CameraX: Touch-to-focus API

Status: Draft
Created: 2019-6-10

Objective

Provides a simple API that developers can specify only an X, Y and the View, and then touch-to-focus just works like magic. The API should also be flexible that developers can specify advanced options.

Background

Most camera apps need to have touch-to-focus capability but it is difficult to implement using Camera2 API. It has the following challenges:

(1) **Translating the view coordinate into the sensor coordinates is hard:**
Device orientation / sensor orientation / front facing mirroring / crop region / sensor aspect ratio / view cropping need to be considered.

(2) **The Camera2 API flow is complicated and is not well documented.**
Developers need to set AF/AF/AWB regions, and then set CONTROL_AF_MODE to AUTO , and then trigger Set CONTROL_AF_TRIGGER to START and more. You can imagine how difficult it is to implement it correctly.

(3) Too many options
Developers are asked to specify the size of metering and focus area and the weights. Also, developers need to reset the areas when necessary. Most of the time they simply want the best default options.

The new API aims to handle all these challenges for developers.

Overview

This API will be exposed at Camera Control. Developer can get the CameraControl via

```
CameraX.getCameraControl(LensFacing);
```

The new API offers below advantages:

(1) No coordinate translation at Apps side.
Apps just pass X, Y and View , and that's all.

(2) **Best touch-to-focus flow which works on all camera2 devices.**
We use best practice touch-to-focus flow to implement it and have tests in our test lab to make sure it works on most devices.

(3) **Simple yet powerful**
Use the best default options for developers so that they don't have to specify too much details. Meanwhile they have the flexibility to set the options. Options like the metering area's width , the weight and when to cancel the touch-to-focus action. Apps can also register a callback to be notified when the focus is done.

Code snippet that demonstrates minimized codes required for touch-to-focus.

```
MeteringPointFactory factory = new TextureViewMeteringPointFactory(textureView);
MeteringPoint point = factory.createPoint(x, y);
MeteringAction action = MeteringAction.Builder.from(point).build();
CameraX.getCameraControl(lensFacing).startFocusAndMetering(action);
```

CameraX uses the builder pattern in order to both keep the API simple and allow more options available for developers. In the above code snippet, all it needs are just a textureView instance and x , y from the textureView.

CameraX also provides other options available like below.

```
MeteringAction action = MeteringAction.Builder.from(meteringPoint1)
            .addPoint(meteringPoint2)  // could have multiple addPoint call
            .setAreaSize(0.2f)  // 0~  1.0f
            .setWeight(1.0f)   // 0 ~1.0f
            .setMode(MeteringMode.AF_AE_AWB)
            .setAutoFocusCallback(new OnAutoFocusListener(){
                  public void onFocusCompleted(boolean isSuccess) {
                  }
            })
            // auto calling cancelFocusAndMetering in 5 sec.
            .setAutoCancelDuration(5, TimeUnit.Second)
            .build();
```

Detailed Design

Create the MeteringPoint : Factory for the different needs

CameraX supports multiple ways to create MeteringPoint. There are 3 kinds of factory to create a MeteringPoint corresponding to 3 scenarios respectively:

(1) Preview using TextureView

```
// TextureViewMeteringPointFactory is in View module
MeteringPointFactory factory = new TextureViewMeteringPointFactory(textureView);
MeteringPoint point = factory.createPoint(x, y);
```

For TextureView (which most basic camera apps use), no coordinates translation is required at app side. A TextureView, X and Y are all it needs. **It is not only the simplest but also the most complete form** because it takes into account the conditions that SurfaceTexture could be cropped/scaled/rotated inside TextureView via #setTransform. **Basically it can be translated perfectly WYSIWYG to the sensor coordinates without any efforts from apps**.

Active Preview UseCase is chosen for the FOV and final sensor coordinates will be adjusted by the aspect ratio in this FOV. FOV is required for the final conversion because the sensor FOV aspect ratio could mismatch the chosen FOV.

Note: TextureViewMeteringPointFactory is put in the CameraX view module.

