Implementing the basic CT backprojecting algorithm in android mobile application

Arman S Kussainov¹,², Maxim Em¹, Yernar Myrzabek¹ and Maksat Mukhatay¹
¹ Department of Physics and Technology, al-Farabi Kazakh National University, 71, al-Farabi Ave, Almaty, 050040, Republic of Kazakhstan
² National Nanotechnology Laboratory of Open Type, 71, al-Farabi Ave, Almaty, 050040, Republic of Kazakhstan
E-mail: arman.kussainov@gmail.com

Abstract. We have implemented the basic steps for the FDK backprojecting algorithm in computed tomography. Application works from the set of preloaded projections and uses OpenCV libraries for FFT, convolution, frequency space image filtering, image’s brightness, contrast and quality manipulation. Compared to the desktop implementation, the calculation-intensive part of the application was moved to the asynchronous background task hosted by an android fragment. This allows the task to survive the application’s configuration changes and to run in the background even if the main activity was destroyed. The minimalistic interface with the access to all main backprojecting parameters was implemented. The result of backprojection is saved as an image in the download folder of the phone. The user also has the control over the reconstructed slice location along the Z axis.

1. Introduction
The immense growth of computational power of the mobile platforms includes the availability of multicore CPU, high-performance graphic chips, and the increase in the bandwidth of the electronic component interfaces. Interfaces with multiple peripheral devices as well as internal sensors are available on smartphones as well. This means that it is almost obligatory to implement basic, if not all, computational algorithms in physics and mathematics on mobile and similar android operated devices.

The 3D reconstruction on mobile phones is currently in active development phase, and it came from being just promising [1] to robust innovative solutions like [2]. Not only the processing power of the phone is used, but also the plethora of its sensors [3]. Slightly outdated comprehensive review for radiology smartphone applications is given at [4], where the most sophisticated are represented by DICOM viewers. In the best-case scenario, if the calculation intensive part exists, it is usually moved to the remote server for execution. We are personally focused on the limited task of creating an application to process the different sets of medical images on handheld devices to implement computed tomography algorithms.

Users of the more mature desktop operating systems like Windows, Mac an Linux with corresponding programming languages are offered with the complete set of prebuilt graphic interfaces and default multithreading optimization like OpenMP, OpenMPI etc. The situation is different for the programmer on the mobile platforms, where he should take care of freeing
the main thread for user interface action, implement this interface in xml code, and connect it to the computational logic of the app. These computations should be implemented separately, on a different from the main thread, keeping the user interface responsive and ensure that the computations survive the orientation and other application’s state changes.

Additionally, android devices programmer needs to implement the major image processing routines as simple as matrix multiplication, regular FFT or convolution by himself or to delegate these tasks to some OpenCV libraries. These libraries are the only multi-platform and multi-language libraries for the entry-level computer vision and image processing researcher. In this paper, we implemented the FDK backprojection algorithm using OpenCV libraries; see [5] and [6] for algorithm description. The application is tested to be supported by the Android API version as old as 18th.

2. Methods

2.1. General software and IDE configuration

The current version of Android Studio (Android Studio Arctic Fox | 2020.3.1) and OpenCV libraries, prebuilt for use in the Android app project, were used for our simulation. OpenCV (Open Source Computer Vision Library) is a cross platform, an open source computer image processing and software library [7]. The latest version of OpenCV for Android SDK (opencv-3.4.3-android-sdk.zip) was downloaded using the link [8]. The OpenCV deployment examples, reprinted and compiled from multiple sources, for Eclipse and Android Studio IDEs, are readily available for reference. In particular, the instructions explaining how to use OpenCV in your Android project could be found in [9].

All our application’s modules were written using Java programming language. The current Java SDK version is automatically maintained by the Android IDE. As an alternative, the Kotlin programming language is actively promoted by the Google developers as the programming language of choice. Nevertheless, nobody, in the foreseeable future, seems to be going to abandon Java, native C/C++, or Python libraries.

For the development and testing purposes our application was initially written and run on the Android Virtual Device simulator for API 28 and 30 as well as tested on physical devices with Android Lollipop, Marshmallow and Oreo systems installed.

2.2. Basic data projecting algorithm and its implementation

In the FDK algorithm, the data from a recorded projection is projected back towards the imaginary source, following the path of the original X-ray beam; see Figure 1.

