channel.

Finally the values for all color channels are combined together to form a single value of the pixel and this value is stored in the original bitmap at respective location. After this is finished, we call the “AndroidBitmap_unlockPixels” method. Finally we return the original bitmap object which will have the transformed values for all the pixels.

6) **<u>SobelEdgeFilterTransform:</u>**

The sobel filter is derivative mask used for edge detection. It is used to detect two types of edges in image. The operator uses two 3*3 matrices. These matrices are convoluted with the gray image obtained from original image. The gray image is obtained from the original image by multiplying the constant across each channel.

The grayscale brightness image is obtained by using below formula:

**q = 0.2989*red+ 0.5870*green + 0.1140*blue**

In the Sobel Edge Filter, the obtained gray image is convoluted using filters which are specified below:

```
SobelX ={{-1, 0, 1},                    SobelY ={{-1, -2, -1},
         {-2, 0, 2},                             {0, 0, 0},
         {-1, 0, 1}};                            {1, 2, 1}};
```

When the parameter passed from the library is 0, then the gray image is convoluted with only SobelX elements. It will find edges in the horizontal direction and this is because zeros column is in horizontal direction. When convolution with the SobelX mask is done then it would show only horizontal edges in the image. It calculates difference among the pixel intensities of particular edge.  The centre column of SobelX mask is consists of zeros, hence it does not inlcude the original values in the image. Instead of that, it calculates the difference of above and below pixel intensities of specific edge.

When the parameter passed from the library is 1, then the gray image is convoluted with only SobelY elements. It will find edges in the vertical direction and this is because of centre column is zero. When convolution with the SobelY mask is done then it would show only verticaledges in the image. As the centre column is zero, it will not include the original values of image but it calculates the difference of right and left pixel values around that edge.

When the parameter passed is 2, then the transformed image contains edges in both vertical and horizontal direction. This is implemented using convolving gray image with both X and Y sobel operator matrices.

7) **MotionBlurTransform:**

In the Motion Blur Transform, the transformed image is the arithmetic mean i.e horizontal or vertical mean of nearby input pixel values of the input image. The motion blur filter creates a blur movement. This filter takes in two parameters which are:

**a0-Direction of blur(it can be horizontal or vertical)**

**a1- Positive integer representing the radius of nearby pixels being averaged.**

The a0 can take 2 values and it defines the direction of blur.

**a0 = 0 For the horizontal blur motion**

**a1 = 1 For the vertical blur motion**

The averaged pixel value is valid if it lies in the coordinate range (0 to N-1) and (0 to M-1). The pixels outside of this range are assumed to be zero. This transform is implemented in native C++ library. The class MotionBlurTransform will extract the arguments, load the native library and call the native function “motionblurndk”, which in turn returns the transformed image.

8) **Native Library :** The name of the native library created is “MyLibs”. The associated file is “libsMyLibs.so”. It is located inside directory path - “Artceleration-EEE598-Assn2-AbhiOmi\artcelerationlibrary\src\main\jniLibs\armeabi-v7a”.

9) **edu_asu_msrs_artcelerationlibrary_NativeClass.h:** This is header file for native C++ library. It contains function declarations for all the native functions defined in native library.

10) **edu_asu_msrs_artcelerationlibrary_NativeClass.cpp:** This is C++ code which contains the implementations of all the native functions and its helper functions. There are following important functions defined in this file for implementing the color filter and motion blur transform using NDK:

    a) ***Java_edu_asu_msrs_artcelerationlibrary_NativeClass_colorfilterndk* :** In this function we are implementing the color filter transform using C++. We are extracting the color channels from each of the pixels in the bitmap. Then for each color channel value, we are mapping it to the corresponding output color value from the mapping provided via int array. Basically we are finding the equation of each line between 2 adjoining points in integer array. Then using color value of pixel as input, we calculate the output value of the color. Finally, we combine all the color channels to form final pixel value. Then we assign this value to the pixel in original bitmap.

b) ***Java_edu_asu_msrs_artcelerationlibrary_NativeClass_motionblurndk***: In this function we are implementing the motion blur transform using C++. According to the transform requested is horizontal or vertical blur, we call corresponding helper function. We are extracting the color channels from each of the pixels in the bitmap. Then we are storing these color channel values in 4 integer arrays corresponding to alpha, red, green and blue channel in pixel. Then iterate through these arrays and taking the average of pixels within radius distance from input pixel, in rows for horizontal blur and in columns for vertical blur. Finally we store these average values back into original pixel and return the bitmap.