(2) Preview using GLSurfaceView or SurfaceView. (SurfaceView not supported in CameraX yet)

```
// For GLSurfaceView or SurfaceView, same XY orientation as the View(and display
orientation).
MeteringPointFactory factory = new DisplayOrientedMeteringPointFactory(viewWidth,
viewHeight);
MeteringPoint point = factory.createPoint(x, y);
```

DisplayOrientedMeteringPointFactory is used for translating view x/y in display orientation. View's width and height is passed to create the factory. It works well to create a point by x, y if the camera preview fills the whole view without any cropping or rotating. CameraX is responsible for taking care of the device orientation, mirroring, sensor orientation to map it to sensor coordinates.

if preview is cropped or rotated, it is apps' duty to correctly transform x, y and viewWidth , viewHeight into the correct value.

Active Preview UseCase is chosen for the FOV and final sensor coordinates will be adjusted by the aspect ratio in this FOV.

(3) Sensor coordinates / ImageAnalysis

```
// XY has the same orientation as the sensor.
MeteringPointFactory factory = new SensorOrientedMeteringPointFactory(width,
height);
MeteringPoint point = factory.createPoint(x, y);
```

There are two scenarios for this:
#1 Apps already translated the point to sensor coordinates , they want to pass in the sensor coordinates directly to our touch focus API. In this case, app can call it like

```
MeteringPointFactory factory = new SensorOrientedMeteringPointFactory(sensorWidth,
sensorHeight);
MeteringPoint point = factory.createPoint(sensorX, sensorY);
```

If they are using normalized coordinates like 0 to 1, they can use 1.0 as the width / height.

#2 Apps use ImageAnalysis to detect something (like people, faces, etc.) and set focus to that objects.

```
MeteringPointFactory factory = new SensorOrientedMeteringPointFactory(imageWidth,
imageHeight, imageAnalysis);
MeteringPoint point = factory.createPoint(imageX, imageY);
```

Trigger Touch-To-Focus action

After action is built, pass it to below method to start the touch-to-focus action.

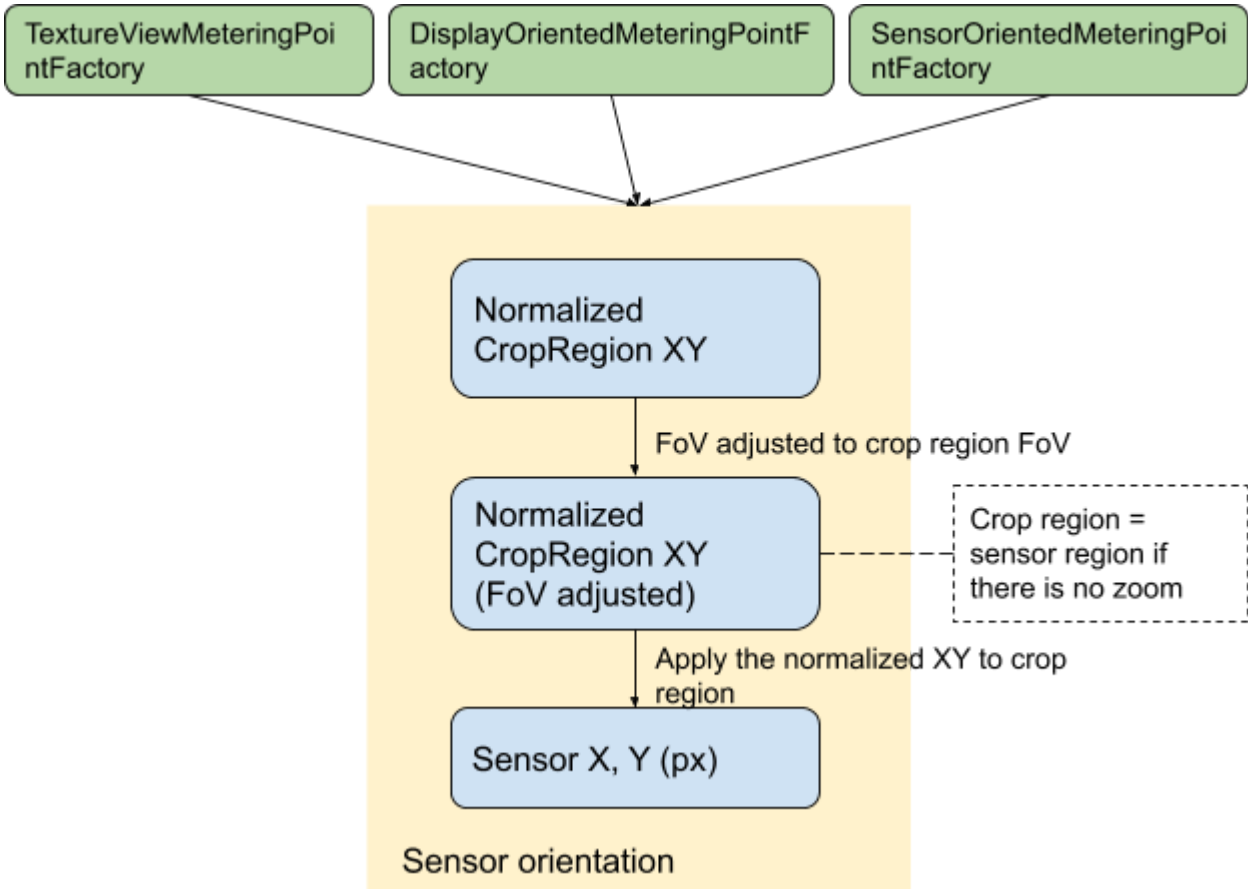
```
getCameraControl(lensFacing).startFocusAndMetering(action);
```

