package com.android.camera.ui;

import android.content.Context;

import android.graphics.Bitmap;

import android.graphics.Canvas;

import android.graphics.Matrix;

import android.graphics.Paint;

import android.graphics.Rect;

import android.util.AttributeSet;

import android.util.Log;

import android.view.KeyEvent;

import android.view.MotionEvent;

import android.view.View;

import android.widget.FrameLayout;

import com.android.camera.Util;

public class DragLayer extends FrameLayout{

/\*\*

\* Utility rectangle

\*/

private Rect mDragRect = new Rect();

/\*\*

\* X offset from where we touched on the cell to its upper-left corner

\*/

private float mTouchOffsetX;

/\*\*

\* Y offset from where we touched on the cell to its upper-left corner

\*/

private float mTouchOffsetY;

private float mLastMotionX;

private float mLastMotionY;

/\*\*

\* The bitmap that is currently being dragged

\*/

private Bitmap mDragBitmap = null;

// Number of pixels to add to the dragged item for scaling

private static final float DRAG\_SCALE = 0.0f;

private int mBitmapOffsetX;

private int mBitmapOffsetY;

private Paint mDragPaint;

private View mOriginator;

private boolean mDragging;

/\*\*

\* What's purpose of it? CXF???

\*/

private final Rect mRect = new Rect();

private DragInterface mDragger;

int mWidthPx;

int mHeightPx;

int mTop;

private float mX0;

private float mX1;

//private boolean mSingleTapState;

private static final int DRAG\_ACTION\_MOVE = 7;

private static final int DRAG\_ACTION\_UP = 6;

private static final int DRAG\_ACTION\_DOWN = 5;

private boolean mCanBack;

private float mChangeMotionX;

private boolean mPreview4\_3 = true;

private int mDragType;

//public void setPreview4\_3(boolean is4\_3) {

// mPreview4\_3 = is4\_3;

//}

public DragLayer(Context context) {

super(context);

mDragging = false;

}

public DragLayer (Context context, AttributeSet attr){

super(context , attr);

mDragging = false;

}

public void startDrag(View v, DragInterface dragger, int dispWidth, int dispHeight, int top, boolean is4\_3, int type) {

mPreview4\_3 = is4\_3;

mDragType = type;

mWidthPx = dispWidth;

mHeightPx = dispHeight;

mTop = top;

mDragger = dragger;

//mDragger.onDragRect(mLeftRect, mRightRect, mUpRect, mDownRect);

Rect r = mDragRect;

r.set(v.getScrollX(), v.getScrollY(), 0, 0);

offsetDescendantRectToMyCoords(v, r);

mTouchOffsetX = mLastMotionX - r.left;

mTouchOffsetY = mLastMotionY - r.top;

v.clearFocus();

v.setPressed(false);

boolean willNotCache = v.willNotCacheDrawing();

v.setWillNotCacheDrawing(false);

// Reset the drawing cache background color to fully transparent

// for the duration of this operation

int color = v.getDrawingCacheBackgroundColor();

v.setDrawingCacheBackgroundColor(0);

if (color != 0) {

v.destroyDrawingCache();

}

v.buildDrawingCache();

Bitmap viewBitmap = v.getDrawingCache();

int width = viewBitmap.getWidth();

int height = viewBitmap.getHeight();

Matrix scale = new Matrix();

float scaleFactor = v.getWidth();

scaleFactor = (scaleFactor + DRAG\_SCALE) /scaleFactor;

scale.setScale(scaleFactor, scaleFactor);

mDragBitmap = Bitmap.createBitmap(viewBitmap, 0, 0, width, height, scale, true);

v.destroyDrawingCache();

v.setWillNotCacheDrawing(willNotCache);

v.setDrawingCacheBackgroundColor(color);

final Bitmap dragBitmap = mDragBitmap;

mBitmapOffsetX = (dragBitmap.getWidth() - width) / 2;

mBitmapOffsetY = (dragBitmap.getHeight() - height) / 2;

v.clearAnimation();

v.setVisibility(GONE);

mDragPaint = null;

mDragging = true;

mOriginator = v;

// CXF: 2009DEC14 for SDK build

// if (null != mVibrator){

// mVibrator.vibrate(VIBRATE\_DURATION);

// }

invalidate();

}

@Override

protected void dispatchDraw(Canvas canvas) {

super.dispatchDraw(canvas);

/\*if (mDragging && mDragBitmap != null) {

//canvas.drawBitmap(mDragBitmap,getScrollX() + 50, getScrollY() + 50, mDragPaint);

float tmp = getScrollX() + mLastMotionX - mTouchOffsetX - mBitmapOffsetX;

if (tmp < 0) {

canvas.drawBitmap(mDragBitmap, 0, mTop, mDragPaint);

mX0 = mDragBitmap.getWidth()/2;

//mX0 = mDragBitmap.getWidth() -mWidthPx/2;

} else if (tmp > mWidthPx - mDragBitmap.getWidth()) {

canvas.drawBitmap(mDragBitmap, mWidthPx - mDragBitmap.getWidth(), mTop, mDragPaint);

mX0 = mWidthPx - mDragBitmap.getWidth()/2;

//mX0 = mWidthPx + mWidthPx/2 - mDragBitmap.getWidth();

} else {

canvas.drawBitmap(mDragBitmap, tmp, mTop, mDragPaint);

if (tmp > mWidthPx/2 && tmp < mWidthPx-mDragBitmap.getWidth()/2) {

} else if (tmp < mWidthPx/2 && tmp > mDragBitmap.getWidth()/2) {

//mX0 = mDragBitmap.getWidth() -mWidthPx/2 + (mWidthPx/2-(getScrollX() + mLastMotionX - mTouchOffsetX - mBitmapOffsetX))\*2;

} else {

//mX0 = mWidthPx/2;

}

mX0 = tmp + mDragBitmap.getWidth()/2;

}

}\*/

if (mDragging && mDragBitmap != null) {

float tmpX = getScrollX() + mLastMotionX - mTouchOffsetX - mBitmapOffsetX;

float tmpY = getScrollY() + mLastMotionY - mTouchOffsetY - mBitmapOffsetY;

if(tmpX < 0) {

tmpX = 0;

}

if(tmpX > mWidthPx - mDragBitmap.getWidth()) {

tmpX = mWidthPx - mDragBitmap.getWidth();

}

float top = 0f;

float bottom = 0.f;

if(mPreview4\_3) {

top = Util.PREVIEW\_4\_3\_TOP\_OFFSET;

bottom = Util.PREVIEW\_4\_3\_BOTTOM\_OFFSET;

}

if(tmpY < top) {

tmpY = top;

}

if(tmpY > mHeightPx - mDragBitmap.getHeight()-bottom) {

tmpY = mHeightPx - mDragBitmap.getHeight()-bottom;

}

//Log.d("zzpDrag", "draw ACTION\_MOVE mHeightPx = "+mHeightPx+", mWidthPx = "+mWidthPx+", tmpX = "+tmpX+", tmpY = "+tmpY+", Width = "+mDragBitmap.getWidth()+", height = "+mDragBitmap.getHeight());

mDragger.onDragMove((int)(tmpX+mDragBitmap.getWidth()/2), (int)(tmpY+mDragBitmap.getHeight()/2), mPreview4\_3, mDragType);//, mWidthPx, mHeightPx);

canvas.drawBitmap(mDragBitmap,

tmpX,//getScrollX() + mLastMotionX - mTouchOffsetX - mBitmapOffsetX,

tmpY,//getScrollY() + mLastMotionY - mTouchOffsetY - mBitmapOffsetY,

mDragPaint);

}

}

@Override

public boolean dispatchKeyEvent(KeyEvent event) {

return mDragging || super.dispatchKeyEvent(event);

}

@Override

public boolean onInterceptTouchEvent(MotionEvent ev) {

final int action = ev.getAction();

final float x = ev.getX(0);

final float y = ev.getY();

switch (action) {

case MotionEvent.ACTION\_MOVE:

//Log.d("zzpDrag", "onInterceptTouchEvent ACTION\_MOVE x = "+x+", y = "+y);

//mCanBack = true;

break;

case MotionEvent.ACTION\_DOWN:

mCanBack = false;

// Remember location of down touch

mLastMotionX = x;

mLastMotionY = y;

mChangeMotionX = x;

break;

case MotionEvent.ACTION\_CANCEL:

case MotionEvent.ACTION\_UP:

//Log.d("zzp", "mSingleTapState3 = " + mSingleTapState);

if(action == MotionEvent.ACTION\_CANCEL)

Log.d("zzpCancel", "MotionEvent.ACTION\_CANCEL 2222");

endDrag(x,y);

break;

}

return mDragging;

}

private void endDrag(float x, float y) {

if (mDragging) {

mDragging = false;

if (mDragBitmap != null) {

mDragBitmap.recycle();

}

if (mOriginator != null) {

mOriginator.setVisibility(VISIBLE);

}

mDragger.onDragEnd((int)x, (int)y, mPreview4\_3, mDragType);

//Log.d("zzp", "get2DStateDrag is " + mLauncher.get2DStateDrag()+", mCanBack = "+mCanBack);

//Log.d("CXF", "x is " + (long)x + "y is " + (long)y);

}

}

@Override

public boolean onTouchEvent(MotionEvent ev) {

if (!mDragging) {

return false;

}

final int action = ev.getAction();

final float x = ev.getX(0);

final float y = ev.getY();

switch (action) {

case MotionEvent.ACTION\_DOWN:

// Remember where the motion event started

mLastMotionX = x;

mLastMotionY = y;

mChangeMotionX = x;

mX0 = mWidthPx / 2;

//mLauncher.m3dView.mRenderer.stopRotate();

/\*if (mLauncher.get3Dstate()) {

mSingleTapState = true;

Log.d("zzp", "mSingleTapState4 = " + mSingleTapState);

return true;

}

Log.d("zzp", "mSingleTapState5 = " + mSingleTapState);\*/

break;

case MotionEvent.ACTION\_MOVE:

// CXF: 2009DEC14 for SDK build

{

float tmpX = getScrollX() + x - mTouchOffsetX - mBitmapOffsetX;

float tmpY = getScrollY() + y - mTouchOffsetY - mBitmapOffsetY;

if(tmpX < 0) {

tmpX = 0;