11) **NativeClass :** This is a java class which contains the declaration of all native methods. All native methods are declared as static so that they can be called in service using the class name. Using this class name, we are calling the native methods inside our android code.

12) **CheckPermission helper in Library :** For API level 23 +, Google has introduced new set of permissions which will ask users to grant permissions to apps while the app is running. With this, user can basically control the applications functionality. These permissions are divided into two categories i.e. normal and dangerous. Normal permissions do not risks user's information and if these permissions are listed in the manifest file then device/system will grant permission automatically. Dangerous permissions can risks user's privacy and these permissions can't be granted without user's invocation.

To take care of this permission check, checkPermission is defined which allow user to have more control over storage permissions. If the API level is greater than application will prompt user to grant permission. If user allows then the media file i.e. processed bitmap is stored in the device storage directory.

**Interfaces between Application Components :**

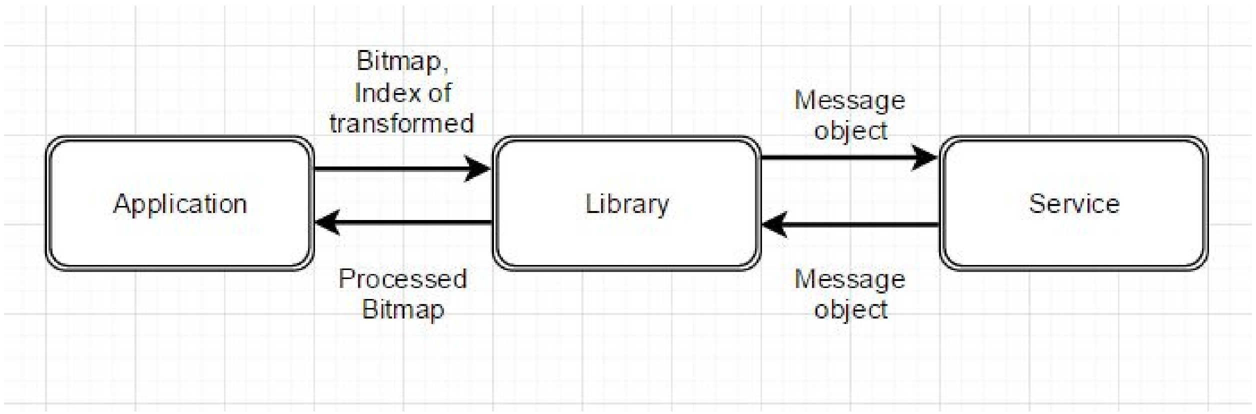
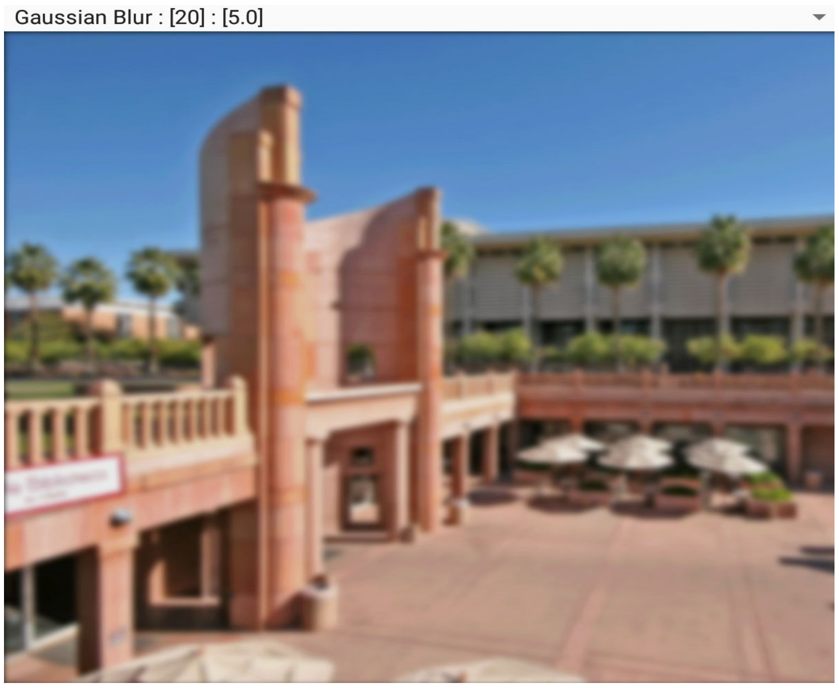


