Android GPUImage

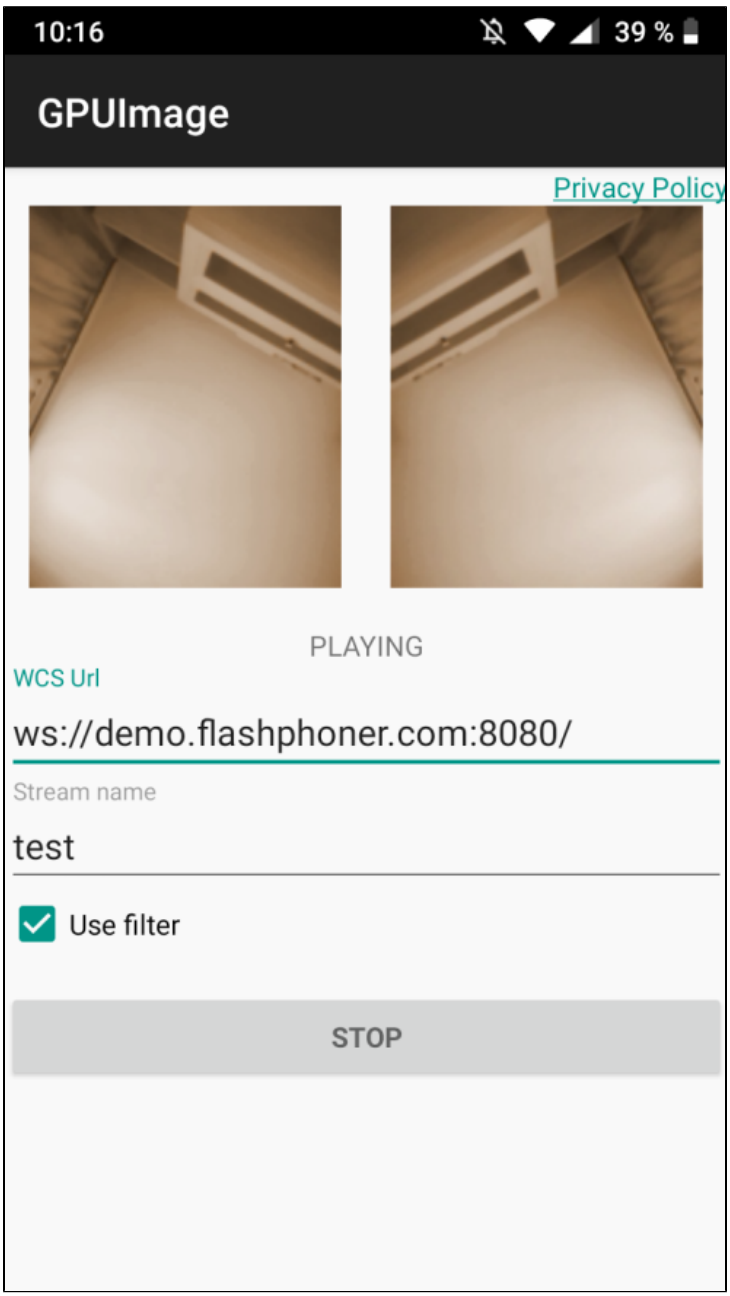
GPUImage library integration example to apply filters to a publishing stream

This example shows how to integrate the GPUImage library to the application for applying filters to the image while publishing a WebRTC stream. To do this, video capturing from custom software source feature is used. This example works with Android SDK builds since1.1.0.27

On the screenshot below, the sepia filter enabled is shown

Input fileds:

- WCS Url - WCS server address for Websocket connection
- Stream name - stream name to publish and play
- Use filter - checkbox to enable or disable the filter



Analyzing example code

To analize the code, take the following classes of gpu-image example, which is available to download in build1.1.0.27:

- main application activity classGPUImageActivity.java
- Camera1Capturer interface implementation classGPUImageCameraCapturer.java
- Camera1Enumerator interface implementation classGPUImageCameraEnumerator.java
- CameraSessioninterface implementation classGPUImageCameraSession.java

Note that interfaces implementation classes are placed to org.webrtc package,it is necessary to access camera capturing and management functions

1. API initializing

Flashphoner.init() code

```
Flashphoner.init(this);
```

2. Websocket session creation

Flashphoner.createSession()code

The SessionOptions object with the following parameters is passing to the method

- URL of WCS server
- SurfaceViewRenderer localRenderer to use to display a stream publishing (after filter applying)
- SurfaceViewRenderer remoteRenderer to use to display the stream playing

```
sessionOptions = new SessionOptions(mWcsUrlView.getText().toString());
sessionOptions.setLocalRenderer(localRender);
sessionOptions.setRemoteRenderer(remoteRender);

/**
  * Session for connection to WCS server is created with method createSession().
  */
session = Flashphoner.createSession(sessionOptions);
```

