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, TILT_SHIFT and NEON_EDGES.

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 can 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 in the yellow region which is a dummy image transformation we implement for this checkout point.

Sample app

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 argument, when this function is called in activity. The `Bitmap` object `img` is firstly compressed and put in to a `ByteArrayOutputStream` object `stream` and then this `stream` is converted to byteArray saved in `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` is bundled and 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) {

        try {
            ByteArrayOutputStream stream = new ByteArrayOutputStream();
            img.compress(Bitmap.CompressFormat.PNG, 100, stream);
            byte[] byteArray = stream.toByteArray();

            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();
            dataBundle.putParcelable("pfd", pfd);
            dataBundle.putIntArray("intArgs", intArgs);
            dataBundle.putFloatArray("floatArgs", floatArgs);

            int what = index;
            Message msg = Message.obtain(null, what);
            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;
    }
```

Once the request or message is sent to service, the `handleMessage()` callback function is invoked and the transformation index is retrived by calling `msg.what` in `transformHelper(msg)` method.

```
@Override
        public void handleMessage(Message msg) {
            trasnformHelper(msg);
        }
```

```
private void trasnformHelper(Message msg) {
        switch (msg.what) {
            case COLOR_FILTER:
                break;
            case MOTION_BLUR:
                break;
            case GAUSSIAN_BLUR:
                break;
            case TILT_SHIFT:
                break;
            case NEON_EDGES:
                break;
            case TEST_TRANS:
            default:
                break;
        }
        Bitmap mutableBitmap = getBitmap(msg);
        testTransform(mutableBitmap);
        imageProcessed(mutableBitmap, msg);
    }
```

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 the image is built up by calling `decodeStream()` method which has `istream` as input argment and image is saved in a `Bitmap` object `img` and then this Bitmap is converted to `ARGB_8888` format and then returned.

```
private Bitmap getBitmap(Message msg) {
        Bundle dataBundle = msg.getData();
        ParcelFileDescriptor pfd = (ParcelFileDescriptor) dataBundle.get("pfd");
        InputStream istream = new ParcelFileDescriptor.AutoCloseInputStream(pfd);

        Bitmap img = BitmapFactory.decodeStream(istream);
        return img.copy(Bitmap.Config.ARGB_8888, true);
    }
```

Then we did a very simple change to the image just as a test, we change a certain part of the image into Yellow. Once the image processing is done, we call the `imageProcessed()` function to send the image back to library. This method has two arguments, one the the process `Bitmap` image the other is `Message`. The Bitmap should be the processed image, the Message should be the message which was sent in before. Firstly, the image is compressed and changed to `byte[]` and saved into another ashmem using the same method before. Then the ashmem info is bundled and saved in the message using the same method as before. 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, Message msg){
        int what = 0;
        Bundle dataBundle = new Bundle();
        mClients.add(msg.replyTo);
        if (msg.replyTo == null) {
            Log.d("mclient is ", "null");
        }
        try {
            ByteArrayOutputStream stream = new ByteArrayOutputStream();
            img.compress(Bitmap.CompressFormat.PNG, 100, stream);
            byte[] byteArray = stream.toByteArray();
            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, get `Bundle` - get `ParcelFileDescriptor`- get `InputStream` and get `Bitmap`. 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) {
```

```
            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);
                Bitmap img = BitmapFactory.decodeStream(istream);
                if (artlistener != null) {//triger the listener to send back the
processed image to the activity
                    artlistener.onTransformProcessed(img);
                }
            }
        }
    }
```

Strategy

1. In general, we discuss coding logic and brainstorm ideas together. As for task, one person majorly dedicated on writing code and the other person focuses on debugging and documentation writing.
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;
```

Challenges

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.

Improvement

We will try to improve the performance of AsyncTask and using multi-threading method to do multiple image processing in the same time.