**Figure 1.** The backprojecting algorithm’s geometry for the FDK method. An upper cross-sectional view is given. The slice, with phantom shown in blue, is reconstructed in the XY fixed laboratory frame. The black spheres represent the inner structure of the phantom. The field of view, which determines the size of the reconstructed slice, is shown as a dashed rectangle and could be smaller or larger than the real phantom. The reconstruction of the single point/pixel P’ in image is given. The SOO and SP’P triangles are used to calculate all weighting coefficients. The projection is digitized with step ∆p.
The projected data is shown as a dashed line to represent the discrete nature of the digital projection. In this interpretation the $\Delta p$ refers to the distance between individual radiation sensitive elements of the detector or the data sampling rate $1/\Delta p$ for reconstruction purposes. The data, with corresponding scaling, convolution and weighting coefficients, is summed iteratively to the field of interest from all projections; see below for the details. The Fourier transform apparatus allows one to interpret this convolution as matrix multiplication, while from the computer vision and image processing point of view, this situation could be interpreted as filtering in frequency domain.

Only this backprojecting part of the CT experiment was implemented. The projections themselves were generated by our desktop full CT simulation software package [10, 11] implemented for Windows OS. Some of these sample projections, for the different angles of rotation, are shown in Figure 2.

Our digital phantom is constructed as a simple biconvex shape ellipsoid with the single spherical cavity centered along all three axes of the ellipsoid. The material of the phantom is chosen to be of pure water and the cavity inside the phantom is vacuum. The size of the ellipsoid is $160 \times 120 \times 160$ voxels, and the size of the cavity is $90 \times 90 \times 90$ voxels. To relate the CT setup to the real physical experiment, the size of the voxel is taken to be $1/72$ fraction of a centimeter. The distance from source to phantom is 2000 voxels, and from phantom to detector is 1040 voxels.

The X-ray radiation interaction with digital phantom is based on the absorption coefficients database; see [12]. For the X-ray photons energy provided in the range from 1 and up to 2000 KeV, the water’s value of the mass attenuation coefficient $\mu/\rho$ starts from the maximum at $4.078 \times 10^3$ g/cm$^3$, for 1 KeV, and goes down to $1.813 \times 10^{-2}$ for 2 MeV. The density of the water is taken to be of a reference 1 g/cm$^3$ value. The X-ray spectrum was chosen to be produced by a 600 kV accelerating voltage and as powerful as 10 mGy of specific air kerma for the simulated G-297 rotating anode from the Varex Imaging X-ray source.

Provided projections are the square $512 \times 512$ *.png* type image files uploaded to the Downloads folder of the phone. There are 361 of them taken with 1 degree step from 0 to full 360 degree rotation of the source and detector around the phantom. They are available for download from a Google Drive link in the built-in application manual; see [13].

The mathematical part of the FDK algorithm is extensively studied in different implementations and for various trajectory shapes. For the uniform divergent fan beam and the equidistantly spaced detector cells, one needs to implement the following three steps to backproject the data back on the plane of interest

(i) Each digitized projection $P_{\beta_i}(n\Delta p)$ (see Figure 1), acquired for $\beta_i$ angle of rotation and sampled with step size $\Delta p$, should be multiplied by a factor $D(n\Delta p)$; see the expression in

---

**Figure 2.** Sample X-ray projections used for the reconstruction algorithm. From top to bottom, the sample projections taken for the 0°, 45°, and 90° degrees values of the rotation angle; see the text for full description of the phantom and recording conditions.
the brackets in the next equation

\[ P'_{\beta_i}(n\Delta p) = \left( \frac{SO}{\sqrt{SO^2 + n^2\Delta p^2}} \right) \cdot P_{\beta_i}(n\Delta p), \quad (1) \]

where the number of steps (detector’s pixels) \( n \) is counted both ways to the left and to the right from the center of the detector.

(ii) Each modified projection should be convolved with a frequency space high pass filter \( g(n\Delta p) \), which, in the real space, has the form

\[
g(n\Delta p) = \begin{cases} 
  +1/(8\Delta p^2), & \text{if } n = 0 \\
  0, & \text{if } n \text{ is even} \\
  -1/(2n^2\pi^2\Delta p^2), & \text{if } n \text{ is odd}, 
\end{cases}
\]

(2)

to produce filtered backprojection

\[ P_{\beta_i}^{filt}(n\Delta p) = P'_{\beta_i}(n\Delta p) * g(n\Delta p). \quad (3) \]

(iii) Ultimately, all contributions to the field of interest, from all recorded projections \( P_{\beta_i}^{filt}(n\Delta p) \), should be added with the weighting coefficient to form a reconstructed slice of the phantom

\[
slice(x, y) = A \sum_{i=1}^{\text{all projections}} \frac{1}{U^2(x, y, \beta_i)} \cdot P_{\beta_i}^{filt}(n\Delta p), \quad (4)
\]

where \( U(x, y, \beta_i) \) is given by the \( SO/(SO + OP) \) ratio for this particular backprojection ray and the pixel of reconstructed image; see Figure 1. The weighting coefficient \( A \), basically, defines, in a simplified way, the contrast that should be applied to each projection included in the reconstruction loop according to the simple formula \( P = A \cdot P + B \), where \( B \) is related to the brightness of the projection image. \( U(x, y, \beta_i) \) and \( P_{\beta_i}^{filt}(n\Delta p) \) are related through the beam trace connecting the \( n \)-th pixel on the projection and all points along this trace in \((x,y)\) plane.

