ArtCeleration Full Review Report

Group Members:

Yitao Chen 1206293582

Miao Tang 1207245100

1 Functionality

We implemented five transforms, three of them use Java, and two of them use NDK. In the following subsection, we will introduce more detail about each transform.

1.1 FIFO Implementation

We implement the FIFO using a message queue. We created a linked list on the ArtLib to receive the request from the application. We tested this structure by passing multiple request to the service to make sure that it satisfies the FIFO requirement. Our application is single-threaded. We are aware of that this structure may not bring us the optimal performance. It is a trade-off between performance and complexity compared to the multi-threaded counterpart. Considering that both of us have very limited experience on image process, we decided to use the single threaded method.

1.2 Color Filter

Color Filter is the first transform we worked on. This transform changes the color value of every pixel into another color value for each channel of RGB image. The transform was based on the specified piece-wise function. Since there were three channels, there were also three piecewise functions. Every piecewise function has eight numbers, which represented 4 points (x value and value) on the linearpiece wise function plot. Since there were three channels, we would be given an array including 24 numbers in total. These number would determine how the original figure will be transformed.

1.3 Ascii Art

The Ascii Art transform uses an enormous amount of (9 x 17 x 4) ASCII pictures to cover the original picture. It was more replacing or changing a small area of pixels than transforming because there was almost none calculation inside every pixel. Ascii Art will cut the whole image into many small regions that each region has the equal area with the ASCII images. Then the program will calculate the average number for each area and use an ASCII image with the closest average number to replace the original pixels inside it.

1.4 Motion Blur

The Motion Blur find the arithmetic average of the nearby pixels. For different "radius," a different number of terms are needed to process. So first is to find out the sum of the area of interest and then divided by the number of terms, which is 2*radius + 1.

1.5 Gaussian Blur

The Gaussian Blur transforms the input pixel values using Gaussian weighted kernel vector. The vector is first applied to the x-direction and then apply to the y direction. The radius determines how many terms are to multiply by the Gaussian weight vector.

1.6 Sobel Edge

The Sobel Edge filter transforms the input image first into a grayscale brightness image. Then use different edge filter to high light edges in the image.By applying different edge filters, we can obtain the gradients in horizontal and vertical direction. Once the pixel values are set, we can find the result.

2 Code Structure

artcelerationlibrary/java/edu.asu.msrs.artcelerationlibrary

- ArtLib
- ArtTransformService
- ColorFilter (Java, also converted into native)
- GaussianBlur (use Bitmap method)

- GassianBlurByte (use byte array method, also converted into native)
- MotionBlur
- SobelEdge

artcelerationlibrary/cpp/

- my-native-lib (includes native version of color filter and Gaussian blur)

3 Project Experience Description

3.1 Goals

For the full review of the assignment, we created 5 Art Transforms. We finished Color Filter, Motion Blur, Gaussian Blur, Sobel Edge Filter, and ASCII Art. Our goal for each is to realize the function of they were supposed to be, and also minimized the time as possible as we can.

3.2 Design

Describe your application components, including any activities or classes.

- Class Color Filter:
  - ArrayOperater()
    The is the method to proceed the color transform. The input of the method is all the image pixels, shift values for each color channels, and a piecewise array which includes 24 values. The output of this method is the image after transformed.
  - piecewise process()
    This is the main method to process the image. The input is the original image byte array, and a piecewise array which includes 24 values.The output is the image after processed.
- Class ASCII Art :

- ASCII()

  The main method in ASCII Art. The input is the pixel byte array; the output is the image after processed.

- getAvg()

  Get the average value of from every ASCII pixel. The input is every ASCII pixel; the output is the average value of all the pixels in channel R, G, B (not include A).

- PixelimageAve()

  Get the average value of every region on the original image. The input is the left top point of every region. The coordinate is (k,p). The k and the p are loop controlling variables in the code. The image byte array is also the input. The output is the average value of the region.

- findMin()

  Find the index of the element in the ASCII image array, which give you a reference indicating which ASCII image will be inserted in the closest average region. The input is The input is the left top point of every region. The image byte array is also the input. The output is the index of the selected ASCII image.