}

if(tmpX > mWidthPx - mDragBitmap.getWidth()) {

tmpX = mWidthPx - mDragBitmap.getWidth();

}

float top = 0f;

float bottom = 0.f;

if(mPreview4\_3) {

top = Util.PREVIEW\_4\_3\_TOP\_OFFSET;

bottom = Util.PREVIEW\_4\_3\_BOTTOM\_OFFSET;

}

if(tmpY < top) {

tmpY = top;

}

if(tmpY > mHeightPx - mDragBitmap.getHeight()-bottom) {

tmpY = mHeightPx - mDragBitmap.getHeight()-bottom;

}

final Rect rect = mRect;

// Invalidate current icon position

//rect.union(left - 1, top - 1, left + width + 1, top + height + 1);

//rect.union(0, 0, mWidthPx, mHeightPx);

rect.union((int)tmpX, (int)tmpY, (int)(tmpX+180), (int)(tmpY+180));

//invalidate();

invalidate(rect);

//Log.d("zzpDrag", "onTouchEvent ACTION\_MOVE x = "+x+", y = "+y);

mLastMotionX = x;

mLastMotionY = y;

}

break;

case MotionEvent.ACTION\_UP:

case MotionEvent.ACTION\_CANCEL:

{

if(action == MotionEvent.ACTION\_CANCEL)

Log.d("zzpCancel", "MotionEvent.ACTION\_CANCEL 111");

float tmpX = getScrollX() + x - mTouchOffsetX - mBitmapOffsetX;

float tmpY = getScrollY() + y - mTouchOffsetY - mBitmapOffsetY;

if(tmpX < 0) {

tmpX = 0;

}

if(tmpX > mWidthPx - mDragBitmap.getWidth()) {

tmpX = mWidthPx - mDragBitmap.getWidth();

}

float top = 0f;

float bottom = 0.f;

if(mPreview4\_3) {

top = Util.PREVIEW\_4\_3\_TOP\_OFFSET;

bottom = Util.PREVIEW\_4\_3\_BOTTOM\_OFFSET;

}

if(tmpY < top) {

tmpY = top;

}

if(tmpY > mHeightPx - mDragBitmap.getHeight()-bottom) {

tmpY = mHeightPx - mDragBitmap.getHeight()-bottom;

}

endDrag((tmpX+mDragBitmap.getWidth()/2), (tmpY+mDragBitmap.getHeight()/2));//(x,y);

}

break;

}

return true;

}

public interface DragInterface {

void onDragEnd(int x, int y, boolean is4\_3, int type);

void onDragMove(int x, int y, boolean is4\_3, int type);//, int w, int h);

}

}

package com.android.camera.ui;

public interface FocusIndicator {

void showStart();

void showSuccess(boolean timeout);

void showFail(boolean timeout);

void clear();

}

package com.android.camera.ui;

import android.content.Context;

import android.graphics.Region;

import android.util.AttributeSet;

import android.view.View;

import com.android.camera.Log;

import com.android.camera.R;

//add by zzp

import com.android.camera.Camera;

import com.android.camera.manager.ModePicker;

// A view that indicates the focus area or the metering area.

public class FocusIndicatorRotateLayout extends RotateLayout implements FocusIndicator {

private static final String TAG = "FocusLayout";

// Sometimes continuous autofucus starts and stops several times quickly.

// These states are used to make sure the animation is run for at least some

// time.

private int mState;

private static final int STATE\_IDLE = 0;

private static final int STATE\_FOCUSING = 1;

private static final int STATE\_FINISHING = 2;

private Runnable mDisappear = new Disappear();

private Runnable mEndAction = new EndAction();

private static final int SCALING\_UP\_TIME = 1000;

private static final int SCALING\_DOWN\_TIME = 200;

private static final int DISAPPEAR\_TIMEOUT = 200;

private Camera mContext; //add by zzp

public FocusIndicatorRotateLayout(Context context, AttributeSet attrs) {

super(context, attrs);

if(context instanceof Camera) { //add by zzp

mContext = (Camera) context;

}

}

public void setDrawable(int resid) { //modi by zzp

setVisibility(View.VISIBLE);

mChild.setBackgroundDrawable(getResources().getDrawable(resid));

}

@Override

public void showStart() {

Log.d(TAG, "showStart mState = " + mState);

//if(mContext.getCurrentMode() == ModePicker.MODE\_PROFESSIONAL) {

// setAlpha(0.5f);

// setDrawable(R.drawable.ic\_focus\_pro\_focused);

//}

if (mState == STATE\_IDLE) {

//modi by zzp start

if(mContext.getCurrentMode() != ModePicker.MODE\_PROFESSIONAL) {

setDrawable(R.drawable.ic\_focus\_focusing);

animate().withLayer().setDuration(SCALING\_UP\_TIME)

.scaleX(1.5f).scaleY(1.5f);

} else {

//setAlpha(0.5f);

setDrawable(R.drawable.ic\_focus\_pro\_focused);

}

//modi by zzp end

mState = STATE\_FOCUSING;

}

}

@Override

public void showSuccess(boolean timeout) {

//modi by zzp

Log.i(TAG, "showSuccess mState = " + mState);

if(mContext.getCurrentMode() == ModePicker.MODE\_PROFESSIONAL) {

setDrawable(R.drawable.ic\_focus\_pro\_focused);

//setScaleX(1f);

//setScaleY(1f);

animate().withLayer().setDuration(SCALING\_DOWN\_TIME).alpha(1f).scaleX(1f).scaleY(1f);

mState = STATE\_FINISHING;

}

else if (mState == STATE\_FOCUSING) {

setDrawable(R.drawable.ic\_focus\_focused);

animate().withLayer().setDuration(SCALING\_DOWN\_TIME).scaleX(1f)

.scaleY(1f).withEndAction(timeout ? mEndAction : null);

mState = STATE\_FINISHING;

}

}

@Override

public void showFail(boolean timeout) {

//modi by zzp

if(mContext.getCurrentMode() == ModePicker.MODE\_PROFESSIONAL) {

setDrawable(R.drawable.ic\_focus\_pro\_failed);

setScaleX(1f);

setScaleY(1f);

setAlpha(1f);

mState = STATE\_FINISHING;

}

else if (mState == STATE\_FOCUSING) {

setDrawable(R.drawable.ic\_focus\_failed);

animate().withLayer().setDuration(SCALING\_DOWN\_TIME).scaleX(1f)

.scaleY(1f).withEndAction(timeout ? mEndAction : null);

mState = STATE\_FINISHING;

}

}

@Override

public void clear() {

Log.d(TAG, "clear mState = " + mState);

//modi by zzp

if(mContext.getCurrentMode() != ModePicker.MODE\_PROFESSIONAL) {

animate().cancel();

removeCallbacks(mDisappear);

mDisappear.run();

setScaleX(1f);

setScaleY(1f);

}

}

//add by zzp

public void clearForFace() {

animate().cancel();

removeCallbacks(mDisappear);

mDisappear.run();

setScaleX(1f);

setScaleY(1f);

}

//add by zzp

public void cancelAnimation() {

animate().cancel();

removeCallbacks(mDisappear);

setScaleX(1f);

setScaleY(1f);

}

private class EndAction implements Runnable {

@Override

public void run() {

// Keep the focus indicator for some time.

postDelayed(mDisappear, DISAPPEAR\_TIMEOUT);

}

}

private class Disappear implements Runnable {

@Override

public void run() {

Log.d(TAG, "Disappear run mState = " + mState);

mChild.setBackgroundDrawable(null);

setVisibility(View.GONE);

mState = STATE\_IDLE;

}

}

public boolean isFocusing() {

return mState != STATE\_IDLE;

}

@Override

public boolean gatherTransparentRegion(Region region) {

Log.i("faceView", "gatherTransparentRegion = " + region);

if (region !=null) {

final int[] location = new int[2];

int width = getWidth();

int height = getHeight();

getLocationInWindow(location);

int l = location[0] + width / 2 - width;

int t = location[1] + height / 2 - height;

int r = l + width \* 2;

int b = t + height \* 2;

region.op(l, t, r, b, Region.Op.DIFFERENCE);

}

return true;

}

}

package com.android.camera.ui;

import android.content.Context;

import android.graphics.Region;

import android.util.AttributeSet;

import android.view.View;

import com.android.camera.Log;

import com.android.camera.R;

import com.android.camera.Camera;

import com.android.camera.manager.ModePicker;

// A view that indicates the focus area or the metering area.

public class MeteringIndicatorRotateLayout extends RotateLayout implements FocusIndicator {

private static final String TAG = "MeteringLayout";

// Sometimes continuous autofucus starts and stops several times quickly.

