Artceleration Library and Service

Author

By Jianan Yang and Wenhao Chen

Goal

Artceleration is an Android library framework / service which enables user application to implement artistic transforms for images. In current version, it includes 5 different image tranformation functions for app developer to use - COLOR_FILTER, MOTION_BLUR, GAUSSIAN_BLUR, SOBEL_EDGES and NEON_EDGES.

Achievements

We successfully implemented five different image transform algorithms. Out of five, three of them are written in `Java` and two of them are written in `C++`. We tried to use neon to perform the last part of `NeonEdge` algorithm which is the linear combination of the original image and processed image. However, we think we failed at the last step. We think the code should work but noting happened to the image. Due to the time limit we instead just implement the `NeonEdge` using `Java`.

Client-end App

The general idea of this library/service is to realize image transformation for app developers so they don't have to worry about building their own image process algorithm, instead they can just pick and use. Below is a sample application which uses our Artceleration library framework/service. In this app, user can specify the image transformation type from the drop-down located on top of screen, the transformed image will be showing if you drag the image left and right. Below are the examples:

1. Color_Filter

2. Motion_Blur

3. Gaussian_Blur

4. Sobel_Edge

5. Neon_Edge

Framework/Service Design

In summary, the client's requests are sent to service for image processing, once processing is done, the processed image is sent back to client and shown on app's UI.

In the first step, the user must create a libaray object `artlib` through `artlib = new ArtLib(this)`. During the library object creation, the activity is binded to service using `init()` method, the binding request is sent through the `Intent()` object, in which the inputs are an activity object and service object.

```
public void init() {
        mActivity.bindService(new Intent(mActivity, TransformService.class),
mServiceConnection, Context.BIND_AUTO_CREATE);
}
```

In the service class, the the `onBind()` callback function is trigured by `bindService()` method and a `IBinder` is returned by `onBind()` method as the binder corresponding to a messenger `mMessenger` which will be used to hanlde message transferring.

```
@Override
    public IBinder onBind(Intent intent) {
        return mMessenger.getBinder();
    }
```

If the service is connected succefully, the `onServiceConnected()` callback function is triggered, one

of the input is the `IBinder` replied from the `onBind()` method and this `IBinder` object is connected to another `Messenger` object `mMessenger`in library object to communiate with the `Messenger` in service object.

```
        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            mMessenger = new Messenger(service);
            mBound = true;
        }
```

As of here, the communication `bridge` - `IBinder-Messenger` between client and service is setup via `init()` method in libaray.

Next, the user must register a `TransformHandler()` interface oject into the libaray. This is done by calling `registerHandler()` method with an `TransformHandler` oject as input, here we are using anonymous inner class. The `onTransformProcessed` method is over written and will be called once the transformation is done, this callback funciton has a `Bitmap` object as input argument and it is the processed image. The transformed image is shown on UI by using `setTransBmp()` method.

```
artlib.registerHandler(new TransformHandler() {
            @Override
            public void onTransformProcessed(Bitmap img_out) {
                Log.d("In the mainviewr","img_out");
                artview.setTransBmp(img_out);
            }
        });
```

In library, the function `requestTransform(Bitmap img, int index, int[] intArgs, float[] floatArgs)` is defined, it has four arguments, when this function is called in activity. Before anything happens, it firstly check if the input arguments for image tranform is valid or not, through `argumentValidation()` method which takes the `index`, `intArgs`, `floaArgs` as its input. It it passess the validation check, it will proceed to send image and parameters for process.