- Class MotionBlur

  - motionBlur() The main method of this class. It takes the byte array from the library as one of the input arguments. The input arguments also include image width and the integer array for transform test. The input byte array is converted into a bitmap object for easier pixel color information access. But the trade off is that the image processing time will be large. We are aware of this but still decided to do it this way so that we have less chance of facing pixel indexing issues.

    The color value of each pixel is obtained by using getPixel method under the Color class. The method returns an int value of a particular color channel for the pixel at a given coordinate. Then we find the arithmetic mean of the adjacent input pixel using the sum divided by 2*radius + 1. After calculating the arithmetic average value, use setPixel method to set the new color value back in the pixel.

- Class GaussianBlur Before
  - GaussianBlurFromJNI() This is the main method in GaussianBlur native code. We extract the color values of the input byte array and store them into three 2D arrays. We first generate Gaussian weight vectors and then perform two independent transforms to obtain the final result. We found the Gaussian vector first and then use it to find the output. The input arguments include radius and sigma. We find first find the result of the middle term then make a for loop to find out the result from the terms on the left and on the right.
  - convertToInt() This method converts the input byte array into three 2D arrays. Each of the array sizes is equal to image width multiply by height.
  - gKernel() This method will be re-used to finish the first transform and the second transform.
  - processOne() We implemented the first transform in this method.
  - processTwo() We implemented the second transform in this method. After performing the two-step process, the color values stored in the array will be converted into a byte array and send back to the client.
- Class SobelEdge
  - getColorValue() Extracts the color information from the bitmap object.
  - grayScale() This method is to calculate the grayscale brightness image color values and store the values in a 2D array.
  - setGrayScale() Set the color values calculated from the grayScale() method into the image to convert the image into a grayscale brightness image.
  - getGrx() and getGry() These two methods find the gradients using filters Sx and Sy. We separate the gradients calculation, and the gradient color setting is to simplify the process of finding the Gr.
  - gradient() This method sets the color values to the image using the gradient result obtained previously. According to the input arguments, one of the results from cases in Grx, Gry and Gr will be selected and set to the image.

Describe interfaces between application components.

ArtLib: in this class, we created two messengers to send and receive an image from the service. RequestTransform passes the source image() method. We created a MemoryFile for sharing an image between library and service. The ParcelFileHandler obtains the MemoryFile object file descriptor. The message sent through the data bundle to the service.

ArtTransformService: In this class, we use two handlers. The first one is to receive data from the library side while the second one is to send the processed image using MemoryFile.

3.3 Strategy

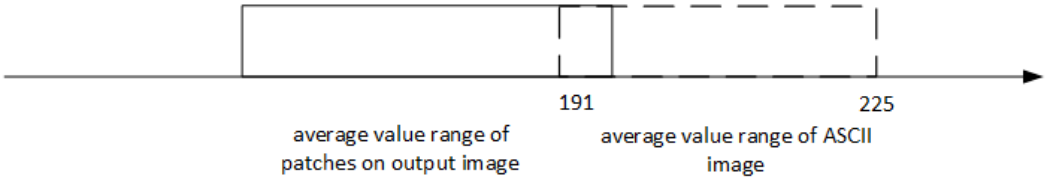
- **How did your team manage a division of labor?** We worked on the project together. Yitao focused more on the Motion Blur, Gaussian Blur, Sobel Edge Filter, and NDK. Miao focused on the Color Filter, ASCII Art, and lab report.

- **Describe challenges encountered along the way**

  - **ASCII Art**

    We encountered difficulties when we were dealing with Array operation in the ASCII Art transform. We solved this problem by splitting the whole question into seven steps, and every time focused on one-dimension array. We used the different layer of for loops to work on various tasks. We also found the range of the average of ASCII images was away from the range of the average of each region in the original pictures. This leads to the effect that the minimum indexes are all the same. We solved this problem by multiply an factor to each ASCII average value.

    The algorithm, in our opinion, has some imperfection. We collected the calculated all the ASCII image average values and found that the range falls from 191 to 225. However, There exists a bunch of regions whose average is below 191, which means 191 is the closest number of all the region average values. As a result, we are probably not able to see the ASCII Art effect as it supposed to be.