// These states are used to make sure the animation is run for at least some

// time.

private int mState;

private static final int STATE\_IDLE = 0;

private static final int STATE\_FOCUSING = 1;

private static final int STATE\_FINISHING = 2;

private Runnable mDisappear = new Disappear();

private Runnable mEndAction = new EndAction();

private static final int SCALING\_UP\_TIME = 1000;

private static final int SCALING\_DOWN\_TIME = 200;

private static final int DISAPPEAR\_TIMEOUT = 200;

private Camera mContext;

public MeteringIndicatorRotateLayout(Context context, AttributeSet attrs) {

super(context, attrs);

if(context instanceof Camera) {

mContext = (Camera) context;

}

}

private void setDrawable(int resid) {

setVisibility(View.VISIBLE);

mChild.setBackgroundDrawable(getResources().getDrawable(resid));

}

@Override

public void showStart() {

Log.d(TAG, "showStart mState = " + mState);

if (mState == STATE\_IDLE) {

setDrawable(R.drawable.ic\_metering\_focused);

//animate().withLayer().setDuration(SCALING\_UP\_TIME)

// .scaleX(1.5f).scaleY(1.5f);

mState = STATE\_FOCUSING;

}

}

@Override

public void showSuccess(boolean timeout) {

if (mState == STATE\_FOCUSING) {

setDrawable(R.drawable.ic\_metering\_focused);

//animate().withLayer().setDuration(SCALING\_DOWN\_TIME).scaleX(1f)

// .scaleY(1f).withEndAction(timeout ? mEndAction : null);

setScaleX(1f);

setScaleY(1f);

mState = STATE\_FINISHING;

}

}

@Override

public void showFail(boolean timeout) {

if (mState == STATE\_FOCUSING) {

setDrawable(R.drawable.ic\_metering\_focused);

//animate().withLayer().setDuration(SCALING\_DOWN\_TIME).scaleX(1f)

// .scaleY(1f).withEndAction(timeout ? mEndAction : null);

setScaleX(1f);

setScaleY(1f);

mState = STATE\_FINISHING;

}

}

@Override

public void clear() {

Log.d(TAG, "clear mState = " + mState);

if(mContext.getCurrentMode() != ModePicker.MODE\_PROFESSIONAL) {

animate().cancel();

removeCallbacks(mDisappear);

mDisappear.run();

setScaleX(1f);

setScaleY(1f);

}

}

private class EndAction implements Runnable {

@Override

public void run() {

// Keep the focus indicator for some time.

postDelayed(mDisappear, DISAPPEAR\_TIMEOUT);

}

}

private class Disappear implements Runnable {

@Override

public void run() {

Log.d(TAG, "Disappear run mState = " + mState);

mChild.setBackgroundDrawable(null);

setVisibility(View.GONE);

mState = STATE\_IDLE;

}

}

public boolean isFocusing() {

return mState != STATE\_IDLE;

}

@Override

public boolean gatherTransparentRegion(Region region) {

Log.i("faceView", "gatherTransparentRegion = " + region);

if (region !=null) {

final int[] location = new int[2];

int width = getWidth();

int height = getHeight();

getLocationInWindow(location);

int l = location[0] + width / 2 - width;

int t = location[1] + height / 2 - height;

int r = l + width \* 2;

int b = t + height \* 2;

region.op(l, t, r, b, Region.Op.DIFFERENCE);

}

return true;

}

}

package com.android.camera;

import android.app.Service;

import android.content.BroadcastReceiver;

import android.content.Context;

import android.content.Intent;

import android.content.IntentFilter;

import android.content.SharedPreferences;

import android.content.pm.ActivityInfo;

import android.content.res.Configuration;

import android.graphics.SurfaceTexture;

import android.hardware.Camera.CameraInfo;

import android.hardware.Camera.Parameters;

import android.hardware.Camera.Size;

import android.location.Location;

import android.media.CamcorderProfile;

import android.net.Uri;

import android.os.AsyncTask;

import android.os.Bundle;

import android.os.ConditionVariable;

import android.os.Handler;

import android.os.Looper;

import android.os.Message;

import android.os.MessageQueue;

import android.os.PowerManager;

import android.os.Vibrator;

import android.os.SystemClock;

import android.os.storage.StorageVolume;

import android.provider.MediaStore;

import android.provider.Settings;

import android.view.KeyEvent;

import android.view.LayoutInflater;

import android.view.OrientationEventListener;

import android.view.View;

import android.view.ViewGroup;

import android.view.WindowManager;

import android.hardware.Camera.PreviewCallback;

import com.android.camera.actor.AsdActor;

import com.android.camera.actor.CameraActor;

import com.android.camera.actor.NightShotActor; //bobsong added

import com.android.camera.actor.FaceBeautyActor;

//import com.android.camera.actor.GestureActor;

import com.android.camera.actor.HdrActor;

import com.android.camera.actor.MavActor;

//import com.android.camera.actor.MotionTrackActor;

import com.android.camera.actor.PanoramaActor;

import com.android.camera.actor.PhotoActor;

//import com.android.camera.actor.SingleStereoPhotoActor;

import com.android.camera.actor.SmileActor;

import com.android.camera.actor.VideoActor;

import com.android.camera.actor.VideoLivePhotoActor;

import com.android.camera.actor.ProfessionalActor;

import com.android.camera.manager.FrameManager;

import com.android.camera.manager.IndicatorManager;

import com.android.camera.manager.InfoManager;

import com.android.camera.manager.TouchlessGuideManager;

import com.android.camera.manager.MMProfileManager;

import com.android.camera.manager.ModePicker;

//import com.android.camera.manager.MotionTrackViewManager;

import com.android.camera.manager.OnScreenHint;

import com.android.camera.manager.PickerManager;

//import com.android.camera.manager.RemainingManager;

import com.android.camera.manager.ReviewManager;

import com.android.camera.manager.RotateDialog;

import com.android.camera.manager.RotateProgress;

import com.android.camera.manager.SelfTimerManager;

import com.android.camera.manager.SettingManager;

import com.android.camera.manager.LittleIManager;

import com.android.camera.manager.ShowCSSpeedManager;

import com.android.camera.manager.ShutterManager;

import com.android.camera.manager.ThumbnailManager;

import com.android.camera.manager.ViewManager;

import com.android.camera.manager.ZoomManager;

import com.android.camera.ui.FrameView;

import com.android.camera.ui.PreviewFrameLayout;

import com.android.camera.ui.RotateLayout;

import com.android.camera.ui.ShutterButton;

import com.android.camera.ui.ShutterButton.OnShutterButtonListener;

import com.mediatek.camera.FrameworksClassFactory;

import com.mediatek.camera.ext.ExtensionHelper;

import com.mediatek.camcorder.CamcorderProfileEx;

import com.mediatek.common.featureoption.FeatureOption;

import com.mediatek.camera.ext.IAppGuideExt;

import com.android.camera.R;

import android.graphics.ImageFormat;

import java.io.FileDescriptor;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import java.util.concurrent.CopyOnWriteArrayList;

//ysfeagle add for frame begin

import com.android.camera.actor.SimpleActor;

import com.android.camera.ui.Panel;

import com.android.camera.ui.Panel.OnPanelListener;

import android.widget.AdapterView;

import android.widget.BaseAdapter;

import android.os.Bundle;

import android.view.View;

import android.view.ViewGroup;

import android.view.Window;

import android.view.animation.AnimationUtils;

//import android.view.ViewConfiguration;

import android.widget.Gallery;

import android.widget.ImageSwitcher;

import android.widget.ImageView;

import android.widget.AdapterView.OnItemClickListener;

import android.widget.AdapterView.OnItemSelectedListener;

import android.widget.Gallery.LayoutParams;

import android.widget.ViewSwitcher.ViewFactory;

import com.android.camera.ui.SUI\_firstCircleLayout;

import com.android.camera.ui.SUI\_CircleLayout;

import com.android.camera.ui.SUI\_CircleLayout.SUI\_OnItemSelectedListener;

import com.android.camera.ui.SUI\_firstCircleLayout.SUI\_firstcircle\_OnItemSelectedListener;

import android.widget.RelativeLayout;

import com.crunchfish.helper.camera.CameraSurface;

import com.crunchfish.touchless\_a3d.TA3DGesture;

import com.crunchfish.touchless\_a3d.TA3DGestureListener;

import com.crunchfish.touchless\_a3d.TouchlessA3D;

import com.crunchfish.touchless\_a3d.exception.LicenseNotValidException;

import com.crunchfish.touchless\_a3d.exception.LicenseServerUnavailableException;

import com.crunchfish.touchless\_a3d.TA3DObjectPresence;

import com.crunchfish.touchless\_a3d.TA3DObjectPresence.Action;

import com.crunchfish.touchless\_a3d.gestures.TA3DThumbsUpPresence;

import com.crunchfish.touchless\_a3d.gestures.TA3DOpenHandPresence;

import com.crunchfish.touchless\_a3d.gestures.TA3DVSignPresence;

import com.crunchfish.touchless\_a3d.gestures.TA3DFacePresence;

import com.crunchfish.touchless\_a3d.gestures.TA3DSwipe;

import com.crunchfish.helper.integration.ObjectStabilityTracker;

import com.crunchfish.helper.integration.ObjectStabilityTracker.StabilityListener;

import android.widget.FrameLayout;

import android.graphics.PixelFormat;

import com.android.camera.manager.TouchlessSelfTimerManager;

import com.android.camera.manager.TouchlessSelfTimerManager.TouchlessSelfTimerListener;

import android.hardware.Camera.FaceDetectionListener;

import android.hardware.Camera.Face;

import com.android.camera.ui.DragLayer; //add by zzp

import android.media.MediaPlayer;

import android.media.MediaPlayer.OnCompletionListener;

import android.os.Vibrator;

//ysfeagle add end

/\*

\* --ActivityBase

\* ----Camera //will control CameraActor, ModePicker, CameraPicker, SettingChecker, FocusManager, RemainingManager

\* --CameraActor

\* ------VideoActor

\* ------PhotoActor

\* --------NormalActor //contains continuous shot

\* --------HdrActor

\* --------FaceBeautyActor

\* --------AsdActor

\* --------SmileShotActor

\* --------PanaromaActor

\* --------MavActor

\*/

public class Camera extends ActivityBase implements PreviewFrameLayout.OnSizeChangedListener,

CameraScreenNail.FrameListener,

ViewFactory,OnPanelListener,

SUI\_OnItemSelectedListener ,SUI\_firstcircle\_OnItemSelectedListener{

private static final String TAG = "Camera";

private static final boolean LOG = Log.LOGV; //zzp

public interface OnOrientationListener {

void onOrientationChanged(int orientation);

}

public interface OnParametersReadyListener {

void onCameraParameterReady();

}

public interface OnPreferenceReadyListener {

void onPreferenceReady();

}

public interface Resumable {

void begin();

void resume();

void pause();

void finish();

}

public interface OnFullScreenChangedListener {

void onFullScreenChanged(boolean full);

}

public interface OnSingleTapUpListener {

void onSingleTapUp(View view, int x, int y);

}

public interface OnLongPressListener {

void onLongPress(View view, int x, int y);

}

public static final int UNKNOWN = -1;

public static final int STATE\_PREVIEW\_STOPPED = 0;

public static final int STATE\_IDLE = 1; // preview is active

public static final int STATE\_FOCUSING = 2; // Focus is in progress. The exact focus state is in Focus.java.