Apps can call below method to cancel the metering actions.

```
getCameraControl(lensFacing).cancelFocusAndMetering();
```

Coordinates translation

This section describes the details regarding how CameraX translates the coordinates to the final sensor coordinates.



The Above diagram illustrates the main flow of the coordinates translation. The 3 MeteringPointFactory we introduced in previous section will all convert to the **normalized XY in crop region** and then follow the same procedure to be translated to the final sensor coordinates. Let's explain some steps below:

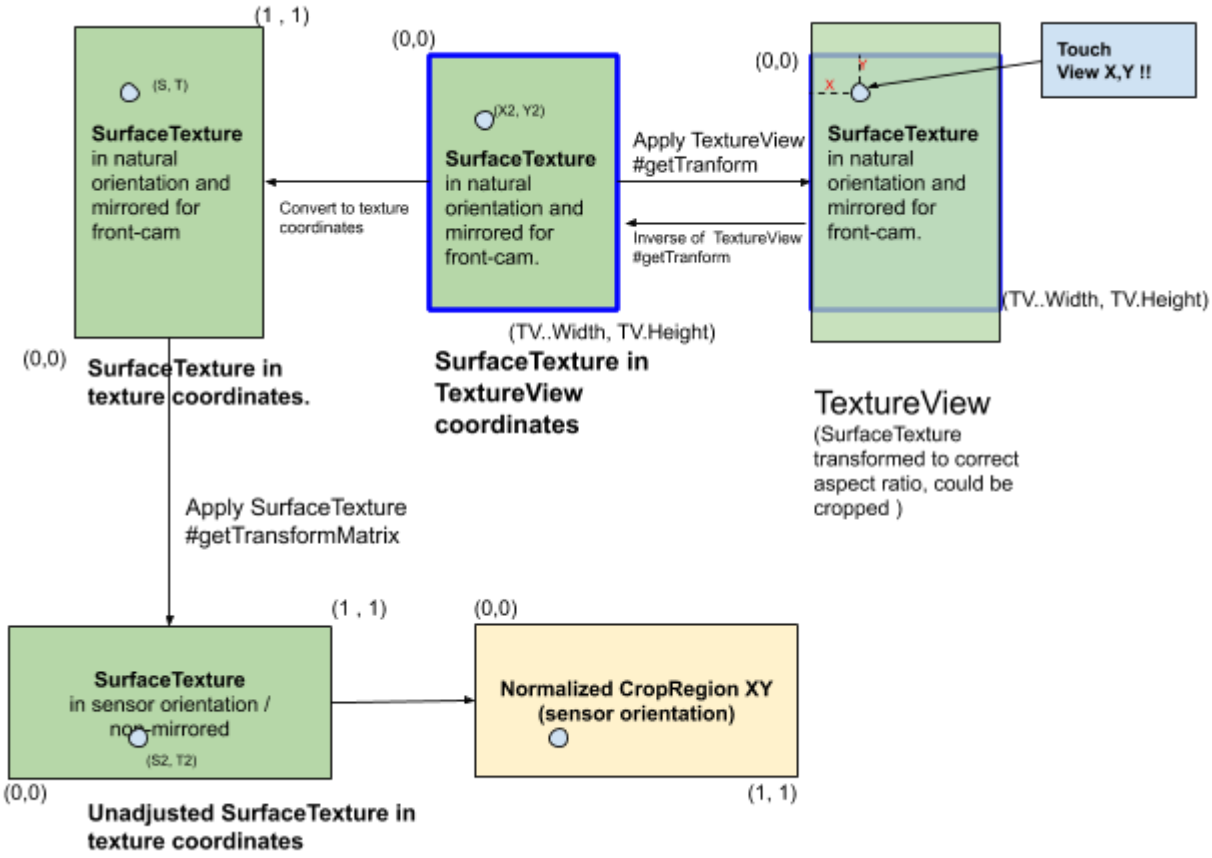
TextureViewMeteringPointFactory: TextureView X, Y