3. Connection establishing

Session.connect()code

```
session.connect(new Connection());
```

4. Receiving the event confirming successful connection

session.onConnected()code

```
@Override
public void onConnected(final Connection connection) {
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            mStatusView.setText(connection.getStatus());
            ...
        }
    });
});
```

5. Stream creation

Session.createStream()code

```
                                StreamOptions streamOptions = new StreamOptions(streamName);
                                Constraints constraints = new Constraints(true, true);
                                streamOptions.setConstraints(constraints);

                                /**
                                 * Stream is created with method Session.createStream().
                                 */
                                publishStream = session.createStream(streamOptions);
```

6. Permissions requesting

ActivityCompat.requestPermissions()code

```
@Override
public void onConnected(final Connection connection) {
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            ...
            ActivityCompat.requestPermissions(StreamingMinActivity.this,
                  new String[]{Manifest.permission.RECORD_AUDIO, Manifest.permission.CAMERA},
                  PUBLISH_REQUEST_CODE);
            ...
        }
        ...
    });
});
```

7. Stream publishing after permissions are granted

Stream.publish()code

```
@Override
 public void onRequestPermissionsResult(int requestCode,
                                        @NonNull String permissions[], @NonNull int[] grantResults) {
     switch (requestCode) {
         case PUBLISH_REQUEST_CODE: {
             if (grantResults.length == 0 ||
                     grantResults[0] != PackageManager.PERMISSION_GRANTED ||
                     grantResults[1] != PackageManager.PERMISSION_GRANTED) {
                 muteButton();
                 session.disconnect();
                 Log.i(TAG, "Permission has been denied by user");
             } else {
                 /**
                  * Method Stream.publish() is called to publish stream.
                  */
                 publishStream.publish();
                 Log.i(TAG, "Permission has been granted by user");
             }
             break;
         }
         ...
    }
}
```

8. Stream playback after successful stream publishing

Session.createStream(), Stream.play()code

```
                                    publishStream.on(new StreamStatusEvent() {
                                        @Override
                                        public void onStreamStatus(final Stream stream, final StreamStatus
streamStatus) {

                                            runOnUiThread(new Runnable() {
                                                @Override
                                                public void run() {
                                                    if (StreamStatus.PUBLISHING.equals(streamStatus)) {

                                                        /**
                                                         * The options for the stream to play are set.
                                                         * The stream name is passed when StreamOptions object
is created.
                                                         */
                                                        StreamOptions streamOptions = new StreamOptions
(streamName);
                                                        streamOptions.setConstraints(new Constraints(true,
true));


                                                        /**
                                                         * Stream is created with method Session.createStream().
                                                         */
                                                        playStream = session.createStream(streamOptions);
                                                        ...
                                                        /**
                                                         * Method Stream.play() is called to start playback of
the stream.
                                                         */
                                                        playStream.play();
                                                    } else {
                                                        Log.e(TAG, "Can not publish stream " + stream.getName()
+ " " + streamStatus);
                                                        onStopped();
                                                    }
                                                    mStatusView.setText(streamStatus.toString());
                                                }
                                            });
                                        }
                                    });
```

9. Close connection

Session.disconnect()code

```
mStartButton.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View view) {
        muteButton();
        if (mStartButton.getTag() == null || Integer.valueOf(R.string.action_start).equals(mStartButton.
getTag())) {
            ...
        } else {
            /**
             * Connection to WCS server is closed with method Session.disconnect().
             */
            session.disconnect();
        }
        ...
    }
});
```

10. Receiving the event confirming disconnection

session.onDisconnection()code

```
@Override
public void onDisconnection(final Connection connection) {
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            mStatusView.setText(connection.getStatus());
            mStatusView.setText(connection.getStatus());
            onStopped();
        }
    });
}
```