Ave char0 =205	Ave char17 =225
Ave char1 =196	Ave char18 =217
Ave char2 =220	Ave char19 =215
Ave char3 =195	Ave char20 =208
Ave char4 =209	Ave char21 =220
Ave char5 =209	Ave char22 =200
Ave char6 =192	Ave char23 =210
Ave char7 =199	Ave char24 =209
Ave char8 =224	Ave char25 =213
Ave char9 =216	Ave char26 =198
Ave char10 =203	Ave char27 =211
Ave char11 =219	Ave char28 =208
Ave char12 =200	Ave char29 =213
Ave char13 =209	Ave char30 =211
Ave char14 =207	Ave char31 =215
Ave char15 =199	Ave char32 =199
Ave char16 =200	Ave char33 =226
Ave char34 =191	Ave char35 =200

– **Gaussian Blur**

We encounter memory allocation problem when converting the Gaussian Blur Java code into native. The error message says that certain memory address region can not be released since is not allocated. But the error message didn't indicate which array or each line is causing the problem. It took us some time to find out that the problem problem is related to memory release. The reason is that we flip the width and the height of one of the arrays during the release process.

Also, we encounter the a problem when the sigma becomes large (i.e., 12), the image becomes darker. According to the Wikipedia, Gaussian kernel will goes to one if sum up all the terms. Then we realized that we should not convert the float intermediate result into int.

3.4 Describe ways in which you would improve the assignment

It would be better if the class cover more contents about our projects.

