```
 1
 2  #include <jni.h>
 3  #include <string>
 4  #include "native-lib.h"
 5  #define _USE_MATH_DEFINES // for C++
 6  #include <math.h>
 7  #include <android/log.h>
 8
 9
10  #define  LOG_TAG    "DEBUG"
11  #define  LOGD(...)  __android_log_print(ANDROID_LOG_DEBUG,LOG_TAG,
    __VA_ARGS__)
12  extern "C"
13  //jstring
14  /**
15   * Created by tangmiao on 11/27/2016.
16   * This transform was basically trans-form the color value of every pixel into another
17   * color value for each channel of RGB image.  The transform was based on the specified
18   * piece-wise function.  Since there were three channels, there were also three piecewise
    functions.
19   * Every piecewise function was given by eight numbers, which represented 4 points(x value
    and value)
20   * onthe linear piece wise function plot.  Since there were three channels, we would
    be given
21   * an array including 24 numbers in total.  These number would determine how the original
    figure will be transformed.
22   */
23  jbyteArray
24
25  Java_edu_asu_msrs_artcelerationlibrary_ArtTransformService_ColorFilterFromJNI(
26          JNIEnv *env,
27          jobject /* this */,
28          jbyteArray array,
29          jintArray intArgs) {
30
31      jbyte* pixels = env->GetByteArrayElements(array, 0); //pass byte array to pointer
32      int length = env-> GetArrayLength(array);
33      int* piecewiseArray = env->GetIntArrayElements(intArgs,NULL);
34
35
36
37          for (int i = 0; i < length/4; i++) {
38              pixels[4*i+1] = ArrayOperater(pixels[4*i+1],0, piecewiseArray);
39              pixels[4*i+2] = ArrayOperater(pixels[4*i+2],8, piecewiseArray);
40              pixels[4*i+3] = ArrayOperater(pixels[4*i+3],16, piecewiseArray);
41          }
42
43          env->SetByteArrayRegion (array, 0, length, pixels); // c++ to java, return java
44          env->ReleaseByteArrayElements(array, pixels, 0);
```

```
45          return array;
46
47
48  }
49  //Function: transform the pixel values according input args
50  //Input: Original image pixels, different channel indexes, and piecewiseArray
51  //Output: all the  image pixels after processed
52  jbyte ArrayOperater(jbyte pixel1,int colorshift, int* piecewiseArray) {
53
54      int pixel = pixel1 & 0xFF;
55
56      if (pixel< 0) {
57          pixel = 0;
58      }else if (pixel >= 0 || pixel < piecewiseArray[0+colorshift]) {
59          pixel = (pixel)*(piecewiseArray[1+colorshift])/(piecewiseArray[0+colorshift]);
60      }else if (pixel >= piecewiseArray[0+colorshift] || pixel < piecewiseArray[2+
    colorshift]) {
61          pixel= piecewiseArray[0+colorshift]+(pixel-piecewiseArray[0+colorshift])*((
    piecewiseArray[3+colorshift]-piecewiseArray[1+colorshift])/(piecewiseArray[2+
    colorshift]-piecewiseArray[0+colorshift]));
62      }else if (pixel >= piecewiseArray[2+colorshift] || pixel < piecewiseArray[4+
    colorshift]) {
63          pixel = (piecewiseArray[2+colorshift]+(pixel-piecewiseArray[2+colorshift])*((
    piecewiseArray[5+colorshift]-piecewiseArray[3+colorshift])/(piecewiseArray[4+
    colorshift]-piecewiseArray[2+colorshift])));
64      }else if (pixel >= piecewiseArray[4+colorshift]|| pixel < piecewiseArray[6+colorshift
    ]) {
65          pixel = (piecewiseArray[4+colorshift]+(pixel-piecewiseArray[4+colorshift])*((
    piecewiseArray[7+colorshift]-piecewiseArray[5+colorshift])/(piecewiseArray[6+
    colorshift]-piecewiseArray[4+colorshift])));
66      }else if (pixel >= piecewiseArray[6+colorshift]|| pixel < 255){
67          pixel = (piecewiseArray[6+colorshift]+ (pixel - piecewiseArray[6+colorshift])* (
    255 - piecewiseArray[7+colorshift])/(255 - piecewiseArray[6+colorshift]));
68      } else {
69          pixel = 255;
70      }
71
72
73      return  (jbyte)pixel;
74    }
75
76  /*The Gaussian Blur transforms the input pixel values using Gaussian weighted kernelvector
    .