11. CustomCameraCapturerOptions preparation

code

```
    private CustomCameraCapturerOptions createCustomCameraCapturerOptions() {
        return new CustomCameraCapturerOptions() {

            private String cameraName;
            private CameraVideoCapturer.CameraEventsHandler eventsHandler;
            private boolean captureToTexture;

            @Override
            public Class<?>[] getCameraConstructorArgsTypes() {
                return new Class<?>[]{String.class, CameraVideoCapturer.CameraEventsHandler.class, boolean.
class};
            }

            @Override
            public Object[] getCameraConstructorArgs() {
                return new Object[]{cameraName, eventsHandler, captureToTexture};
            }

            @Override
            public void setCameraName(String cameraName) {
                this.cameraName = cameraName;
            }

            @Override
            public void setEventsHandler(CameraVideoCapturer.CameraEventsHandler eventsHandler) {
                this.eventsHandler = eventsHandler;
            }

            @Override
            public void setCaptureToTexture(boolean captureToTexture) {
                this.captureToTexture = captureToTexture;
            }

            @Override
            public String getCameraClassName() {
                return "org.webrtc.GPUImageCameraCapturer";
            }

            @Override
            public Class<?>[] getEnumeratorConstructorArgsTypes() {
                return new Class[0];
            }

            @Override
            public Object[] getEnumeratorConstructorArgs() {
                return new Object[0];
            }

            @Override
            public String getEnumeratorClassName() {
                return "org.webrtc.GPUImageCameraEnumerator";
            }
        };
    }
```

12. Custom camera choosing before publishing

code

```
        CameraCapturerFactory.getInstance().setCustomCameraCapturerOptions(createCustomCameraCapturerOptions());
        CameraCapturerFactory.getInstance().setCameraType(CameraCapturerFactory.CameraType.CUSTOM);
```

13. Enabling/disabling filter

code

```
        mUseFilter = (CheckBox) findViewById(R.id.use_filter);
        mUseFilter.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
                GPUImageCameraSession.setUsedFilter(isChecked);
            }
        });
        GPUImageCameraSession.setUsedFilter(mUseFilter.isChecked());
```

14. Camera1Capturer interface implementation

code

```
public class GPUImageCameraCapturer extends Camera1Capturer {

    private GPUImageCameraSession cameraSession;
    private boolean captureToTexture;

    public GPUImageCameraCapturer(String cameraName, CameraEventsHandler eventsHandler, boolean
captureToTexture) {
        super(cameraName, eventsHandler, captureToTexture);
        this.captureToTexture = captureToTexture;
    }

    @Override
    protected void createCameraSession(CameraSession.CreateSessionCallback createSessionCallback, CameraSession.
Events events, Context applicationContext, SurfaceTextureHelper surfaceTextureHelper, String cameraName, int
width, int height, int framerate) {
        CameraSession.CreateSessionCallback myCallback = new CameraSession.CreateSessionCallback() {
            @Override
            public void onDone(CameraSession cameraSession) {
                GPUImageCameraCapturer.this.cameraSession = (GPUImageCameraSession) cameraSession;
                createSessionCallback.onDone(cameraSession);
            }

            @Override
            public void onFailure(CameraSession.FailureType failureType, String s) {
                createSessionCallback.onFailure(failureType, s);
            }
        };

        GPUImageCameraSession.create(myCallback, events, captureToTexture, applicationContext,
surfaceTextureHelper, Camera1Enumerator.getCameraIndex(cameraName), width, height, framerate);
    }
}
```

15. Camera1Enumerator implementation

code

```
public class GPUImageCameraEnumerator extends Camera1Enumerator {
    @Override
    public CameraVideoCapturer createCapturer(String deviceName, CameraVideoCapturer.CameraEventsHandler
eventsHandler) {
        return new GPUImageCameraCapturer(deviceName, eventsHandler, true);
    }
}
```

16. GPUImage packages import

code

```
import jp.co.cyberagent.android.gpuimage.GPUImage;
import jp.co.cyberagent.android.gpuimage.GPUImageRenderer;
import jp.co.cyberagent.android.gpuimage.PixelBuffer;
import jp.co.cyberagent.android.gpuimage.filter.GPUImageMonochromeFilter;
import jp.co.cyberagent.android.gpuimage.util.Rotation;
```