public static final int STATE\_SNAPSHOT\_IN\_PROGRESS = 3;

public static final int STATE\_RECORDING\_IN\_PROGRESS = STATE\_SNAPSHOT\_IN\_PROGRESS; //share the same value

public static final int STATE\_SWITCHING\_CAMERA = 4;// Switching between cameras.

public static final int STATE\_PREVIEW\_START\_ING = 5;// add this state to work wrong for Gallery,when back key press after onPause

private static final int OT\_TOAST\_SHOW\_MAX\_NUM = 3;

private int mCameraState = STATE\_PREVIEW\_STOPPED;

private boolean mCameraOpened;

private CameraManager.CameraProxy mCameraDevice;

private Parameters mInitialParams;

private Parameters mParameters;

private boolean mOpenCameraFail;

private boolean mCameraDisabled;

private boolean mIsModeChanged;

private CameraActor mCameraActor;

private ConditionVariable mStartPreviewPrerequisiteReady = new ConditionVariable();

private PreviewFrameLayout mPreviewFrameLayout;

public DragLayer mFrameParent; //add by zzp

public FrameLayout mFrameParent\_land; //add by zzp

//we should cache preview width and height for onConfigurationChanged().

private int mPreviewFrameWidth;

private int mPreviewFrameHeight;

private SurfaceTexture mSurfaceTexture;

private CameraStartUpThread mCameraStartUpThread;

private int mNumberOfCameras;

private int mCameraId;

// The activity is going to switch to the specified camera id. This is

// needed because texture copy is done in GL thread. -1 means camera is not

// switching.

private int mPendingSwitchCameraId = UNKNOWN;

private long mOnResumeTime;

private boolean mAppGuideFinished;

private static int mOtToastShowNum;

// The display rotation in degrees. This is only valid when mCameraState is

// not PREVIEW\_STOPPED.

private int mDisplayRotation;

// The value for android.hardware.Camera.setDisplayOrientation.

private int mCameraDisplayOrientation;

// The value for UI components like indicators.

private int mDisplayOrientation;

// The degrees of the device rotated clockwise from its natural orientation.

private int mOrientation = OrientationEventListener.ORIENTATION\_UNKNOWN;

// The orientation compensation for icons and thumbnails. Ex: if the value

// is 90, the UI components should be rotated 90 degrees counter-clockwise.

private int mOrientationCompensation = 0;

private MyOrientationEventListener mOrientationListener;

private ModePicker mModePicker;

public ShutterManager mShutterManager;

private ThumbnailManager mThumbnailManager;

private SettingManager mSettingManager;

private LittleIManager mLittleIManager;

private IndicatorManager mIndicatorManager;

private PickerManager mPickerManager;

//private RemainingManager mRemainingManager;

private InfoManager mInfoManager;

private TouchlessGuideManager mTouchlessGuideManager;//wenguangyu, add for touchless

private ReviewManager mReviewManager;

private ZoomManager mZoomManager;

private FileSaver mFileSaver;

private SettingChecker mSettingChecker;

private VoiceManager mVoiceManager;

private CameraSettings mCameraSettings;

private FrameManager mFrameManager;

private ShowCSSpeedManager mCsSpeedManager; // add for the cs speed

private ViewGroup mViewLayerBottom;

private ViewGroup mViewLayerNormal;

private ViewGroup mViewLayerTop;

private ViewGroup mViewLayerShutter;

private ViewGroup mViewLayerSetting;

private ViewGroup mViewLayerOverlay;

private FocusManager mFocusManager;

private RotateLayout mFocusAreaIndicator;

private RelativeLayout mFocusAreaFrame;

//add by zzp start

private RotateLayout mMeteringAreaIndicator;

private RelativeLayout mMeteringAreaFrame;

//add by zzp end

//private View mFrameParent;

private ComboPreferences mPreferences;

private PreferenceGroup mPreferenceGroup;

private LocationManager mLocationManager;

private RotateDialog mRotateDialog;

private RotateProgress mRotateProgress;

private OnScreenHint mRotateToast;

private LayoutInflater mInflater;

private String mFlashMode;

private String mSlowMotion;

private WfdManagerLocal mWfdLocal;

private PowerManager mPowerManager;

// This is the timeout to keep the camera in onPause for the first time

// after screen on if the activity is started from secure lock screen.

private static final int KEEP\_CAMERA\_TIMEOUT = 1000; // ms

private static final int MSG\_CAMERA\_OPEN\_DONE = 1;

private static final int MSG\_CAMERA\_PARAMETERS\_READY = 2;

private static final int MSG\_CAMERA\_PREFERENCE\_READY = 3;

private static final int MSG\_CHECK\_DISPLAY\_ROTATION = 4;

private static final int MSG\_SWITCH\_CAMERA = 5;

private static final int MSG\_SWITCH\_CAMERA\_START\_ANIMATION = 6;

private static final int MSG\_CLEAR\_SCREEN\_DELAY = 7;

private static final int MSG\_SHOW\_ONSCREEN\_INDICATOR = 8;

private static final int MSG\_OPEN\_CAMERA\_FAIL = 9;

private static final int MSG\_OPEN\_CAMERA\_DISABLED = 10;

// private static final int MSG\_FIRST\_FRAME\_ARRIVED = 14;

private static final int MSG\_APPLY\_PARAMETERS\_WHEN\_IDEL = 12;

private static final int MSG\_SET\_PREVIEW\_ASPECT\_RATIO = 15;

private static final int MSG\_DELAY\_SHOW\_ONSCREEN\_INDICATOR = 16;

private static final int MSG\_UPDATE\_SWITCH\_ACTOR\_STATE = 17;

//private static final int DELAY\_MSG\_SCREEN\_SWITCH = 2 \* 60 \* 1000;

private static final int DELAY\_MSG\_SCREEN\_SWITCH = 60 \* 1000;//songyutao add power-saving mode

private static final int DELAY\_MSG\_SHOW\_ONSCREEN\_VIEW = 3 \* 1000;

//add for Show CS Speed

private static final int DELAY\_MSG\_SHOW\_ONSCREEN\_CS\_SPEED\_VIEW = 1 \* 1000;

public static final int SHOW\_INFO\_LENGTH\_LONG = 3 \* 1000;

//SMB:VR will plug in/out of HDMI,will use this

public boolean isOnsaveInstance = false;

/\*add by wenguangyu ,for touchless shutter hand, START\*/

private android.hardware.Camera.Size mTouchlessPreviewSize;//wenguangyu

private int mTouchlessThumbs = 0;

private int mTouchlessFaces = 0;

private TouchlessTask mTouchlessTask;

private TouchlessA3D mTouchlessEngine;

private byte[][] mPreviewBuffers;

//private byte[] mPreviewTempBuffer;

private int mCurrentPreviewBuffer = 0;

private int fps\_frames = 0;

private long fps\_lastTime = 0;

public enum PresenceType {

Presence\_Normal,

Presence\_TA3DVSign,

Presence\_TA3DThumbsUp,

Presence\_TA3DOpenHand

};

private PresenceType mPresenceType;

// ObjectStabilityTracker

public static ObjectStabilityTracker mObjectStabilityTracker;// = new ObjectStabilityTracker();

/\*add by wenguangyu ,for touchless shutter hand, END\*/

private static final int DELAY\_MSG\_SHOW\_ONSCREEN\_TIME = 1 \* 1000;

private CharSequence mDelayShowInfo;

private int mDelayOtherMessageTime;

private boolean mIsSwitchActorRunning = false;

//ivate boolean mIsFirstSwitchFrontCamera = true;

private int mCurrentBackMode = ModePicker.MODE\_PHOTO;

private int mCurrentFrontMode = ModePicker.MODE\_FACE\_BEAUTY;

private Vibrator mVibrator;

//ysfeagle add for frame begin

private RotateProgress mTinnoRotateProgress;

private int mTinnoRecNum=1;

private int mTinnoRecMax=1;

private static final int MSG\_CAMERA\_SHOW\_PROGRESS = 30;

public int current\_index = 0;

public int isfullscreen=0;//0==4:3/ 1==16:9

public int lastfullscreen=0;//0==4:3/ 1==16:9

public boolean is\_full\_screen\_for\_frame = false;

//public boolean is\_open\_precapture = false;

//public boolean key\_open\_precapture = false;

private SharedPreferences mSharedPreference;

private static final String FRAME\_PREFERENCE = "pref\_camera\_frame\_key";

private static final String CAMERA\_FRAME\_KEY = "camera\_frame\_id";

private ImageSwitcher is;

private Gallery gallery;

private Panel panel;

private Panel bottomPanel;

public SUI\_firstCircleLayout first\_circle\_meun;

public SUI\_CircleLayout circle\_main;

FrameLayout circlelayout;

public Handler mhandler;

private Handler mhandlerframe;

//ysfeagle add for fix bug DSAGDG-2

protected boolean IsSnapCamera=false;

//ysfeagle add for frame end

private static String getMsgLabel(int msg) {

switch (msg) {

case MSG\_CAMERA\_OPEN\_DONE: return "MSG\_CAMERA\_OPEN\_DONE";

case MSG\_CAMERA\_PARAMETERS\_READY: return "MSG\_CAMERA\_PARAMETERS\_READY";

case MSG\_CAMERA\_PREFERENCE\_READY: return "MSG\_CAMERA\_PREFERENCE\_READY";

case MSG\_CHECK\_DISPLAY\_ROTATION: return "MSG\_CHECK\_DISPLAY\_ROTATION";

case MSG\_SWITCH\_CAMERA: return "MSG\_SWITCH\_CAMERA";

case MSG\_SWITCH\_CAMERA\_START\_ANIMATION: return "MSG\_SWITCH\_CAMERA\_START\_ANIMATION";

case MSG\_CLEAR\_SCREEN\_DELAY: return "MSG\_CLEAR\_SCREEN\_DELAY";

case MSG\_SHOW\_ONSCREEN\_INDICATOR: return "MSG\_SHOW\_ONSCREEN\_INDICATOR";

case MSG\_OPEN\_CAMERA\_FAIL: return "MSG\_OPEN\_CAMERA\_FAIL";

case MSG\_OPEN\_CAMERA\_DISABLED: return "MSG\_OPEN\_CAMERA\_DISABLED";

// case MSG\_FIRST\_FRAME\_ARRIVED: return "MSG\_FIRST\_FRAME\_ARRIVED";

case MSG\_APPLY\_PARAMETERS\_WHEN\_IDEL: return "MSG\_APPLY\_PARAMETERS\_WHEN\_IDEL";

case MSG\_SET\_PREVIEW\_ASPECT\_RATIO: return "MSG\_SET\_PREVIEW\_ASPECT\_RATIO";

default:

break;

}

return "unknown message";

}

private Handler mMainHandler = new Handler() {

@Override

public void handleMessage(Message msg) {

Log.i(TAG, "handleMessage(" + msg + ")");

MMProfileManager.startProfileHandleMessage(getMsgLabel(msg.what));

switch(msg.what) {

case MSG\_CAMERA\_OPEN\_DONE:

mCameraStartUpThread = null;

//mRecorderSwitcher.setEnabled(true);

break;

case MSG\_CAMERA\_PARAMETERS\_READY:

notifyParametersReady();

applyMyParamsTinno();

break;

case MSG\_CAMERA\_PREFERENCE\_READY:

notifyPreferenceReady();

break;

case MSG\_CHECK\_DISPLAY\_ROTATION:

// Set the display orientation if display rotation has changed.