Thus, our implementation requires some regular, nested, loops for calculating matrices \( U(x, y, \beta_i) \), \( D(n\Delta p) \) and \( g(n\Delta p) \), as well as OpenCV functions for elementwise matrix multiplication and FFT library to implement convolution. The android framework will provide the rest, including Mat-to-Bitmap transformation, etc.

2.3. Android project structure

It is important to preserve the long-running ongoing calculations from being destroyed and reinitialized after application configuration has changed. Although deprecated at API level 30, one of the graceful implementations recommends placing the calculations thread inside the android fragment [14]. The main thread is usually concerned with UI (user interface) only and communicates with calculations thread through callback interface. For the reference, the most up-to-date method requires the construction of the \texttt{ViewModel}; see [15].

Therefore, all calculations are moved to the \texttt{BackprojectionFragment} android fragment with a callback interface to control the execution, cancellation, and completion of the background task. This fragment hosts the private class \texttt{BackProjectionTask}, which extends \texttt{AsyncTask} and contains the \texttt{doInBackground} method. The last method runs in background and iterates through the supplied projections. For each \texttt{projection}, taken at specific angle \texttt{beta}, it calls the \texttt{Backprojection} class method \texttt{getprojected}, to cast data back on the plane of interests, which is called \texttt{slice}. Methods from \texttt{CalculateMatrices} and \texttt{Convolve} classes are used to calculate matrices \( U^2(x, y, \beta_i) \), \( D(n\Delta p) \) and \( g(n\Delta p) \) to convolve them with modified \texttt{projection} or multiply with reconstructed \texttt{slice}; see the code snippet below and Figure 3.
public class BackprojectionFragment extends Fragment {

    private class BackProjectionTask extends AsyncTask<Void, Bitmap, String> {

        @Override
        protected String doInBackground(Void... voids) {
            CalculateMatrices.D_and_g(g, D, ...);
            for (beta = 0; beta <= 360; beta++) {
                projection = projection.mul(D);
                projection = Convolve.getconvolved(projection, g);
                Backprojection.getprojected(projection, slice,...);
                CalculateMatrices.U2(U2, ...);
                slice = slice.mul(U2);
            }
        }
    }
}

For brevity and space preservation the presence of the missing code in the snippet indicated only by nonuniform changes in line numbers. The matrix multiplication method .mul() is provided by an OpenCV library.

Figure 3. Project’s diagram generated with Visual Paradigm Enterprises for Java source code.

The implementation of the equations 1 through 4 is straightforward; see [16] for $U^2(x,y,\beta_i)$, $D(n\Delta p)$ and $g(n\Delta p)$ matrix calculation methods.

3. Results
The minimalistic user interface helps to enter the source_to_phantom, phantom_to_detector and slice_size values; see Figure 4. Here, the slice_size corresponds to the size of the field of view.
(FOV) for the reconstructed cross-section image. The slice_position value places the FOV at the selected position along the Z axis. Reconstruction takes place in the plane parallel to the XY plane. The rotation_step says how often the original X-ray projections are taken, while rotating the coupled source and detector around the sample. Default values are pre-written in the corresponding fields. The default values in the input form correspond to the particular set of parameters with which the set of provided sample projections was recorded. Everything but the rotation_step is measured in voxels (or pixels). For now, the thickness of the reconstructed slice is fixed at 1 voxel.

Before the start of the simulation, the main screen displays the classic projection recording scheme for FDK method. Backprojection is started by clicking the "START BACKPROJECTION" button. During reconstruction, this button becomes "DISABLED" and the current state of the reconstructed slice replaces the previous picture; see Figure 4(b). You can cancel the reconstruction anytime by pressing the "CANCEL BACKPROJECTION" button and restart it again from the beginning with the same or different parameters. In both cases, the final or intermediate result will be saved in the Downloads folder as *.png image file with the name concatenated from the word "slice" and the current time stamp (for example, slice_26072021124605.png).