During the data transition, firstly, we create a `ByteBuffer` object `buffer` which will be used to store the `Bitmap`. The `Bitmap` object `img` is then put in to the `buffer`. By doing so, we avoid the compression of `Bitmap` which significantly speed up the message transition. The `ByteBuffer` is then transformed into a `byte[]` object `byteArray`. This byteArray is then write in to ashmem `MemoryFile` object `memoryFile` using `writeBytes()` function which takes the `byteArray` two offsets and the size of the memory as arguments. Then the information of this ashmem is storted in `ParcelFileDescriptor` object `pfd` and this `pfd`, along with the other input arguments `intArgs` and `floatArgs`, are bundled using `putParcelable()`, `putIntArray()` and `putFloatArray()` method respectively. Then this `dataBundle` is stored in a `Message` object `msg` USING `setData()` function. The image transformation algorithm index is stored in the `msg` as well which determines what function should the service perform. Then this `Message` is send to service by `Messenger` using `send()` function. An another important thing is set up anthoer `Messenger` to handle the msg send back by

service, here we use `msg.replyTo` to set this Messenger as `inMessenger`.

```
public boolean requestTransform(Bitmap img, int index, int[] intArgs, float[]
floatArgs) {
        //validate the input parameter first
        if(!argumentValidation(index, intArgs, floatArgs))
            return false;
        try {
            //Write the image to the memory file
            //Firstly,convert bitmap to byte array
            //Without using compress, to speed up.
            int width = img.getWidth();
            int height = img.getHeight();
            int  bytes = img.getByteCount();
            ByteBuffer buffer = ByteBuffer.allocate(bytes); //Create a new buffer
            img.copyPixelsToBuffer(buffer); //Move the byte data to the buffer
            byte[] byteArray = buffer.array();

            //Secondly, put the stream into the memory file.
            MemoryFile memoryFile = new MemoryFile("someone", byteArray.length);
            memoryFile.writeBytes(byteArray, 0, 0, byteArray.length);
            ParcelFileDescriptor pfd =
MemoryFileUtil.getParcelFileDescriptor(memoryFile);
            memoryFile.close();
            Bundle dataBundle = new Bundle();

            //put the image in the bundle, sharing the memory with ashmen
            dataBundle.putParcelable("pfd", pfd);
            dataBundle.putIntArray("intArgs", intArgs);
            dataBundle.putFloatArray("floatArgs", floatArgs);

            //index means the type of the transform.
            int what = index;
            Message msg = Message.obtain(null, what,width,height);
            msg.setData(dataBundle);
            msg.replyTo = inMessenger;
            try {
                mMessenger.send(msg);
            } catch (RemoteException e) {
                e.printStackTrace();
                return false;
            }
        } catch (IOException e) {
            e.printStackTrace();
            return false;
        }
        return true;
    }
```

```
private boolean argumentValidation(int index, int[] intArgs, float[] floatArgs){
        int intLength = intArgs.length;
        float floatLength = floatArgs.length;
        switch (index) {
            case TransformService.COLOR_FILTER:
                if(intLength!=24){
```

```
                return false;
            }
            int pre = intArgs[0];
            for(int i = 0; i<intLength;i++){
                if(intArgs[i]<0 || intArgs[i]>255)
                    return false;
                if(i%2 == 0 && i%8 != 0) {
                    if(intArgs[i] <= pre)
                        return false;
                    else {
                        if(i+2<intLength)
                            pre = intArgs[i];
                    }
                }
                if(i%8 == 0)
                    pre = intArgs[i];
            }
            break;
        case TransformService.MOTION_BLUR:
            if(intLength != 2)
                return false;
            if(intArgs[0] != 0 && intArgs[0] != 1)
                return false;
            if(intArgs[1] <= 0)
                return false;
            break;
        case TransformService.GAUSSIAN_BLUR:
            if(intLength != 1 || floatLength != 1)
                return false;
            if(intArgs[0]<=0||floatArgs[0]<=0)
                return false;
            break;
        case TransformService.SOBEL_EDGE:
            if( intLength != 1 )
                return false;
            if(intArgs[0] != 0 && intArgs[0] != 1 && intArgs[0] != 2)
                return false;
            break;
        case TransformService.NEON_EDGES:
            if(floatLength != 3)
                return false;
            if(floatArgs[0]<=0)
                return false;
            //if()
            break;
        default:
            return false;

    }
    return true;
}
```