3.5 Unfinished

NEON

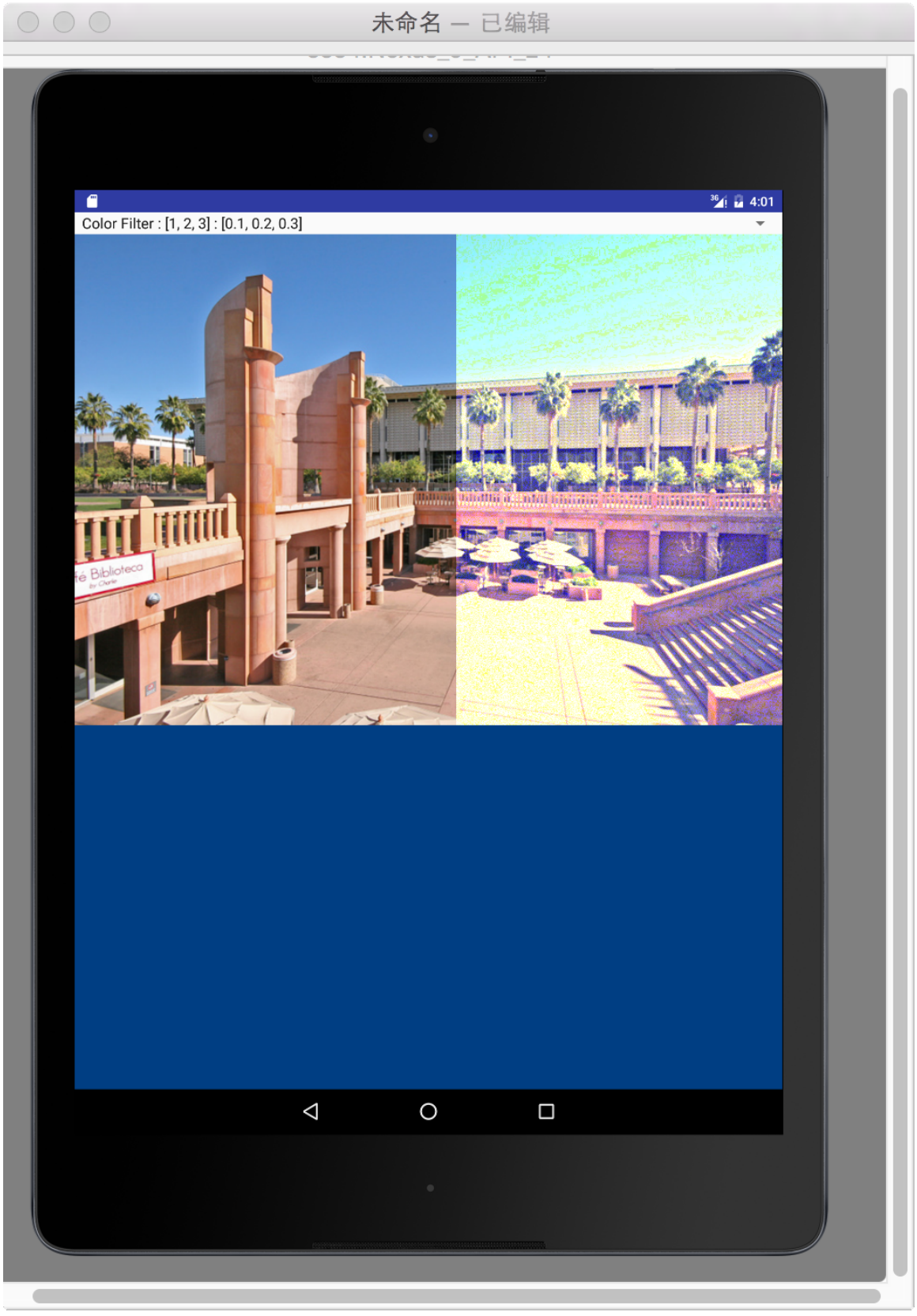


Figure 1: Color Filter