For the application to work, you should have projection files named projection_0, projection_1, ... projection_365 in your Downloads folder. The last digits in the file name stand for the rotation angle $\beta$ value. Some of these projections may be missing but the algorithm will skip their processing if it the case.

The application scrolling view hosts the short operation manual as well as provides the link for downloading the sample projections’ set from a Google Drive.

![Figure 4. Snapshots of the application's user interface taken before and during backprojection.](image)
The beta version implements only the slice reconstruction in the planes perpendicular to the 
Z plane. The other orientations are straightforward to implement and are the matter of the 
nearest future implementation and UI development.

Next Figure 5 displays the set of the coronal slices obtained for different number of projections 
used in reconstruction process, see the top row please, and, the set of slices reconstructed at 
different positions along the Z axis from the all 361 projections available, see the bottom row.

**Figure 5.** Top row, from left to right. Coronal slices reconstructed from a set of 19, 25, 37, 73 
and 360 projections, taken with a step of 25, 15, 10, 5 and 1 degrees correspondingly. Bottom 
row, from left to right. Coronal slices taken at different location along the Z axis, namely at 
210, 192, 180, 150 and 140 value. The FOV size is 256 $\times$ 256 pixels. Slice thickness in all cases 
is equal to 1 voxel.

All results are fully equivalent in quality to the similar desktop implementation of CT 
experiment. The performance of the application is very reasonable and has a considerable 
room for improvement.

4. Conclusions
We have implemented the fully functional reconstruction part of the FDK backprojection 
algorithm for android applications. The compact user interface provides control over the major 
reconstruction parameters. If necessary, one can cancel the long-running ongoing simulation and 
resume it with new data and reconstruction parameters. The results of the complete or partial 
reconstruction are automatically saved in the Downloads folder. Java language and OpenCV 
image processing libraries were used to write our project. The application has great potential for 
further development in terms of functionality in image handling and control, as well as in terms 
of its analytical and computational part. The current version of the application is available for 
download from [16].

Acknowledgements
The application development was done using computing resources provided by the Open Type 
National Nanotechnology Laboratory, Almaty, Republic of Kazakhstan.

References
[1] Klein G and Murray D 2009 Proc. of the 8th IEEE Int. Symp. on Mixed and Augmented Reality pp 83–86
[2] Tanskanen P, Kolev K and et al 2013 Proc. of the IEEE Int. Conf. on Computer Vision pp 65–72
[3] Tibana T K, Grubert R M and et al 2019 Radiologia brasileira 52 245–246
[4] Rodrigues M, Visvanathan A and et al 2013 Insights Imaging 4 555–562
[5] Kak A and Slaney M 2001 Principles of Computerized Tomographic Imaging (Philadelphia, PA: Society for Industrial and Applied Mathematics)
[6] Jain A 1989 Fundamentals of Digital Image Processing Prentice-Hall information and system sciences series (Hoboken, NJ: Prentice Hall)
[7] OpenCV Team 2021 OpenCV for Android SDK https://opencv.org/android/ accessed: 2021-08-02
[8] OpenCV Team 2021 OpenCV for Android SDK https://sourceforge.net/projects/opencvlibrary/files/opencv-android/ accessed: 2021-08-02
[9] Gad A 2021 A guide to preparing OpenCV for Android https://heartbeat.fritz.ai/a-guide-to-preparing-opencv-for-android-4e9532677809 accessed: 2021-08-02
[10] Kussainov A S and Saduev N O 2019 Journal of Physics: Conference Series 1391 012086
[11] Kussainov A S and Saduev N O 2021 Journal of Physics: Conference Series 1730 012079
[12] Hubbell J, Seltzer S M and et al 1996 Tables of X-ray mass attenuation coefficients and mass energy-absorption coefficients 1 keV to 20 MeV for elements Z=1 to 92 and 48 additional substances of dosimetric interest [electronic resource] (Gaithersburg, MD: USDOC, Tech.Admin., NIST)
[13] Kussainov A S 2021 Sample projections set https://drive.google.com/drive/folders/1V0_L1weGJE0LoFOQLJgJ4iqSP2c1SPb?usp=sharing accessed: 2021-08-02
[14] Lockwood A 2013 Handling Configuration Changes with Fragments https://www.androiddesignpatterns.com/2013/04/retaining-objects-across-config-changes.html accessed: 2021-08-02
[15] Android Academy Tutorial 2018 Updating the UI with background ViewModel tasks https://www.androidacademy.com/tutorials/architecture/updating-the-ui-with-background-viewmodel-tasks accessed: 2021-08-02
[16] Kussainov A S 2021 Android fdk backprojection algorithm https://github.com/Arman-Kussainov/Android_FDK.git accessed: 2021-08-02