Once the request or message is sent to service, the `ArtTransformHandler()` callback function is invoked and the transformation index is retrived by calling `msg.what` in `handleMessage(Message msg)` callback functions. The previous desinated `Messenger` used to handle the send-back message is set

to be `replyTo`. And this `replyTo Messenger` is going to be used later to send back processed images. Then, the `Bitmap` is retrived by calling `getBitmap()` method. This is doing in background because we are using `AsynTest()` object which inherits `AsynTask` class.

```
class ArtTransformHandler extends Handler {
        @Override
        public void handleMessage(Message msg) {
            replyTo = msg.replyTo;
            TransformType = msg.what;
            try {
                new AsyncTest().execute(getBitmap(msg));
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

    }
```

Then the image saved in the ashmem is retrived from the `getBitmap(msg)` fucntion with the Message as input argument. The `Bundle` is first get by `getData()` method and the ashmem information is get from the key-value pair with key as `pfd`. Then the data stream is get from the `ParcelFileDescriptor` object `pfd` and saved in `InputStream` object `istream` and then `istream` which stores the `Bitmap` data is converted to `byte[]` by calling `IOUtils.toByteArray()` method and saved in `byteArray`, and saved in a `ByteBuffer` object `buffer`. An empty `Bitmap img` is created with `ARGB_8888` configuration. And the `Bitmap` image is created by copy the pixel values from the `buffer`. The othe imformation transmmited by `Messenger` is also retrived from the `Bundle`. All the data are then saved into a `TransformPackage` object for later usage.

```
    private TransformPackage getBitmap(Message msg) throws IOException {
        TransformPackage tP = new TransformPackage();
        Bundle dataBundle = msg.getData();
        ParcelFileDescriptor pfd = (ParcelFileDescriptor) dataBundle.get("pfd");
        InputStream istream = new ParcelFileDescriptor.AutoCloseInputStream(pfd);
        //Convert the istream to bitmap
        byte[] byteArray = IOUtils.toByteArray(istream);
        //The configuration is ARGB_8888, if the configuration changed in the
application, here should be changed
        // a better way is to pass the parameter through the message.
        Bitmap.Config configBmp = Bitmap.Config.valueOf("ARGB_8888");
        Bitmap img = Bitmap.createBitmap(msg.arg1, msg.arg2, configBmp);
        ByteBuffer buffer = ByteBuffer.wrap(byteArray);
        img.copyPixelsFromBuffer(buffer);
        int[] intArgs = dataBundle.getIntArray("intArgs");
        float[] floatArgs = dataBundle.getFloatArray("floatArgs");

        dataBundle.putFloatArray("floatArgs", floatArgs);
        tP.img = img;
        tP.intArgs = intArgs;
        tP.floatArgs = floatArgs;
        return tP;
    }
```

Then the `AsyncTask` callback function `doInBackground` is triggered once the `excute()` is finised, and the returned `TransformPackage` object `tP` is used as its input. Depend on the `TransformType`, which is a `int` value passed by `msg.what`, it will choose certain type of transformation. The detail of image transformation will be discussed later. The input arguments for image transformation is also parsed here. Once the transform finishes, the processed `img` is returned.

```
@Override
        protected Bitmap doInBackground(TransformPackage... tP) {

            Bitmap img = null;
            switch (TransformType) {
                case COLOR_FILTER:
                    NativeTransform n = new NativeTransform();
                    n.colorFilter(tP[0].img, tP[0].intArgs);
                    img = tP[0].img;
                    Log.d("Finished","COLOR_FILTER");
                    break;
                case MOTION_BLUR:
                    NativeTransform m = new NativeTransform();
                    m.motionBlur(tP[0].img,tP[0].intArgs);
                    img = tP[0].img;
                    Log.d("Finished","MOTION_BLUR");
                    break;
                case GAUSSIAN_BLUR:
                    GaussianBlur gaussianBlur=new GaussianBlur(tP[0].img,
tP[0].intArgs,tP[0].floatArgs);
                    img =  gaussianBlur.startTransform();
                    Log.d("Finished","GAUSSIAN_BLUR");
                    break;
                case SOBEL_EDGE:
                    SobelEdgeFilter sobelEdgeFilter=new
SobelEdgeFilter(tP[0].img,tP[0].intArgs[0]);
                    img = sobelEdgeFilter.startTransform();
                    Log.d("Finished","SOBEL_EDGE");
                    break;
                case NEON_EDGES:
                    NeonEdge.NeonEdgeTransForm(tP[0].img,tP[0].floatArgs);
                    img = NeonEdge.NeonEdgeTransForm(tP[0].img,tP[0].floatArgs);
                    Log.d("Finished","NEON_EDGES");
                    break;
                default:
                    break;
            }
            return img;
        }
```