77  The vector is first applied to the x direction and then apply to the y direction.The radius
78  determines how many terms are to multiply by the Gaussian weight vector.*/
79  extern "C"
80  jbyteArray
    Java_edu_asu_msrs_artcelerationlibrary_ArtTransformService_GaussianBlurFromJNI(
81          JNIEnv *env,
```

```
 82          jobject /* this */,
 83          jbyteArray array,
 84          int w,
 85          int h,
 86          jintArray args1,
 87          jfloatArray args2) {
 88
 89      jbyte* b = env->GetByteArrayElements(array, 0); //pass byte array to pointer
 90      int length = env-> GetArrayLength(array);
 91      int *intArray = env->GetIntArrayElements(args1, NULL);
 92      float *floatArray = env->GetFloatArrayElements(args2, NULL);
 93
 94
 95      float** red = create2DArray(w,h);
 96      float** green = create2DArray(w,h);
 97      float** blue = create2DArray(w,h);
 98
 99
100
101      convertToInt(b,w,h,red,green,blue);
102
103
104      processOne(red,w,h,floatArray[0],intArray[0]);
105      processTwo(red,w,h,floatArray[0],intArray[0]);
106
107      processOne(green,w,h,floatArray[0],intArray[0]);
108      processTwo(green,w,h,floatArray[0],intArray[0]);
109
110      processOne(blue,w,h,floatArray[0],intArray[0]);
111      processTwo(blue,w,h,floatArray[0],intArray[0]);
112
113
114
115
116      for (int pixel = 0, row = 0, col = 0; pixel < h*w*4; pixel += 4) {
117
118          b[pixel + 0] = (jbyte)(red[row][col]);
119          b[pixel + 1] = (jbyte)((green[row][col]));
120          b[pixel + 2] = (jbyte)((blue[row][col]));
121          b[pixel + 3] = (jbyte)255;
122          col++;
123          if (col == w) {
124              col = 0;
125              row++;
126          }
127      }
128
129      cleanupArray(red, h);
130      cleanupArray(green, h);
```

```
131         cleanupArray(blue, h);
132
133
134         env->SetByteArrayRegion (array, 0, length, b); // c++ to java, return java
135         env->ReleaseByteArrayElements(array, b, 0);
136         return array;
137     }
138
139
140 //Function: extracts color values from byte array and stores them into 2d array
141 //Input: image byte array, img width, img height, color 2d arrays
142 //Output: null
143
144 void convertToInt(jbyte* b, int w, int h, float** red, float** green, float** blue) {
145         for (int pixel = 0, row = 0, col = 0; pixel < h*w*4; pixel += 4) {
146             red[row][col] = b[pixel + 0] & 0xff;
147             green[row][col] = b[pixel + 1] & 0xff;
148             blue[row][col] = b[pixel + 2] & 0xff;
149             col++;
150             if (col == w) {
151                 col = 0;
152                 row++;
153             }
154         }
155     }
156
157
158 //Function: creates 2d color array
159 //Input: array size
160 //Output: 2d array
161
162 float** create2DArray(int w, int h){
163
164     float** color = new float*[h];
165     for(int i = 0; i < w; ++i)
166         color[i] = new float[w];
167     return color;
168 }
169
170
171 //Function: release 2d color array
172 //Input: array size
173 //Output: null
174
175 void cleanupArray(float** array, int h){
176
177 for(int i = 0; i < h; ++i) {
178     delete[]  array[i];
179 }
```

```
180 delete [] array;
181
182 }
183
184 //Function: Gaussian transform step one
185 //Input: 2d color array, img width, img height, sigma, radius
186 //Output: null
187
188 void processOne(float** color, int w, int h, float sigma, int r) {
189       for (int row = 0; row < h; row++ ){
190          for (int col = 0; col < w; col++){
191             color[row][col] = color[row][col]*gKernel(0,sigma);
192
193             for (int k = 1; k<=r; k++){
194                if((row < k) || ( row + k >= h)){
195                   color[row][col] += 0;
196                } else{
197                   color[row][col] += color[row+k][col]*(gKernel(k,sigma));
198                   color[row][col] += color[row-k][col]*(gKernel(-k,sigma));
199
200                }
201
202             }
203          }
204       }
205
206    }
207
208
209
210 //Function: Gaussian transform step two
211 //Input: 2d color array, img width, img height, sigma, radius
212 //Output: null
213
214    void processTwo(float** color, int w, int h, float sigma,int r) {
215       for (int row = 0; row < h; row++ ){
216          for (int col = 0; col < w; col++){
217             color[row][col] = color[row][col]*(gKernel(0,sigma));
218
219             for (int k = 1; k<=r; k++){
220                if((col < k) || ( col + k >= w)){
221                   color[row][col] += 0;
222                } else{
223                   color[row][col] += color[row][col+k]*(gKernel(k,sigma));
224                   color[row][col] += color[row][col-k]*(gKernel(-k,sigma));
225
226                }
227
228             }
```

```
229
230             }
231         }
232
233     }
234
235 //Function: find Gaussian kernel with given radius and sigma value
236 //Input: radius, sigma
237 //Output: Gaussian kernel
238
239  float gKernel(int k, float t){
240
241         float g;
242         g = (float)exp(-(k*k)/(2*(t*t)));
243         g = g*(float)1/(float)sqrt(2*M_PI*t*t);
244
245         return g;
246     }
247
```