// Sometimes this happens when the device is held upside

// down and camera app is opened. Rotation animation will

// take some time and the rotation value we have got may be

// wrong. Framework does not have a callback for this now.

if (Util.getDisplayRotation(Camera.this) != mDisplayRotation) {

setDisplayOrientation();

mOrientation = OrientationEventListener.ORIENTATION\_UNKNOWN;

mCameraActor.onDisplayRotate();

}

if (SystemClock.uptimeMillis() - mOnResumeTime < 5000) {

mMainHandler.sendEmptyMessageDelayed(MSG\_CHECK\_DISPLAY\_ROTATION,

100);

MMProfileManager.triggersSendMessage(

getMsgLabel(MSG\_CHECK\_DISPLAY\_ROTATION) + ", delayed 100");

}

notifyOrientationChanged();

break;

case MSG\_SWITCH\_CAMERA:

switchCamera();

break;

case MSG\_SWITCH\_CAMERA\_START\_ANIMATION:

mCameraScreenNail.animateSwitchCamera();

//tinno\_Pre\_Camera\_reset\_state();

break;

case MSG\_CLEAR\_SCREEN\_DELAY:

getWindow().clearFlags(WindowManager.LayoutParams.FLAG\_KEEP\_SCREEN\_ON);

if(getCameraState()!=STATE\_RECORDING\_IN\_PROGRESS){//songyutao add power-saving mode

//mPowerManager.goToSleep(SystemClock.uptimeMillis());

finish();

}

break;

case MSG\_SHOW\_ONSCREEN\_INDICATOR:

doShowIndicator();

break;

// case MSG\_FIRST\_FRAME\_ARRIVED:

// doOnFirstFrameArrived();

// break;

case MSG\_APPLY\_PARAMETERS\_WHEN\_IDEL:

applyParameters(false);

break;

case MSG\_OPEN\_CAMERA\_FAIL:

mCameraStartUpThread = null;

mOpenCameraFail = true;

Util.showErrorAndFinish(Camera.this, R.string.cannot\_connect\_camera);

break;

case MSG\_OPEN\_CAMERA\_DISABLED:

mCameraStartUpThread = null;

mCameraDisabled = true;

Util.showErrorAndFinish(Camera.this, R.string.camera\_disabled);

break;

case MSG\_SET\_PREVIEW\_ASPECT\_RATIO:

setPreviewFrameLayoutAspectRatio();

break;

case MSG\_DELAY\_SHOW\_ONSCREEN\_INDICATOR:

// will handle the delay hide the remain when is show remain

//mRemainingManager.hide();

mInfoManager.showText(mDelayShowInfo);

showIndicator(mDelayOtherMessageTime);

break;

case MSG\_UPDATE\_SWITCH\_ACTOR\_STATE:

//mModePicker.setEnabled(true);

break;

case MSG\_CAMERA\_SHOW\_PROGRESS:

Tinno\_Increase\_Progress();

break;

default:

break;

}

MMProfileManager.stopProfileHandleMessage();

};

};

/// M: open camera process functions @{

private class CameraStartUpThread extends Thread {

private boolean mCancel = false;

@Override

public void run() {

MMProfileManager.startProfileCameraStartUp();

//ensure current camera id is correct.

int cameraId = getPreferredCameraId(mPreferences);

if (mCameraId != cameraId) {

Log.w(TAG, "CameraStartUpThread.run() camera id preference=" + cameraId + ", memory=" + mCameraId);

CameraSettings.writePreferredCameraId(mPreferences, mCameraId);

//in order to do filterUnsuportedPreference in initializeCameraPreferences

//To reset DEFAULT\_VALUE\_FOR\_SETTING

mPreferenceGroup = null;

}

try {

mCameraDevice = Util.openCamera(Camera.this, mCameraId);

// M: added for mock camera

prepareMockCamera();

mCameraOpened = true;

} catch (CameraHardwareException e) {

mCameraActor.onCameraOpenFailed();

mOpenCameraFail = true;

mMainHandler.sendEmptyMessage(MSG\_OPEN\_CAMERA\_FAIL);

MMProfileManager.triggersSendMessage(

getMsgLabel(MSG\_OPEN\_CAMERA\_FAIL));

MMProfileManager.stopProfileCameraStartUp();

return;

} catch (CameraDisabledException e) {

mCameraActor.onCameraDisabled();

mCameraDisabled = true;

mMainHandler.sendEmptyMessage(MSG\_OPEN\_CAMERA\_DISABLED);

MMProfileManager.triggersSendMessage(

getMsgLabel(MSG\_OPEN\_CAMERA\_DISABLED));

MMProfileManager.stopProfileCameraStartUp();

return;

}

mInitialParams = CameraHolder.instance().getOriginalParameters();

MMProfileManager.startProfileCameraParameterCopy();

mParameters = mInitialParams.copy();

ModeChecker.updateModeMatrix(Camera.this,mCameraId);

MMProfileManager.stopProfileCameraParameterCopy();

mCameraActor.onCameraOpenDone();

MMProfileManager.startProfileCameraPreviewPreReadyBlock();

mStartPreviewPrerequisiteReady.block();

MMProfileManager.stopProfileCameraPreviewPreReadyBlock();

if (mCancel) {

MMProfileManager.stopProfileCameraStartUp();

return;

}

initializeFocusManager();

setDisplayOrientation();//should be set before initialize surface

//updateSurfaceTexture();//should wait camera screennail.

//setPreviewTextureAsync();

initializeCameraPreferences();//should be rechecked

clearDeviceCallbacks();

applyDeviceCallbacks();

clearViewCallbacks();

applayViewCallbacks();

setTouchlessParameters();//wenguangyu

applyParameters(true);

mOnResumeTime = SystemClock.uptimeMillis();

mMainHandler.sendEmptyMessage(MSG\_CHECK\_DISPLAY\_ROTATION);

MMProfileManager.triggersSendMessage(

getMsgLabel(MSG\_CHECK\_DISPLAY\_ROTATION));

//send message to avoid onResume check

mMainHandler.sendEmptyMessage(MSG\_CAMERA\_OPEN\_DONE);

MMProfileManager.triggersSendMessage(

getMsgLabel(MSG\_CAMERA\_OPEN\_DONE));

//not in main thread to do the mkdir action

//will maybe ANR when IO is busy

Storage.mkFileDir(Storage.getFileDirectory());

MMProfileManager.stopProfileCameraStartUp();

}

public void cancel() {

mCancel = true;

}

public boolean isCanceled() {

Log.i(TAG, "isCanceled() return " + mCancel);

return mCancel;

}

}

private void setTouchlessParameters() {

android.hardware.Camera.Size s = mParameters.getPreviewSize();

int[] fpsRange = new int[2];

mParameters.getPreviewFpsRange(fpsRange);

getMaxFpsRange(mParameters.getSupportedPreviewFpsRange(), fpsRange);

/\*Log.d(TAG, "TA3D Setting preview fps range: " +

fpsRange[android.hardware.Camera.Parameters.PREVIEW\_FPS\_MIN\_INDEX] + " - " +

fpsRange[android.hardware.Camera.Parameters.PREVIEW\_FPS\_MAX\_INDEX]);\*/

mParameters.setPreviewFpsRange(

fpsRange[android.hardware.Camera.Parameters.PREVIEW\_FPS\_MIN\_INDEX],

fpsRange[android.hardware.Camera.Parameters.PREVIEW\_FPS\_MAX\_INDEX]);

//mParameters.setPictureFormat(ImageFormat.JPEG);

mParameters.setPreviewFormat(ImageFormat.NV21);//(ImageFormat.JPEG);

//Log.d(TAG, "TA3D -setTouchlessParameters mParameters getPreviewFormat: " + mParameters.getPreviewFormat());

//Log.d(TAG, "TA3D -setTouchlessParameters mParameters getPictureFormat: " + mParameters.getPictureFormat());

}

private void prepareMockCamera() {

if (null == mCameraDevice) {

return;

}

if (null != mCameraDevice.getCamera() &&

FrameworksClassFactory.isMockCamera()) {

// M: set context to create mock images

mCameraDevice.getCamera().setContext(this);

}

}

private void closeCamera() {

Log.i(TAG, "closeCamera() mCameraDevice=" + mCameraDevice);

if (mCameraDevice != null) {

mCameraActor.onCameraClose();

clearDeviceCallbacks();

CameraHolder.instance().release();

if (mFocusManager != null) {

mFocusManager.onCameraReleased();

}

mCameraDevice = null;

setCameraState(STATE\_PREVIEW\_STOPPED);

mCameraOpened = false;

}

}

private void waitCameraStartUpThread(boolean cancel) {

Log.i(TAG, "waitCameraStartUpThread(" + cancel + ") begin mCameraStartUpThread=" + mCameraStartUpThread);

try {

if (mCameraStartUpThread != null) {

if (cancel) {

mCameraStartUpThread.cancel();

}

mCameraStartUpThread.join();

mCameraStartUpThread = null;

}

} catch (InterruptedException e) {

Log.w(TAG, "waitCameraStartUpThread()", e);

}

Log.i(TAG, "waitCameraStartUpThread() end");

}

private void keepCameraForSecure(){

// When camera is started from secure lock screen for the first time

// after screen on, the activity gets onCreate->onResume->onPause->onResume.