Once the `doInBackground()` method is done, the `onPostExecute()` callback function is invoked. The `imageProcessed()` function to send the image back to library. This method has one argument, the processed `Bitmap` image. A `Messenger` is created and set to be the same value of previously define `replyTo` messenger. It uses the same method to put the image data in ashmem and send the `ParcelFileDescriptor` object `pdf` back through the `Messenger`. The Messenger used to transfer the

image back to library is saved in a Messenger list `ArrayList<Messenger>` with an object `mClicent`. Then the processed image using a specific function is send back to library by calling `mClients.get(0).send(msg);`. The message queue is hanlde by Android, the input message is put into the queue in each thread, and the looper will get the msg from the queue and run it.

```
private void imageProcessed(Bitmap img) {
        if(img == null)
            return;

        int width = img.getWidth();
        int height = img.getHeight();
        int what = 0;
        Message msg = Message.obtain(null, what, width, height);
        msg.replyTo = replyTo;

        //Message msg = Message.obtain(null, what);
        Bundle dataBundle = new Bundle();
        mClients.add(msg.replyTo);
        if (msg.replyTo == null) {
            Log.d("mclient is ", "null");
        }
        try {
            int bytes = img.getByteCount();
            ByteBuffer buffer = ByteBuffer.allocate(bytes); //Create a new buffer
            img.copyPixelsToBuffer(buffer); //Move the byte data to the buffer
            byte[] byteArray = buffer.array();

            /*ByteArrayOutputStream stream = new ByteArrayOutputStream();
            img.compress(Bitmap.CompressFormat.PNG, 100, stream);
            byte[] byteArray = stream.toByteArray();*/
            //Secondly, put the stream into the memory file.
            MemoryFile memoryFile = new MemoryFile("someone", byteArray.length);
            memoryFile.writeBytes(byteArray, 0, 0, byteArray.length);
            ParcelFileDescriptor pfd =
MemoryFileUtil.getParcelFileDescriptor(memoryFile);
            memoryFile.close();
            dataBundle.putParcelable("pfd", pfd);
            msg.setData(dataBundle);
            //msg.obtain(null,6, 2, 3);
            mClients.get(0).send(msg);
        } catch (RemoteException | IOException e) {
            e.printStackTrace();
        }
    }
```

Then in the libraray `ArtLib`, the send back message is handled by a `ImageProcessedHandler` class which extends `Handler`. The callback function `handleMessage(Message msg)` is invoked when receiving a message, and the image is retrived using the same method as described above. Finally, the image is sendback to client through the previously registered interface `TransformHandler` by calling its method `onTransformProcessed()` with the Bitmap image as input argument.

```
static private class ImageProcessedHandler extends Handler {
```

```
        @Override
        public void handleMessage(Message msg) {//Called when get the message from
the service. Usually mean that a transform is finised.
            Bundle dataBundle = msg.getData();
            ParcelFileDescriptor pfd = (ParcelFileDescriptor)
dataBundle.get("pfd");
            if(pfd == null){
                Log.d("pfd","null");
            }else {
                Log.d("image ","has been sent back to the client");
                InputStream istream = new
ParcelFileDescriptor.AutoCloseInputStream(pfd);
                //convertInputStreamToBitmap
                byte[] byteArray = new byte[0];
                try {
                    byteArray = IOUtils.toByteArray(istream);
                } catch (IOException e) {
                    e.printStackTrace();
                }

                Bitmap.Config configBmp = Bitmap.Config.valueOf("ARGB_8888");
                Bitmap img = Bitmap.createBitmap(msg.arg1, msg.arg2, configBmp);
                ByteBuffer buffer = ByteBuffer.wrap(byteArray);
                img.copyPixelsFromBuffer(buffer);
                if (artlistener != null) {//triger the listener to send back the
processed image to the activity
                    artlistener.onTransformProcessed(img);
                }
            }
        }
    }
```