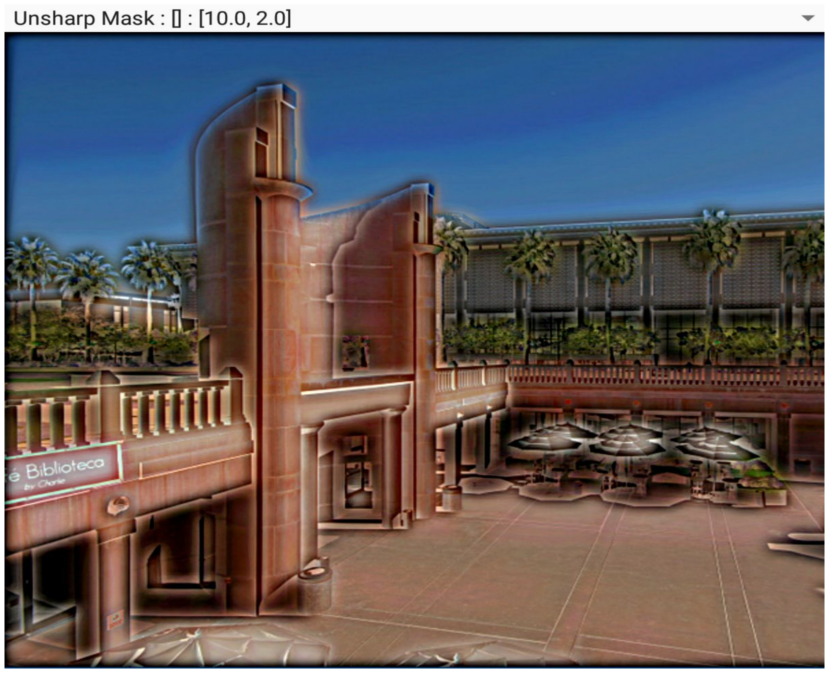
Figure : Block Diagram of complete application designed and Interaction between application components