// To reduce the latency, keep the camera for a short time so it does

// not need to be opened again.

if (mCameraDevice != null && isSecureCamera()

&& isFirstStartAfterScreenOn()) {

resetFirstStartAfterScreenOn();

CameraHolder.instance().keep(KEEP\_CAMERA\_TIMEOUT);

}

}

private Size mLastPictureSize;

private Size mLastPreviewSize;

private String mLastZsdMode;

private int mLastAudioBitRate = UNKNOWN;

private int mLastVideoBitRate = UNKNOWN;

private void applyParameters(boolean force) {

if (cancelApplyParameters()) {

return;

}

MMProfileManager.startProfileApplyParameters();

lockRun(new Runnable() {

@Override

public void run() {

//preview size changed, zsd changed, camera mode changed, open camera.

mSettingChecker.applyPreferenceToParameters();

//Some preference are not set to native,but must do applyPreferenceToParameters

//because these preference must be set before start preview

mSettingChecker.applyParametersToUIImmediately();

}

});

//Aspect ratio default value is 4:3

//in full screen, in order to improve launch performance(reduce RelativeFrame change animation)

//We should setPreviewFrameLayoutAspectRatio as soon as possible

mMainHandler.sendEmptyMessage(MSG\_SET\_PREVIEW\_ASPECT\_RATIO);

//capability table will be used when value is applied.

Size curPictureSize = mParameters.getPictureSize();

Size curPreviewSize = mParameters.getPreviewSize();

String curZsd = mParameters.getZSDMode();

final boolean changedPreviewSize = !curPreviewSize.equals(mLastPreviewSize);

//camera screenail size will changed if preview ratio change

boolean previewRatioChanged = false;

if(mLastPreviewSize != null) {

previewRatioChanged = !(((double)curPreviewSize.width / curPreviewSize.height)

== ((double)mLastPreviewSize.width / mLastPreviewSize.height));

}

final boolean changedPictureSize = !curPictureSize.equals(mLastPictureSize);

final boolean changedZsd = curZsd == null ? mLastZsdMode != null : !curZsd.equals(mLastZsdMode);

final boolean needRestart = changedZsd || changedPreviewSize || force;

boolean vBRateChanged = false;

boolean aBRateChanged = false;

if (mProfile != null) {

vBRateChanged = (mLastVideoBitRate == UNKNOWN ? true : mLastVideoBitRate != mProfile.videoBitRate);

aBRateChanged = (mLastAudioBitRate == UNKNOWN ? true : mLastAudioBitRate != mProfile.audioBitRate);

}

if (cancelApplyParameters()) {

MMProfileManager.stopProfileApplyParameters();

return;

}

mCameraDevice.setDisplayOrientation(mCameraDisplayOrientation);

if (needRestart) {

mCameraActor.stopPreview();

//video wallpaper launch camera, the preview size is not changed, not need copy texture.

if (!isSwitchingCamera() && previewRatioChanged && mSurfaceTexture != null && !isVideoWallPaperIntent()) {

mCameraScreenNail.copyOriginSizeTexture();

} else {

mCameraScreenNail.stopSwitchActorAnimation();

}

mCameraScreenNail.setDrawable(false);

mCameraScreenNail.setOnLayoutChanged(false);

updateSurfaceTexture();

setPreviewTextureAsync();

}

if (cancelApplyParameters()) {

MMProfileManager.stopProfileApplyParameters();

return;

}

lockRun(new Runnable() {

@Override

public void run() {

setRotationToParameters(); //set rotation to parameters for face detection.

setZoomParameter(); //maintain last ZoomValue to parameters after resume

applyParametersToServer();

if (changedPreviewSize) {

// For picture: Zoom related settings will be changed for different preview

// sizes, so set and read the parameters to get latest values

// For video: Keep preview size up to date.

fetchParametersFromServer();

}

}

});

if (cancelApplyParameters()) {

MMProfileManager.stopProfileApplyParameters();

return;

}

//Zoom manager will be notified here.

mMainHandler.sendEmptyMessage(MSG\_CAMERA\_PARAMETERS\_READY);

MMProfileManager.triggersSendMessage(

getMsgLabel(MSG\_CAMERA\_PARAMETERS\_READY));

mCameraActor.onCameraParameterReady(needRestart);

if ((changedPictureSize || changedPreviewSize) || (force || vBRateChanged || aBRateChanged)) {

//put showRemainingAways() after mCameraActor.onCameraParamtersReady() to avoid read

//parameters when write.

//showRemainingAways();

Storage.updateStorage(this);

}

Log.i(TAG, "applyParameters(" + force + ") picturesize=" + SettingUtils.buildSize(curPictureSize)

+ " previewsize=" + SettingUtils.buildSize(curPreviewSize)

+ " oldPictureSize=" + SettingUtils.buildSize(mLastPictureSize)

+ " oldPreviewSize=" + SettingUtils.buildSize(mLastPreviewSize)

+ " changedPreviewSize=" + changedPreviewSize + ", changedPictureSize=" + changedPictureSize

+ " oldZsd=" + mLastZsdMode + ", curZsd=" + curZsd + ", changedZsd=" + changedZsd

+ " vBRateChanged=" + vBRateChanged + ", aBRateChanged=" + aBRateChanged);

mLastPictureSize = curPictureSize;

mLastPreviewSize = curPreviewSize;

mLastZsdMode = curZsd;

if (mProfile != null) {

mLastVideoBitRate = mProfile.videoBitRate;

mLastAudioBitRate = mProfile.audioBitRate;

}

MMProfileManager.stopProfileApplyParameters();

}

private void applyMyParamsTinno(){

if (mCameraDevice == null)

return;

//shutter sound

boolean onoff = isShutterSoundOpen();

mCameraDevice.enableShutterSound(onoff);

mCameraDevice.enableVideoSound(onoff);

//shutter thumb

if(checkHandShutter())

{

initalizeTouchless();

mLittleIManager.show();

}

else

{

stopTouchlessEngine();

mLittleIManager.hide();

}

//goutu

runOnUiThread(new Runnable(){

@Override

public void run(){

getSettingManager().showPicStruct();

}

});

//pic frame

/\*if(!mCameraScreenNail.isAnimatingSwitchCamera())

tinno\_frame\_get\_key();

else

mMainHandler.postDelayed(new Runnable(){

@Override

public void run(){

tinno\_frame\_get\_key();

}

},1000); \*/

tinno\_frame\_get\_key();

//self mirror

//Log.d("songyutao", "getCameraId=" +getCameraId());

//if(getCameraId()==1){

// getParameters().setFlip(1);

//}else{

// getParameters().setFlip(0);

//}

//pic level

mSettingManager.enablePicLevel();

//add other below...

}

private boolean cancelApplyParameters() {

boolean cancel = (mCameraDevice == null || mParameters == null || mPaused

|| (mCameraStartUpThread != null && mCameraStartUpThread.isCanceled()));

if (cancel) {

Log.i(TAG, "cancelApplyParameters() mCameraDevice=" + mCameraDevice + ", " + "mParameters="

+ mParameters + ", mPaused=" + mPaused + " return " + cancel);

}

return cancel;

}

protected void onAfterFullScreeChanged(final boolean full) {

Log.d(TAG, "onAfterFullScreeChanged(" + full + ")");

if (full) {

/\* M: MSG\_CHECK\_DISPLAY\_ROTATION may get a wrong display rotation in this case:

\* Enter camera(protrait) --> change to landscape --> click thumbnail.

\* Then, when back to camera, mDisplayRotation will be 270 instead of 90.

\* So here we recheck it for this case.

\* when CameraStartUpThread done, we can check display rotation.

\*/

//getWindow().addFlags(WindowManager.LayoutParams.FLAG\_TRANSLUCENT\_STATUS);

//getWindow().addFlags(WindowManager.LayoutParams.FLAG\_TRANSLUCENT\_NAVIGATION);

//int flags=0;

//flags |= View.SYSTEM\_UI\_FLAG\_LAYOUT\_STABLE

//| View.SYSTEM\_UI\_FLAG\_LAYOUT\_HIDE\_NAVIGATION

//| View.SYSTEM\_UI\_FLAG\_LAYOUT\_FULLSCREEN;

//findViewById(R.id.camera\_root).setSystemUiVisibility(flags);

//requestRender();

//int flags =0;

//flags=getGLRoot().extGetSystemUiVisibility();

//flags|=View.SYSTEM\_UI\_FLAG\_SHOW\_NAVIGATION;

//getGLRoot().extSetSystemUiVisibility(flags);

//setTheme(android.R.style.Theme\_Translucent\_NoTitleBar);

if (mOnResumeTime != 0L) {

mMainHandler.sendEmptyMessage(MSG\_CHECK\_DISPLAY\_ROTATION);

}

if(ModePicker.MODE\_PANORAMA == getCurrentMode()) { //add by zzp

boolean rotationLocked = (1 != Settings.System.getInt(getContentResolver(), Settings.System.ACCELEROMETER\_ROTATION, 0) );

mCameraScreenNail.setRotationLocked(rotationLocked);

mCameraScreenNail.setPanoIdle(true);

}

MMProfileManager.triggersSendMessage(

getMsgLabel(MSG\_CHECK\_DISPLAY\_ROTATION));

//if (getResources().getConfiguration().orientation == Configuration.ORIENTATION\_LANDSCAPE )

//{

//disableOrientationListener();

//setOrientation(true, 90);

//setRequestedOrientation(ActivityInfo.SCREEN\_ORIENTATION\_PORTRAIT); //SCREEN\_ORIENTATION\_PORTRAIT

//}

//setOrientation(true, 90);

//if(mFocusAreaFrame!=null)mFocusAreaFrame.setVisibility(View.GONE);

mMainHandler.post(new Runnable() {

@Override

public void run() {

//setOrientation(false, 0);

if (getResources().getConfiguration().orientation == Configuration.ORIENTATION\_LANDSCAPE )