See below diagram for the transform flow.
TextureView#getTransform: used to transform the SurfaceTexture inside TextureView.
If not set, the SurfaceTexture is scaled to fill the Textureview(Aspect ratio could be wrong).
CameraX uses the inverse of this transform to convert a (X, Y) in TextureView to the (X2, Y2) in the natural orientation SurfaceTexture (whose width and height is equal to the TextureView), We can then convert the (X2,Y2) to the texture coordinates (S,T).

SurfaceTexture#getTransformMatrix: used to translate the texture coordinates to the correct texture coordinates in source. Camera streamed SurfaceTexture will set this transform so that the SurfaceTexture can be displayed correctly in natural orientation.

CameraX utilizes this transform matrix to convert the (S, T) in previous stage to the (S2, T2) , the texture coordinates in unadjusted SurfaceTexture. Finally, we adjust the coordinates to the normalized crop region XY which the left-top is (0,0) and the right-bottom is (1,1).

Note: The SurfaceTexture can be retrieved from active Preview UseCase.

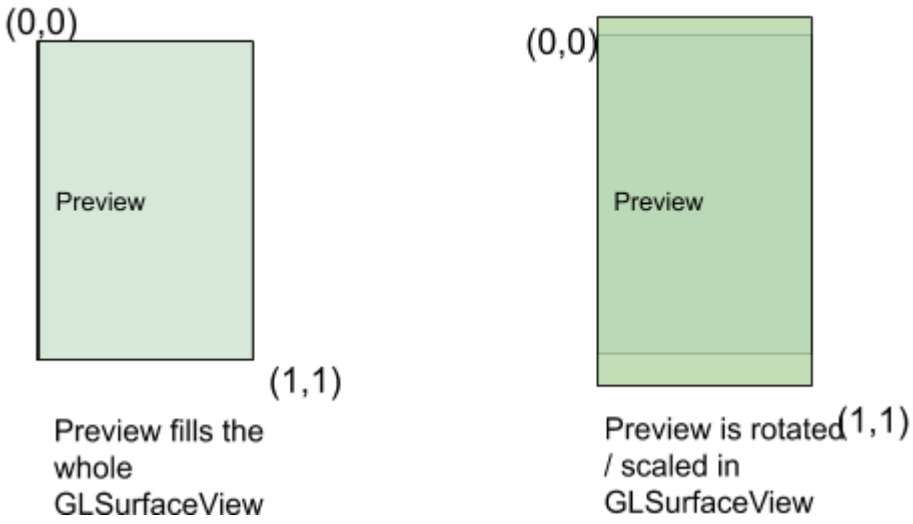


Sample codes are [here]

DisplayOrientedMeteringPointFactory: Display oriented XY

This factory creates the point by the view x, y which is in display orientation. If x , y is view's x, y, then you should set the width / height of the factory to be the view width / view height. If you are using normalized coordinate like 0 to 1, the width / height of factory should be 1.0f.

if preview is cropped or rotated, it is apps' duty to correctly transform x, y and viewWidth , viewHeight into the correct value.