## Output screenshots:

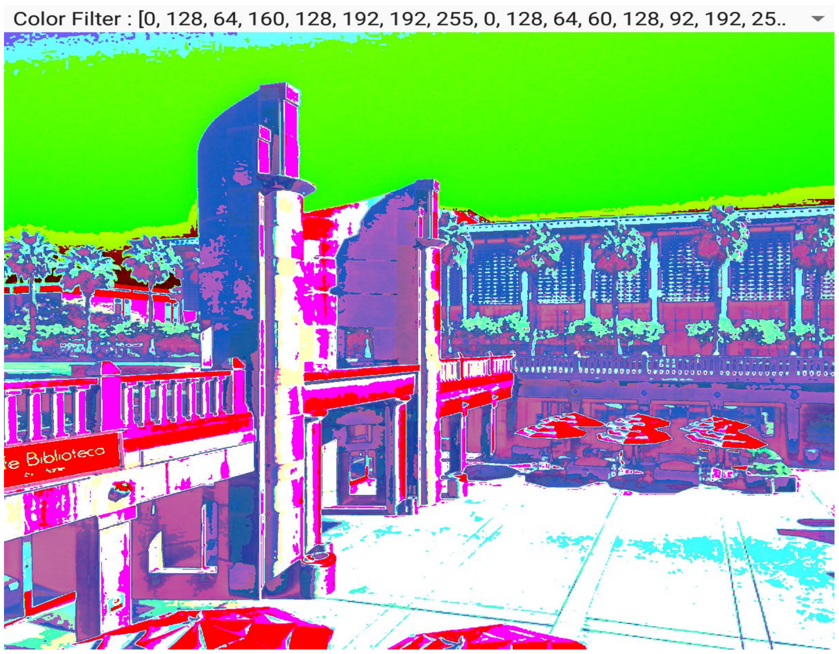
**Gaussian Blur Transform:**



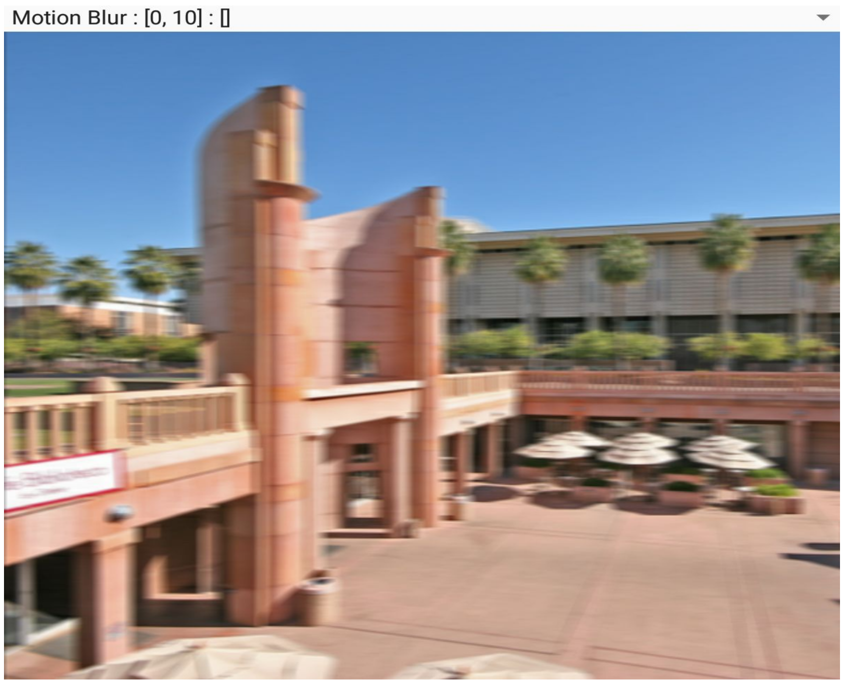
## Unsharp Mask Transform:

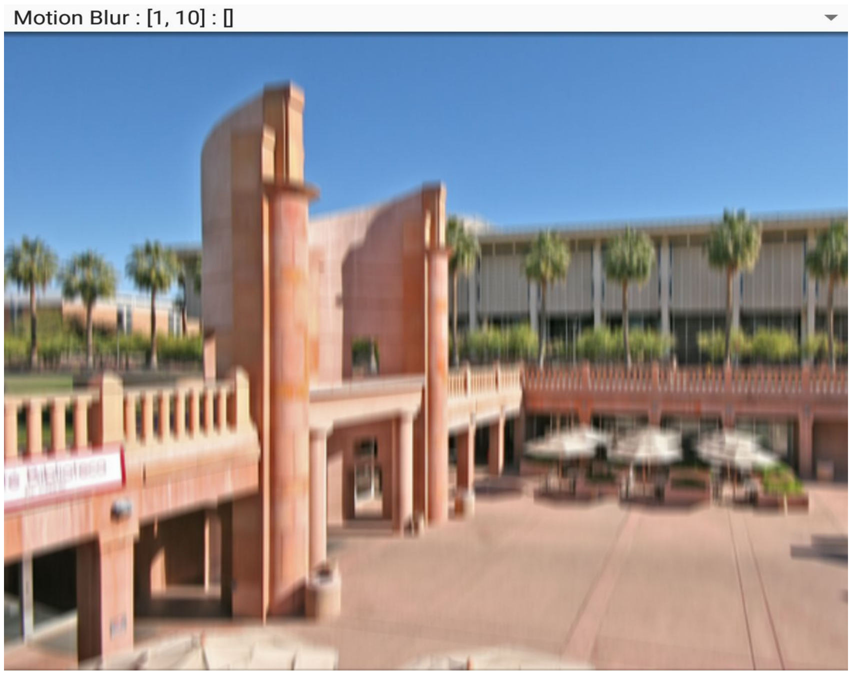

Unsharp Mask : [] : [10.0, 2.0]