//if (newConfig.orientation == Configuration.ORIENTATION\_LANDSCAPE

//&& mShowCameraAppView)

{

//setOrientation(true, 90);

setRequestedOrientation(ActivityInfo.SCREEN\_ORIENTATION\_PORTRAIT); //SCREEN\_ORIENTATION\_PORTRAIT

//Log.w("songyutao", "SCREEN\_ORIENTATION\_PORTRAIT");

}

}

});

} else {

//setTheme(android.R.style.Theme\_Holo\_Light\_NoActionBar\_Fullscreen);

//getWindow().clearFlags(WindowManager.LayoutParams.FLAG\_TRANSLUCENT\_STATUS);

//getWindow().clearFlags(WindowManager.LayoutParams.FLAG\_TRANSLUCENT\_NAVIGATION);

mCameraScreenNail.setPanoIdle(false); //add by zzp

if (mFocusManager != null) {

mFocusManager.stopObjectTrack();

}

hideToast();

}

if (mSettingChecker != null) {

lockRun(new Runnable() {

@Override

public void run() {

if (full) {

mSettingChecker.turnOnWhenShown();

} else {

mSettingChecker.turnOffWhenHide();

}

}

});

}

notifyOnFullScreenChanged(full);

}

public void setCameraState(int state) {

Log.i(TAG, "setCameraState(" + state + ")");

mCameraState = state;

}

public int getCameraState() {

return mCameraState;

}

public boolean isCameraClosed() {

return !mCameraOpened;

}

public boolean isCameraIdle() {

boolean idle = (mCameraState == STATE\_IDLE) || ((mFocusManager != null)

&& mFocusManager.isFocusCompleted() && (mCameraState != STATE\_SWITCHING\_CAMERA));

Log.d(TAG, "isCameraIdle() mCameraState=" + mCameraState + ", return " + idle);

return idle;

}

public boolean isShutterSoundOpen() {

return ("on".equals(getListPreference(SettingChecker.ROW\_SETTING\_CAMERA\_SHUTTER\_SOUND).getValue()));

}

private boolean isNeedToSetLandScape() {

if (FeatureSwitcher.isSmartBookEnabled()) {

return getCurrentMode() == ModePicker.MODE\_MOTION\_TRACK || getCurrentMode() == ModePicker.MODE\_MAV;

} else {

return false;

}

}

protected void onSaveInstanceState(Bundle outState) {

Log.i(TAG, "onSaveInstanceState");

isOnsaveInstance = true;

}

/// M: for activity life cycle @{

@Override

public void onCreate(Bundle icicle) {

getWindow().addFlags(WindowManager.LayoutParams.FLAG\_TRANSLUCENT\_STATUS);

getWindow().addFlags(WindowManager.LayoutParams.FLAG\_TRANSLUCENT\_NAVIGATION);

//getWindow().addFlags(WindowManager.LayoutParams.FLAG\_FULLSCREEN);

super.onCreate(icicle);

Log.d(TAG, "onCreate,icicle = " +icicle);

MMProfileManager.startProfileCameraOnCreate();

mVibrator = (Vibrator)getSystemService(Service.VIBRATOR\_SERVICE);

parseIntent();

//should be checked whether can be moved to opening thread @{

mPreferences = new ComboPreferences(this);

CameraSettings.upgradeGlobalPreferences(mPreferences.getGlobal());

mCameraId = getPreferredCameraId(mPreferences);

mPreferences.setLocalId(this, mCameraId);

CameraSettings.upgradeLocalPreferences(mPreferences.getLocal());

// reset smile shot, hdr, asd sharepreference as off to avoid error

// when kill camera through IPO or other abnormal ways.

if (isNonePickIntent()) {

CameraSettings.updateSettingCaptureModePreferences(mPreferences.getLocal());

}

//@}

initializeStereo3DMode();

//initial managers before new actor, so we can count views created by Camera.java

initializeCommonManagers();

mFileSaver.bindSaverService();

if (isVideoCaptureIntent() || isVideoWallPaperIntent()) {

mCameraActor = new VideoActor(this);

} else if (isStereo3DImageCaptureIntent() && FeatureSwitcher.isStereoSingle3d()) {

//mCameraActor = new SingleStereoPhotoActor(this);

} else {

if(getCameraId() == 1){

if(isNonePickIntent())

mCameraActor = new FaceBeautyActor(this);

else{

mCameraActor = new PhotoActor(this);

mCurrentFrontMode = ModePicker.MODE\_PHOTO;

}

//sFirstSwitchFrontCamera = false;

}else{

if(isNonePickIntent())

mCameraActor = new PhotoActor(this);

else{

mCameraActor = new PhotoActor(this);

mCurrentFrontMode = ModePicker.MODE\_PHOTO;

}

}

}

//start camera opening process

mCameraStartUpThread = new CameraStartUpThread();

mCameraStartUpThread.start();

//for photo view loading camera folder

ExtensionHelper.ensureCameraExtension(this);

Storage.initializeStorageState();

//just set content view for preview

MMProfileManager.startProfileCameraViewOperation();

//if(ViewConfiguration.get(this).hasPermanentMenuKey()){

//int visible = getWindow().getDecorView().getSystemUiVisibility();

//getWindow().getDecorView().setSystemUiVisibility(visible | View.SYSTEM\_UI\_FLAG\_HIDE\_NAVIGATION);

//}

setContentView(R.layout.camera);

//View decorView =getWindow().getDecorView();

//decorView.setSystemUiVisibility( View.SYSTEM\_UI\_FLAG\_LAYOUT\_STABLE

//|View.SYSTEM\_UI\_FLAG\_LAYOUT\_HIDE\_NAVIGATION

//| View.SYSTEM\_UI\_FLAG\_LAYOUT\_FULLSCREEN

//View.SYSTEM\_UI\_FLAG\_HIDE\_NAVIGATION

//| View.SYSTEM\_UI\_FLAG\_FULLSCREEN

//| View.SYSTEM\_UI\_FLAG\_IMMERSIVE\_STICKY

//|View.SYSTEM\_UI\_FLAG\_LOW\_PROFILE

//);

int flags=0;

flags |= View.SYSTEM\_UI\_FLAG\_LAYOUT\_STABLE

| View.SYSTEM\_UI\_FLAG\_LAYOUT\_HIDE\_NAVIGATION

| View.SYSTEM\_UI\_FLAG\_LAYOUT\_FULLSCREEN;

findViewById(R.id.camera\_app\_root).setSystemUiVisibility(flags);

MMProfileManager.stopProfileCameraViewOperation();

//create camera screennail after content view inflated.

//createCameraScreenNail(isNonePickIntent());

createCameraScreenNail(isNonePickIntent());

mCameraScreenNail.setFrameListener(this);

mCameraScreenNail.setSwitchActorStateListener(mStateListener);

//ysfeagle add for init

tinno\_frame\_init();

//tinno\_frame\_get\_key();

//ysfeagle add end

//only initialize some thing for open

initializeForOpeningProcess();

MMProfileManager.startProfileCameraPreviewPreReadyOpen();

mStartPreviewPrerequisiteReady.open();

MMProfileManager.stopProfileCameraPreviewPreReadyOpen();

//may be we can lazy this function.

initializeAfterPreview();

MMProfileManager.stopProfileCameraOnCreate();

//if (mSecureCamera) {

// mPowerManager.wakeUp(SystemClock.uptimeMillis());

//}

if(mCameraActor.getMode() == ModePicker.MODE\_FACE\_BEAUTY){

//Log.d("tao", "onCreate, show fb UI");

((FaceBeautyActor)mCameraActor).showFaceBeauty();

}

//if(checkHandShutter())

//{

// initalizeTouchless();

//}

}

public void initalizeTouchless() {

Size paramsSize = mParameters.getPreviewSize();

Log.d(TAG, "TA3D - initalizeTouchless::: mParameters width: " + paramsSize.width + ", height: " + paramsSize.height);

mObjectStabilityTracker = new ObjectStabilityTracker(paramsSize.width, paramsSize.height);

startTouchlessEngine();

}

public void startTouchlessEngine() {

//Log.d(TAG, "TA3D - startTouchlessEngine:"+mTouchlessTask);

if (mTouchlessTask == null) {

mTouchlessThumbs = 0;//wenguangyu

mTouchlessFaces = 0;//wenguangyu

mTouchlessTask = new TouchlessTask();

mTouchlessTask.execute();

}

else

{

RestartTouchlessEngine();

}

}

public void stopTouchlessEngine() {

//Log.d(TAG, "TA3D - checkHandShutter stopTouchlessEngine successful");

if (mTouchlessTask != null) {

mTouchlessTask.cancel(true);

if(mTouchlessEngine != null)

{

mTouchlessEngine.unregisterGestureListener(TA3DThumbsUpPresence.TYPE,mCameraGestureCallback);//wenguangyu

mTouchlessEngine.unregisterGestureListener(TA3DVSignPresence.TYPE,mCameraGestureCallback);//wenguangyu

}

}

if(mObjectStabilityTracker != null)

{

mObjectStabilityTracker.unregisterStabilityListener(mStabilityListener);

//mObjectStabilityTracker = null;

}

}

public void RestartTouchlessEngine() {

//Log.d(TAG, "TA3D - reStartTouchlessEngine");

if(mTouchlessTask != null)

{

mTouchlessThumbs = 0;//wenguangyu

mTouchlessFaces = 0;//wenguangyu

clearFocusAndFace();

mTouchlessTask = null;

mTouchlessTask = new TouchlessTask();

mTouchlessTask.execute();

}

}

private int getCameraDisplayOrientationEx() {

CameraInfo info = CameraHolder.instance().getCameraInfo()[mCameraId];

// mOrientation contains current rotation of the device. 0 = portrait

int result = mOrientation;

if (result == -1) result = 0; // Happens when device is on a table

if (info.facing == CameraInfo.CAMERA\_FACING\_FRONT) {

//Log.d(TAG, "TA3D - CAMERA\_FACING\_FRONT");

result = (info.orientation - result + 360) % 360;