CameraX translates the normalized view points to the normalized crop region XY by using the display orientation , the sensor orientation and front camera mirroring. The reason CameraX uses display orientation instead of the rotation degree returned in Preview#OnPreviewOutputUpdateListener is because the View XY has the same orientation as the display orientation.

SensorOrientedMeteringPointFactory: Sensor oriented XY

It is similar as DisplayOrientedMeteringPointFactory except that it is in sensor orientation. CameraX assumes it is already rotated / mirrored correctly and just map it into the final sensor coordinates.

Normalized Crop Region XY FOV adjustment

We follow https://source.android.com/devices/camera/camera3_crop_reprocess to adjust the normalized coordinates by the current FOV compared with crop region FOV.

What if the View itself is covered by other Views or the boundary?

In this case, the app should get the X, Y from the View itself (GLSurfaceView or TextureView) instead of getting X, Y in the container which contains the View. This will eliminate the need for extra XY conversion.

Options

All 3 builder creation methods shared the same options. Below are all the options exposed in MeteringAction.Builder:

```
// To add extra metering point. If device doesn't support multiple region
// , this will be a no-ops.
MeteringAction.Builder addPoint(metringPoint);

// To decides which metering region should be enabled. If AF is not enabled, AUTO
// FOCUS will not be triggered.
MeteringAction.Builder setMode(MeteringMode mode);
enum MeteringMode {
   AF_AE_AWB,  // default
   AF_AE,
   AE_AWB,
   AF_AWB,
   AF_ONLY,
   AE_ONLY,
   AWB_ONLY
}

// Sets the auto focus listeners. onFocusCompleted() is called when the focus is
done.
MeteringAction.Builder setAutoFocusCallback(OnAutoFocusListener listener);
interface OnAutoFocusListener {
   void onFocusCompleted(boolean isFocused);
}
// Sets the metering width and height, value is ranged from 0(0%) to 1.0 (100%) ,
// the default/minimal value is 0.1 (10%) , value that is below 0.1 will be set as
// 0.1.
MeteringAction.Builder setAreaSize(float size);

// Sets the metering region weight, ranged from 0 to 1.  The default value is 1
// (max).
MeteringAction.Builder setWeight(float weight);

// Sets if CameraX should call cancelFocusAndMetering() in the given duration.
// By default a 5 second auto cancel duration is set.
MeteringAction.Builder setAutoCancelDuration(long duration, TimeUnit timeUnit);

// Disable auto clear .
MeteringAction.Builder disableAutoCancel();
```

Touch-to-focus flow using camera2

1. User taps an X, Y in view.
2. App translates the view coordinates to coordinates on the sensor active array (see coordinates translation)

3. App calculates a rectangle that surrounds that point to use as a metering region. Use 10% of the field of view by default.
4. App constructs a new capture request with:
    1. Set the CONTROL_AF_REGIONS, CONTROL_AE_REGIONS, and CONTROL_AWB_REGIONS to that rectangle with weights (some regions could be disabled by the options).
    2. Set CONTROL_AF_MODE to AUTO
5. The app sets this request as the repeating request, and also submits a single request with the same settings plus
    1. Set CONTROL_AF_TRIGGER to START
6. With those settings, the camera device will trigger an autofocus sweep with the selected region, along with re-metering for AE and AWB as well.
7. If AutoClear is enabled , CameraX will reset the region and cancel the CONTROL_AF_TRIGGER and set CONTROL_AF_MODE back to CONTINOUS_AF (if supported) , otherwise it will clear the action when camera is closed or app explicitly call `clearMeteringArea`();

Revision History

version	**date**	**description**
1.0	2019/6/10	Initial version
2.0	2019/6/19	● Add MeteringPointFactory interface and 3 implementations ○ TextureViewMeteringPointFactory(View module) ○ DisplayOrientedMeteringPointFactory(Core) ○ SensorOrientedMeteringPointFactory (Core) ● To remove View dependency in Core. TextureViewMeteringPointFactory is put in View module. ● The builder now requires only a MeteringPoint to be constructed. One builder creation method only but there are 3 types of MeteringPointFactory.