## Color Filter Transform:

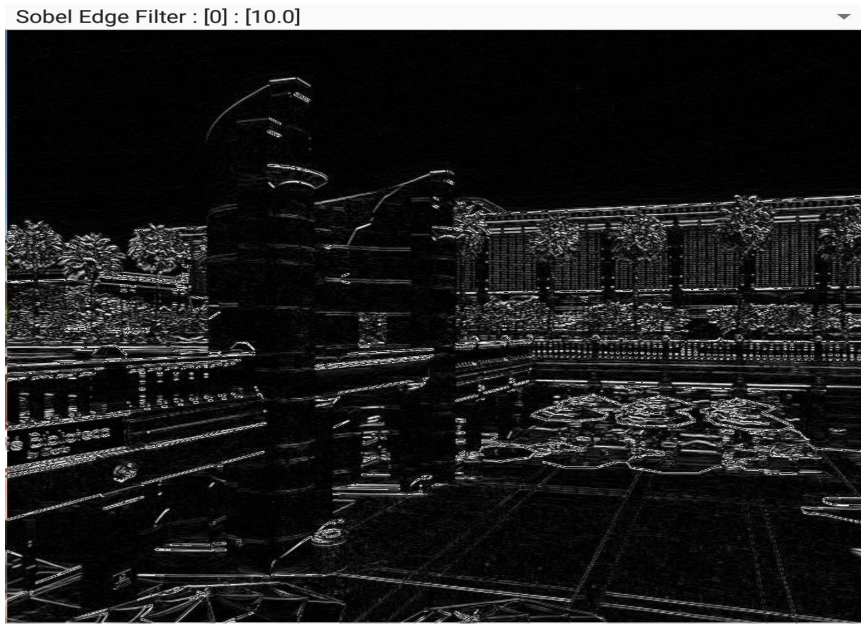

Color Filter : [0, 128, 64, 160, 128, 192, 192, 255, 0, 128, 64, 60, 128, 92, 192, 25..

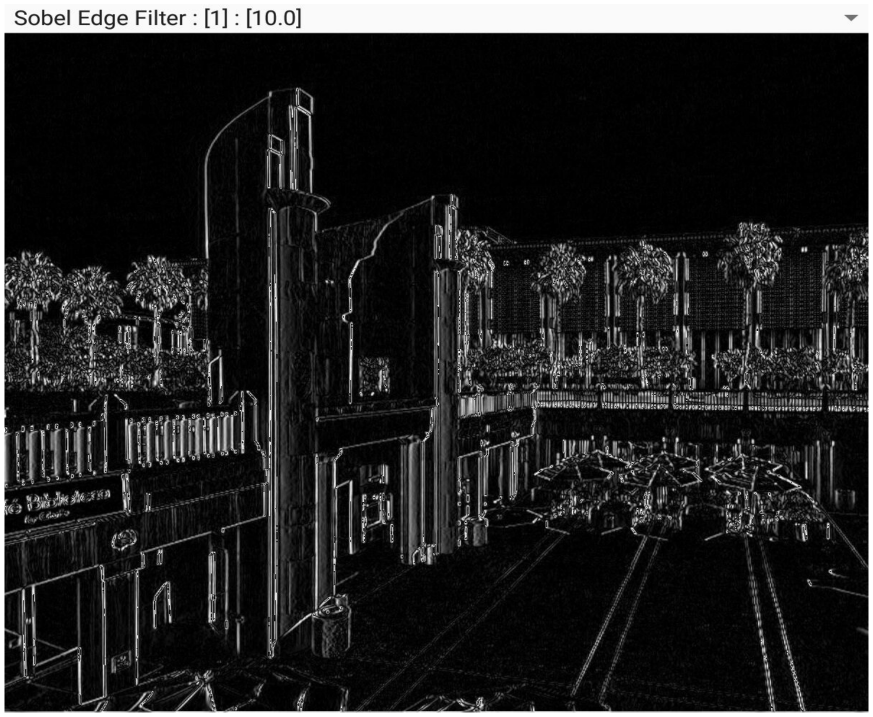
## Motion Blur Transform:

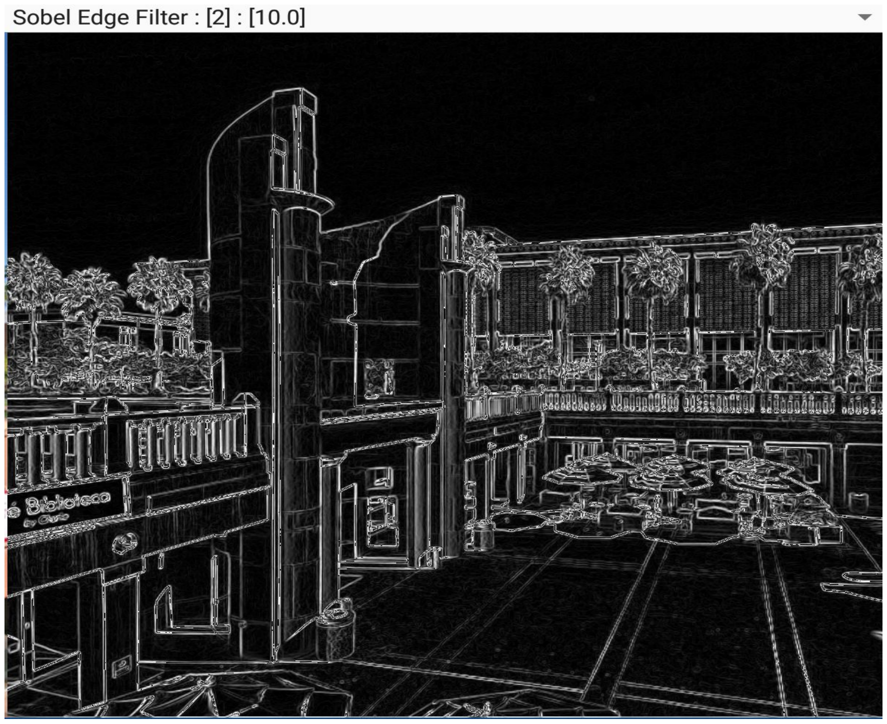

Motion Blur : [0, 10] : []


Motion Blur : [1, 10] : []

**Sobel Edge Filter :**







## Unfinished :

1) **NEON :** We tried to implement the motion blur using neon. But we could not implement it successfully. The code is implemented in method “getVerticalneon” in file “edu_asu_msrs_artcelerationlibrary_NativeClass.cpp”

## Strategy

**Division of labor:**

As the assignment components were completely new to us, we decided to take some time learning about library and service and communication between them. The tutorials given by professor helped a lot in understanding their implementation. Both of us took one major component of the assignment. After this we started working on the internal details of the components. We studied and discussed methods to implement FIFO queue. Then we worked on making the application to handle multiple threads.

For the final phase of the assignment, we divided the transforms among each other and started working on them independently. We implemented 5 transforms in android using JAVA. Then we started following the reference material shared on NDK and NEON. With help of these references we successfully used NDK to implement an image transform.

**Internal Checkpoints:**

| CheckPoint | Deadline | Team Member |
|---|---|---|
| Library implementation | 10/28/2016 | Omkar |
| Service implementation | 10/28/2016 | Abhishek |
| FIFO Queue | 11/04/2016 | Omkar |
| Class to implement threads for servicing multiple requests from library to service | 11/04/2016 | Abhishek |
| Class to implement threads for servicing simultaneous responses from service to library | 11/04/2016 | Abhishek |
| Gaussian Blur Transform | 11/11/2016 | Omkar |
| Unsharp Mask Transform | 11/18/2016 | Omkar |
| Color Filter | 11/25/2016 | Omkar |
| Sobel Edge Filter | 11/11/2016 | Abhishek |
| Motion blur Transform | 11/18/2016 | Abhishek |
| Color filter using NDK library | 11/28/2016 | Omkar |
| Understanding and testing NEON | 11/25/2016 | Abhishek |
| Input Validation | 11/30/2016 | Abhishek |
| Comments for all the code | 11/30/2016 | Omkar and Abhishek |
| Documentation | 11/30/2016 | Omkar and Abhishek |

## Challenges Encountered:

1) **Creation of threads both in Library and Service:**
Thread creation was necessary for handling simultaneous requests and responses between the library and service.

2) **Selection of Data type to implement FIFO queue:**

    There are various ways for implementing first in first out queue in android. Accurate selection of the queue implementation is necessary for non blocking operations.

3) **Gaussian Blur Transform takes large amount of time to complete:**

    We initially implemented the gaussian blur transform in very naive way. Due to this it took a lot of time for computation to complete. As the radius value increased, time required for completion of transform also increased linearly.

4) **Integrating NDK in android:**

    Creating a native library which will provide a function to do an image transform was one of the most difficult task in this assignment. We faced challenge understanding and implementing the format native functions follow.

5) **Using NEON in android:**

    Using NEON for faster image transform was another difficulty we faced while working on this assignment. We faced challenge understanding and implementing the SIMD instructions for processing the pixels of the bitmap. Also we observed that NEON registers are of size 128 bits. So we can store only 4 pixels of 32 bit at a time. As we were trying to implement motion blur using NEON, we needed an average of pixel values. For this we need to acquire the values of all color channels from all the pixels in the bitmap.