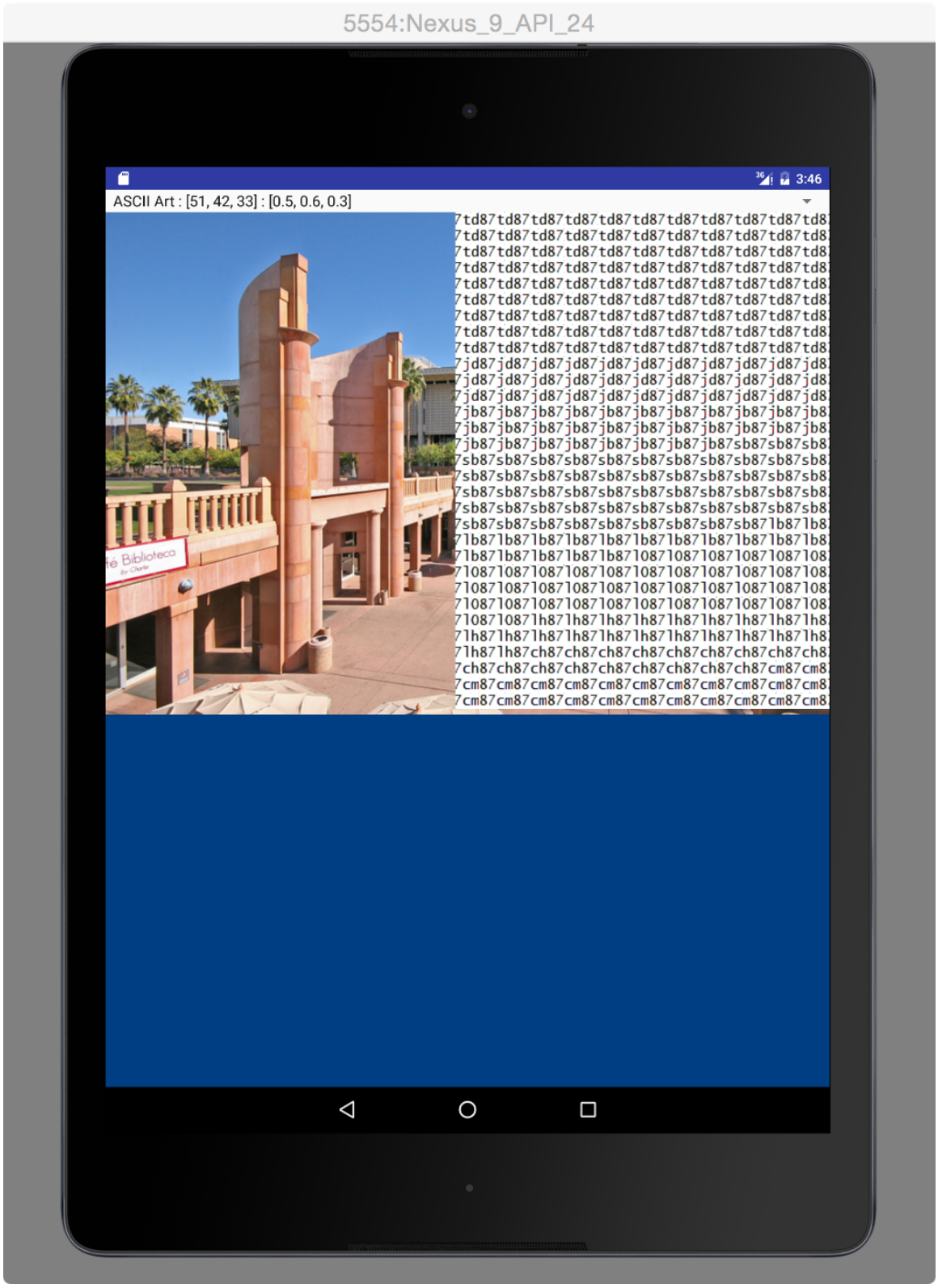


Figure 2: ASCII Art

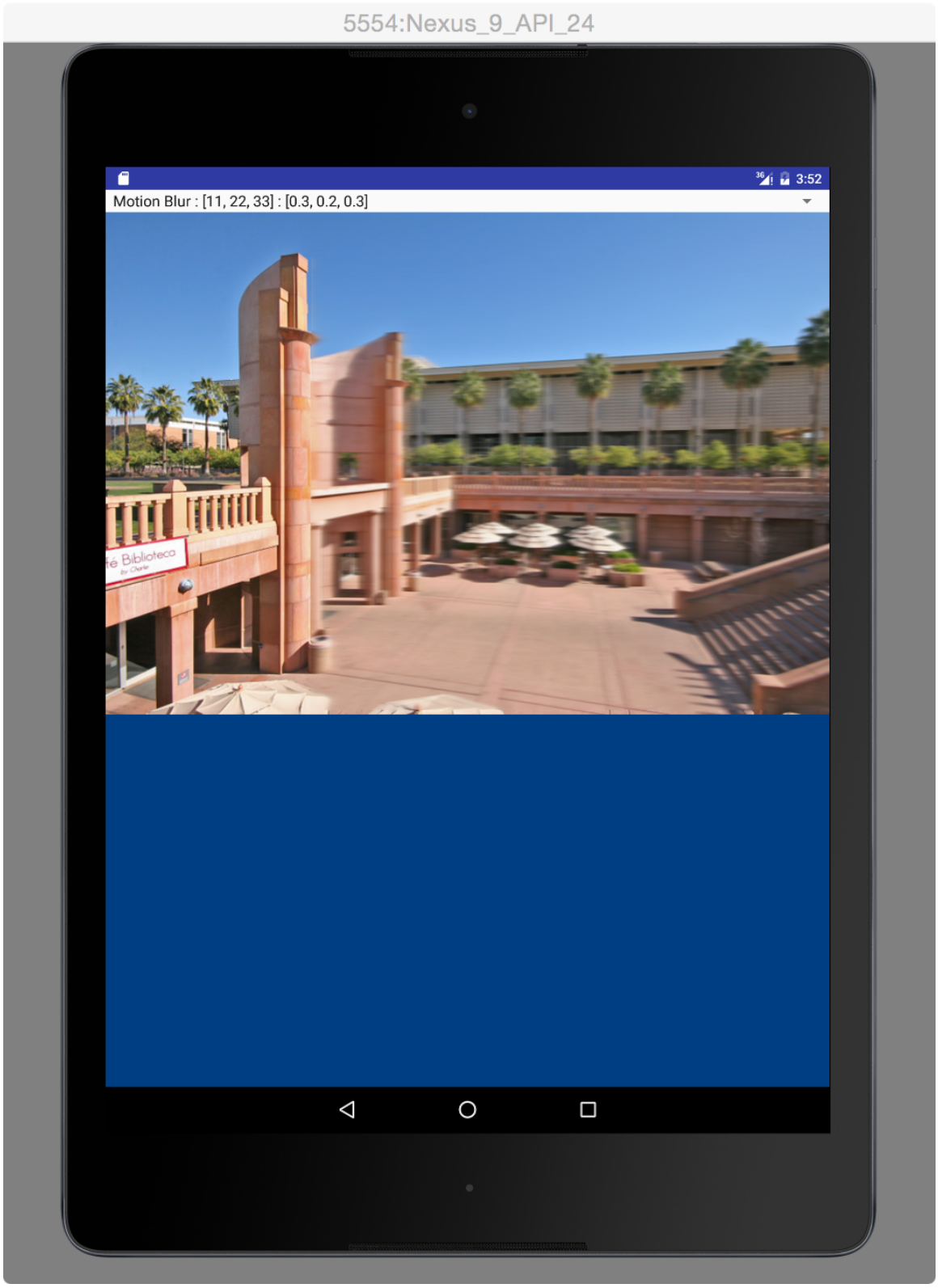