Image Transform Algorithms Implementation

Five different image process algorithms are implemented in this APP. Three of them are written in `Java`, i.e. `GaussianBlur`, `SobelEdge` and `NeonEdge`, while the other two are written in native language `C++`, they are `ColorFilter` and `MotionBlur`. Each of them are created as an individual class.

The `Java` image process classes are located in `Java_edu_asu_msrs_artcelerationlibrary` folder, with the name of `GaussianBlur.class`, `SobelEdgeFilter.class` and `NeonEdge.class`. To use them, you just need to simply create a corresponding class with `Bitmap` image and transform paramters as its inputs. Then call `startTransform()` method and the process will start and the processed `Bitmap` will be returned.

The two `native(C++)` image process classes are located in `cpp_native-lib` folder, with the name of `native-lib.cpp`. The `C++` methods are interfaced with a `Java` class, `NativeTransform.class`, with `JNIEXPORT` and `JNICALL`. To use these algorithms written in `native` language, you can just create a `NativeTransform` object and all its correspoinding methods, i.e. `motionBlur()` and `colorFilter()`, in the body of these two methods, they call `nativeMotionBlur()` and `jniColorFilter()` which runs in `native` library.

```
extern "C"
{
    JNIEXPORT void JNICALL
Java_edu_asu_msrs_artcelerationlibrary_NativeTransform_jniColorFilter(JNIEnv * env,
jobject  obj, jobject bitmap, jintArray args, uint32_t size);
    JNIEXPORT void JNICALL
Java_edu_asu_msrs_artcelerationlibrary_NativeTransform_nativeMotionBlur(JNIEnv *env,
jobject obj, jobject bitmap, jintArray args);
}
```

FIFO implementation

The FIFO is actually realized by relying on the internal control of Android's `MessageQueue`. Everytime you send a `TransformRequest`, the message is queued in the `MessageQueue`, once the image process is done, the processed image is sent back in the order of what is the message queued in `MessageQueue`. We've tested this and the image process indeed is sent back with the same order to `TransformRequest`.

Strategy

1. In general, we discuss coding logic and brainstorm ideas together. As for task, one person majorly dedicated on building up framework/service and the other person focuses on debugging and documentation writing. We also separate the image transform algorithm into half, one person writes three algorithms in Java while the other person writes two algorithms in C++.
2. We meet weekly, checkout progress, solve issues and make plans for the next week. We made several internal check points:

```
            - discuss and try to understand the logic behine assignment;
            - review and type the servie/library-setup code taungt in class;
            - finish the rest of code required for a complete service/library,
majorly how to send processed image back to client side;
            - documentation writing for checkout point 1;
            - setup native labrary;
            - setup neon environment;
            - algorithms development;
            - app integration and debug;
            - documentation writting for final;
```

Challenges

There are several challenges associated with this project.

As for check point-1, the major challenge is to understand how binder and messenger works together to send message and the logic behinde. Another chanllenge is how to send processed image back to client.

As for check point-2, the major challenge is to set up native libaray and neon environment and

interact them with higher layer languages-Java. Also writing code using neon is a big problem for us.

Improvement

The biggest improvement is writting all five algorithms in C++, we believe it will enhance the image process speed significantly! Also, try neon could be another potential solution for speed enhancement, however, it requires a good understanding of neon coding style.

Second, to make it even user friendly, it's better to be able to access the user's photo database and do transforms on costumers' pictures, which could be really fun.