6) **Challenge in working on columns of bitmap using AndroidBitmap_lockPixels() function:**

    Another challenge we faced was that, with "AndroidBitmap_lockPixels" function in NDK, implementing computations on columns of bitmap is bit complicated task.

## How you overcame the challenges:

1) For creating the threads, we created a class which implements Runnable interface. Then we declared all the parameters required for execution inside the threads, as the member variables of the class. Inside the constructor we set all these parameters. This way we are passing the parameters to thread class. Then in run method of this class, we implemented the functionality expected.

2) First we decided to use timestamp as the deciding factor for FIFO queue. But as there was an unused integer parameter 'arg2' remaining in obtain method of Message class, we decided to use a request id for identifying individual request. This request id was deciding factor for FIFO execution. Thus we finalized to use this integer variable i.e. request id as data structure for queue. We decided to use Concurrentlinkedqueue in android for implementing the queue. A ConcurrentLinkedQueue is an appropriate choice when many threads will share access to a common collection. This implementation employs an efficient non-blocking algorithm.

3) To solve the time consumption problem with Gaussian Blur Transform, we utilised the symmetry property of the gaussian weight vector. We computed only 1st half of the values for this vector and copied these values in the remaining locations. This is because the computation of G(k) will be same for radius r and -r . This is because the computation involves the square of the variable k. And k is in the range [r, -r].

4) We followed the references given in assignment tutorials. To get acquainted with NDK, we first implemented a simple function in native library which will return a string. Then we verified if this function is working properly. Once we got the string form this function into the android code, we started looking into how the bitmap can be sent to function. Then we studied and understood data structure used by native code to store the bitmap information. Using this we kept on implementing small sections of the algorithm to do the transform. Finally we successfully implemented a transform in native library.

5) We followed the references given in assignment tutorials. To get acquainted with NEON, we first studied the example “hello-neon” given by google on github. First we tried to separate all color channels from the pixels of the bitmap using NEON instructions. And then again tried to combine these values into 1 pixel value. Then assigned that value in the input bitmap. But this implementation did not work as expected. We got a distorted image instead of original. We could not resolve this issue.

6) We manipulated the index of the array to traverse through the bitmap column wise. We stored pixels of the whole bitmap into 1D array. Then by manipulating the index of array we accessed these pixels row wise and column wise.

## Ways to Improve the Assignment:

**Assignment Level:**

Assignment is appropriate level of challenging for the given time period. At the end of checkpoint-1, we found that one of the fundamental aspect of the application is communication between the service and library.

The transforms given in the project description covers a range of filters which can be very useful from application user perspective. Also, it covers one of most important basic transform which is gaussian transform. The same transform functions can be used in other transform implementations such as Color Filter and Motion Blur transforms.

**More breadth-More depth:**

We think this assignment have appropriate breadth and depth, to understand the communication between the client (application user interface), library framework and the service. The second phase of application will be bit challenging as there are number of transforms to be implemented. Synchronization of all threads and the first in first out concurrent linked queue has to be properly implemented. For example, the implementation of location service and modules for the gallery as well as camera activity needed more synchronization with the application.

**Any fundamental flaws system description:**

At the end of phase-1, we feel that the description of goals for phase-1 has given us basic understanding on components of application. However, we feel that little bit broad description of how to synchronize the communication between the service and library could have made it better. This is not actually a flaw but just a secondary improvement in description.

**Enhancements(Design related developer’s ideas):**

With the architecture of concurrent linked queue, the communication between the service and library is carried out in proper way. There is no unnecessary load on main classes i.e library framework and implemented service. Hence, errors such as the unfortunately closed application are avoided.

The processing of the other transforms such as gaussian and sobel filter can be made faster by using NDK and Neon implementation. The application performance can be significantly improved using NDK and Neon.

Also, The user interface can be improved by giving user options to select the different parameters such as brightness, blur intensity level.

**Privacy/Security/Other Issues:**

- **Some Issues:**

  One of the common security issues with any application on Android is the data processed by application and its storing in the device external storage directory. For bitmaps processed the application uses the media file directory of device. With the internal and external storage permissions, any hacker can access the data stored in the library.

  Also, currently the application uses file descriptor between the library and service. Any open parcel file descriptor will lead to memory leak and the data can be accessed through the memory leak.