17. GPUImageCameraSession object initializing

code

```
    public static void create(CreateSessionCallback callback, Events events, boolean captureToTexture, Context
applicationContext, SurfaceTextureHelper surfaceTextureHelper, int cameraId, int width, int height, int
framerate) {
        long constructionTimeNs = System.nanoTime();
        Logging.d("GPUImageCameraSession", "Open camera " + cameraId);
        events.onCameraOpening();

        Camera camera;
        try {
            camera = Camera.open(cameraId);

        } catch (RuntimeException var19) {
            callback.onFailure(FailureType.ERROR, var19.getMessage());
            return;
        }

        if (camera == null) {
            callback.onFailure(FailureType.ERROR, "android.hardware.Camera.open returned null for camera id = "
+ cameraId);
        } else {
            try {
                camera.setPreviewTexture(surfaceTextureHelper.getSurfaceTexture());
            } catch (RuntimeException | IOException var18) {
                camera.release();
                callback.onFailure(FailureType.ERROR, var18.getMessage());
                return;
            }

            Camera.CameraInfo info = new Camera.CameraInfo();
            Camera.getCameraInfo(cameraId, info);

            CameraEnumerationAndroid.CaptureFormat captureFormat;
            try {
                Camera.Parameters parameters = camera.getParameters();
                captureFormat = findClosestCaptureFormat(parameters, width, height, framerate);
                Size pictureSize = findClosestPictureSize(parameters, width, height);
                updateCameraParameters(camera, parameters, captureFormat, pictureSize, captureToTexture);
            } catch (RuntimeException var17) {
                camera.release();
                callback.onFailure(FailureType.ERROR, var17.getMessage());
                return;
            }

            if (!captureToTexture) {
                int frameSize = captureFormat.frameSize();

                for(int i = 0; i < 3; ++i) {
                    ByteBuffer buffer = ByteBuffer.allocateDirect(frameSize);
                    camera.addCallbackBuffer(buffer.array());
                }
            }

            camera.setDisplayOrientation(0);
            callback.onDone(new GPUImageCameraSession(events, captureToTexture, applicationContext,
surfaceTextureHelper, cameraId, camera, info, captureFormat, constructionTimeNs));
        }
    }
```

18. Filter usage switching

code

```
    public static void setUsedFilter(boolean usedFilter) {
        isUsedFilter = usedFilter;
    }
```

19. Applying the filter to the picture extracted from camera buffer, and returning data changed back to the buffer

code

```
    private void listenForBytebufferFrames() {
        this.camera.setPreviewCallbackWithBuffer(new Camera.PreviewCallback() {
            public void onPreviewFrame(byte[] data, Camera callbackCamera) {
                GPUImageCameraSession.this.checkIsOnCameraThread();
                if (callbackCamera != GPUImageCameraSession.this.camera) {
                    ...
                } else if (GPUImageCameraSession.this.state != GPUImageCameraSession.SessionState.RUNNING) {
                    ...
                } else {
                    long captureTimeNs = TimeUnit.MILLISECONDS.toNanos(SystemClock.elapsedRealtime());
                    if (!GPUImageCameraSession.this.firstFrameReported) {
                        int startTimeMs = (int)TimeUnit.NANOSECONDS.toMillis(System.nanoTime() -
GPUImageCameraSession.this.constructionTimeNs);
                        GPUImageCameraSession.camera1StartTimeMsHistogram.addSample(startTimeMs);
                        GPUImageCameraSession.this.firstFrameReported = true;
                    }

                    applyFilter(data, GPUImageCameraSession.this.captureFormat.width, GPUImageCameraSession.
this.captureFormat.height);

                    VideoFrame.Buffer frameBuffer = new NV21Buffer(data, GPUImageCameraSession.this.
captureFormat.width, GPUImageCameraSession.this.captureFormat.height, () -> {
                        GPUImageCameraSession.this.cameraThreadHandler.post(() -> {
                            if (GPUImageCameraSession.this.state == GPUImageCameraSession.SessionState.RUNNING)
{
                                GPUImageCameraSession.this.camera.addCallbackBuffer(data);
                            }

                        });
                    });
                    VideoFrame frame = new VideoFrame(frameBuffer, GPUImageCameraSession.this.
getFrameOrientation(), captureTimeNs);
                    GPUImageCameraSession.this.events.onFrameCaptured(GPUImageCameraSession.this, frame);
                    frame.release();
                }
            }
        });
    }
```

20. Filter implementation

code

```
    private void applyFilter(byte[] data, int width, int height) {
        if (!isUsedFilter) {
            return;
        }
        GPUImageMonochromeFilter filter = new GPUImageMonochromeFilter();
        filter.setColor(0,0,0);

        GPUImageRenderer renderer = new GPUImageRenderer(filter);
        renderer.setRotation(Rotation.NORMAL, false, false);
        renderer.setScaleType(GPUImage.ScaleType.CENTER_INSIDE);

        PixelBuffer buffer = new PixelBuffer(width, height);
        buffer.setRenderer(renderer);

        renderer.onPreviewFrame(data, width, height);
        Bitmap newBitmapRgb = buffer.getBitmap();
        byte[] dataYuv = Utils.getNV21(width, height, newBitmapRgb);
        System.arraycopy(dataYuv, 0, data, 0, dataYuv.length);

        filter.destroy();
        buffer.destroy();
    }
```