Figure 3: Motion Blur

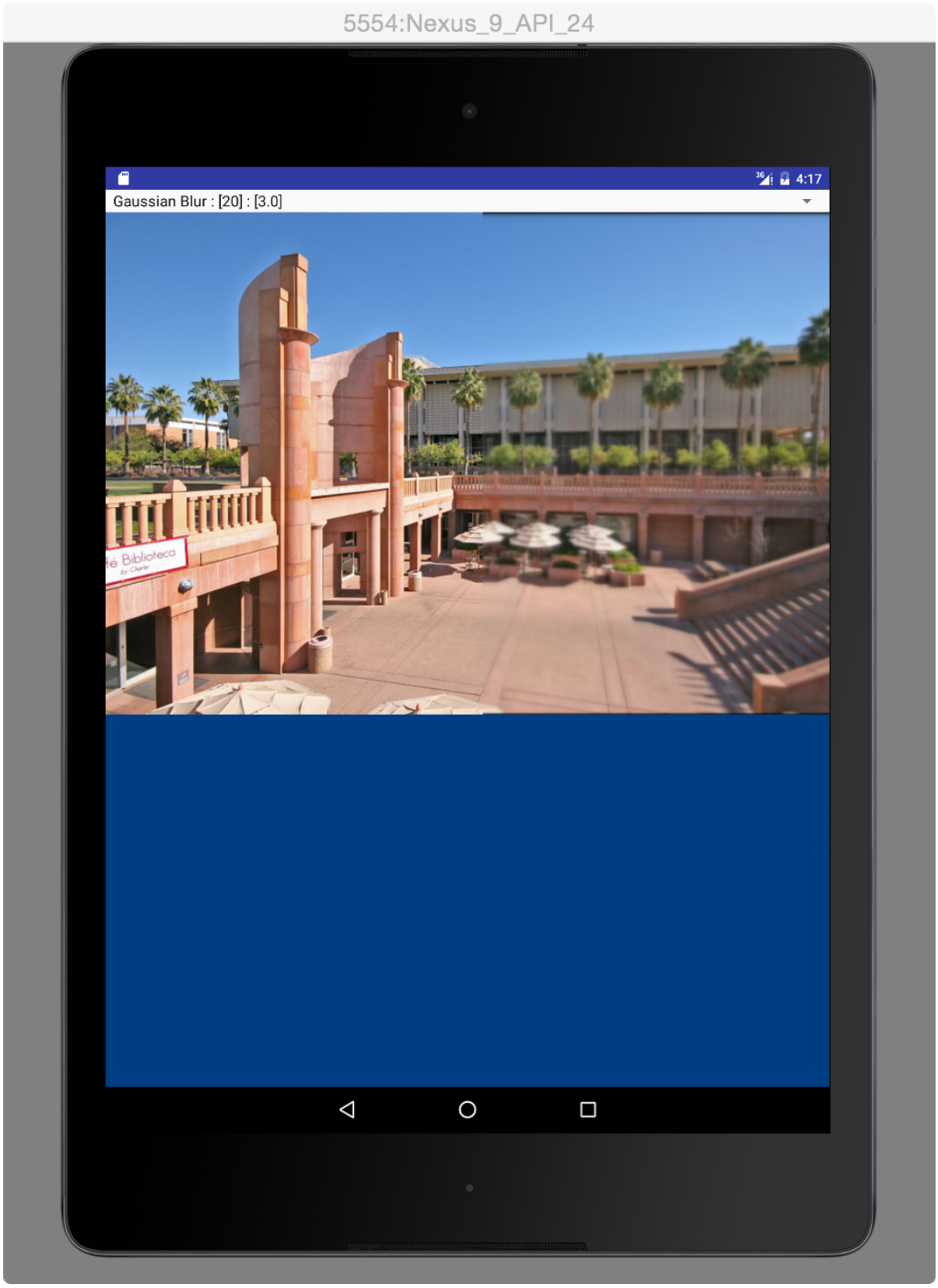


Figure 4: Gaussian Blur

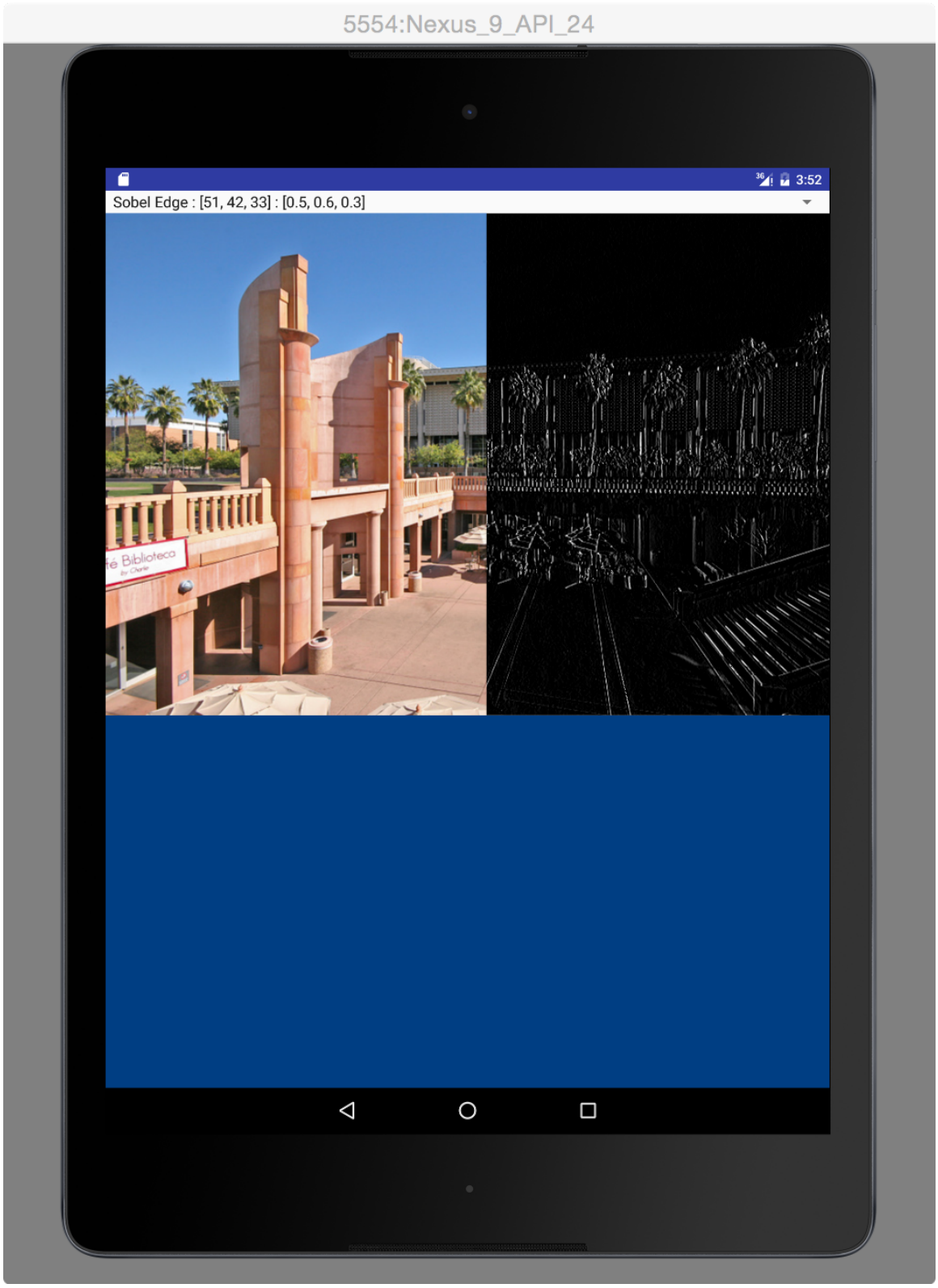


Figure 5: Sobel Edge