} else {

//Log.d(TAG, "TA3D - CAMERA\_FACING\_BACK");

result = (info.orientation + result) % 360;

}

return result;

}

private PreviewCallback mPreviewCallback = new PreviewCallback() {

@Override

public void onPreviewFrame(byte[] data, android.hardware.Camera camera) {

//Log.i(TAG, "TA3D - onPreviewFrame");

if(!checkHandShutter())

return;

TouchlessA3D.Rotate rotation = TouchlessA3D.Rotate.DO\_NOT\_ROTATE;

int rot = getCameraDisplayOrientationEx();

//Log.d(TAG, "TA3D - rotation - rotating image with deg: " + rot);

switch(rot)

{

case 0:

rotation = TouchlessA3D.Rotate.DO\_NOT\_ROTATE;

break;

case 90:

rotation = TouchlessA3D.Rotate.ROTATE\_90;

break;

case 180:

rotation = TouchlessA3D.Rotate.ROTATE\_180;

break;

case 270:

rotation = TouchlessA3D.Rotate.ROTATE\_270;

break;

}

if(mTouchlessEngine != null) {

//Log.d(TAG, "TA3D - mTouchlessEngine.handleImage");

//android.hardware.Camera.Parameters params = camera.getParameters();

//android.hardware.Camera.Size s = params.getPreviewSize();

//Log.d(TAG, "TA3D - width: " + s.width + ", height: " + s.height);

// copy data to mPreviewTempBuffer

//long prof0 = System.currentTimeMillis();

//for (int i = 0; i < data.length; i++) mPreviewTempBuffer[i]=data[i];

//Log.d(TAG, "##### TA3D - deep copy took: " + ( System.currentTimeMillis() - prof0));

//android.hardware.Camera.Parameters params = camera.getParameters();

mTouchlessPreviewSize = mParameters.getPreviewSize();

/\*int actualBpp = ImageFormat.getBitsPerPixel(mParameters.getPreviewFormat())/8;

Log.d(TAG, "TA3D -initializeTouchlessA3D mParameters getPreviewFormat: " + mParameters.getPreviewFormat());

Log.d(TAG, "TA3D -initializeTouchlessA3D mParameters getPictureFormat: " + mParameters.getPictureFormat());

Log.d(TAG, "TA3D -mParameters: " + mParameters.getPreviewFormat());

Log.d(TAG, "TA3D -CameraHolder.instance(): " + camera.getParameters().getPreviewFormat());

Log.d(TAG, "TA3D -actualBpp: " + actualBpp);

Log.d(TAG, "TA3D -actualBpp: " + actualBpp);

\*/

//Log.d(TAG, "TA3D - width: " + mTouchlessPreviewSize.width + ", height: " + mTouchlessPreviewSize.height);

//Log.d(TAG, "TA3D - Size of mPreviewBuffer; " + mPreviewBuffers[0].length);

long prof1 = System.currentTimeMillis();

mTouchlessEngine.handleImage(System.currentTimeMillis(), data, rotation);

// mTouchlessEngine.handleImage(System.currentTimeMillis(), mPreviewTempBuffer, rotation);

//Log.d(TAG, "##### TA3D - engine took: " + ( System.currentTimeMillis() - prof1));

{

fps\_frames++;

if (System.currentTimeMillis() - fps\_lastTime > 3000) {

float fps = ((float)fps\_frames) /

(

(float)(

(System.currentTimeMillis() - fps\_lastTime) / 1000.0f

)

);

fps\_frames = 0;

fps\_lastTime = System.currentTimeMillis();

//Log.d(TAG, "##### TA3D - FPS: " + fps);

}

}

//String bytes = "First bytes: ";

// for (int i = 0; i < 16; i++){

// bytes += " " + data[i];

//}

//Log.d(TAG, bytes);

}

//mCameraDevice.setOneShotPreviewCallback(this);

if (mCameraDevice != null) {

mCameraDevice.addCallbackBuffer(mPreviewBuffers[mCurrentPreviewBuffer]);

mCurrentPreviewBuffer = 1 - mCurrentPreviewBuffer;

//Log.d(TAG, "TA3D - mPreviewCallback Size of mPreviewBuffer[0]: " + mPreviewBuffers[0].length);

//Log.d(TAG, "TA3D - mPreviewCallback Size of mPreviewBuffer[1]: " + mPreviewBuffers[1].length);

//Log.d(TAG, "##### TA3D -mCameraDevice is not NULL\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ");

}

/\*else

{

Log.d(TAG, "##### TA3D -mCameraDevice is NULL\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ");

}\*/

}

};

/\*

private class CameraPreviewCallback implements CameraManager.CameraPreviewDataCallback {

@Override

public void onPreviewFrame(byte[] data, CameraManager.CameraProxy camera){

Log.d(TAG, "TA3D - onPreviewFrame");

TouchlessA3D.Rotate rotation = TouchlessA3D.Rotate.DO\_NOT\_ROTATE;

if (mTouchlessEngine != null) {

Log.d(TAG, "TA3D - mTouchlessEngine.handleImage");

mTouchlessEngine.handleImage(System.currentTimeMillis(), data,

rotation);

}

}

};

private CameraPreviewCallback mCameraPreviewCallback = new CameraPreviewCallback();

\*/

private void initializeTouchlessA3D(TouchlessA3D engine) {

//Log.d(TAG, "TA3D - initializeTouchlessA3D enter");

mTouchlessEngine = engine;

//mTouchlessEngine.registerGestureListener(TA3DThumbsUpPresence.TYPE, this);

//mTouchlessEngine.registerGestureListener(TA3DFacePresence.TYPE, this);//wenguangyu

// mTouchlessEngine.registerGestureListener(TA3DSwipe.TYPE, this);

setExtendedRange(true);

/\*

com.mediatek.camera.ICamera camera = mCameraDevice.getCamera();

android.hardware.Camera.Parameters params = camera.getParameters();

android.hardware.Camera.Size s = params.getPreviewSize();

int[] fpsRange = new int[2];

params.getPreviewFpsRange(fpsRange);

getMaxFpsRange(params.getSupportedPreviewFpsRange(), fpsRange);

Log.d(TAG, "TA3D Setting preview fps range: " +

fpsRange[android.hardware.Camera.Parameters.PREVIEW\_FPS\_MIN\_INDEX] + " - " +

fpsRange[android.hardware.Camera.Parameters.PREVIEW\_FPS\_MAX\_INDEX]);

params.setPreviewFpsRange(

fpsRange[android.hardware.Camera.Parameters.PREVIEW\_FPS\_MIN\_INDEX],

fpsRange[android.hardware.Camera.Parameters.PREVIEW\_FPS\_MAX\_INDEX]);

params.setPictureFormat(ImageFormat.JPEG);

camera.setParameters(params);

\*/

//com.mediatek.camera.ICamera camera = mCameraDevice.getCamera();

//android.hardware.Camera.Parameters params = camera.getParameters();

mTouchlessPreviewSize = mParameters.getPreviewSize();//params.getPreviewSize();

int actualBpp = ImageFormat.getBitsPerPixel(mParameters.getPreviewFormat())/8;

//Log.d(TAG, "TA3D -initializeTouchlessA3D mParameters getPreviewFormat: " + mParameters.getPreviewFormat());

//Log.d(TAG, "TA3D -initializeTouchlessA3D mParameters getPictureFormat: " + mParameters.getPictureFormat());

//Log.d(TAG, "TA3D -mParameters: " + mParameters.getPreviewFormat());

//Log.d(TAG, "TA3D -actualBpp: " + actualBpp);

//Log.d(TAG, "TA3D -actualBpp: " + actualBpp);

int bpp = 4;//ImageFormat.getBitsPerPixel(camera.getParameters().getPreviewFormat()) / 8;

//Log.d(TAG, "TA3D -bpp: " + bpp); // 4

mPreviewBuffers = new byte[2][mTouchlessPreviewSize.width \* mTouchlessPreviewSize.height \* bpp];

// mPreviewTempBuffer = new byte[mTouchlessPreviewSize.width \* mTouchlessPreviewSize.height \* bpp];

//Log.d(TAG, "TA3D - width: " + mTouchlessPreviewSize.width + ", height: " + mTouchlessPreviewSize.height);

//Log.d(TAG, "TA3D - Size of mPreviewBuffer; " + mPreviewBuffers[0].length);

if(mCameraDevice != null)

{

mCameraDevice.addCallbackBuffer(mPreviewBuffers[0]);

mCameraDevice.addCallbackBuffer(mPreviewBuffers[1]);

//mCameraDevice.setOneShotPreviewCallback(mPreviewCallback);

//mCameraDevice.setPreviewCallback(mPreviewCallback);

//camera.setPreviewCallback(mPreviewCallback);

mCameraDevice.getCamera().setPreviewCallbackWithBuffer(mPreviewCallback);

}

//Log.d(TAG, "TA3D - initializeTouchlessA3D exit");

}

private static void getMaxFpsRange(List<int[]> ranges, int[] fpsRange) {

int minIndex = android.hardware.Camera.Parameters.PREVIEW\_FPS\_MIN\_INDEX;

int maxIndex = android.hardware.Camera.Parameters.PREVIEW\_FPS\_MAX\_INDEX;

for (int[] range : ranges) {

if (range[minIndex] > fpsRange[minIndex] ||

(range[minIndex] >= fpsRange[minIndex] &&

range[maxIndex] > fpsRange[maxIndex])) {

fpsRange[minIndex] = range[minIndex];

fpsRange[maxIndex] = range[maxIndex];

}

}

}

public void setExtendedRange(boolean enabled) {

if (mTouchlessEngine != null) {

mTouchlessEngine.setParameter(

TouchlessA3D.Parameters.EXTENDED\_RANGE, enabled ? 1 : 0);

}

}

private class TouchlessTask extends AsyncTask<Void, Void, TouchlessA3D> {

private int errorResId;

@Override

protected TouchlessA3D doInBackground(Void... v) {

TouchlessA3D engine = null;

if (!isCancelled()) {

try {

//Log.d(TAG, "TA3D - engine = new TouchlessA3D()");

/\*if(mCameraDevice != null)

{

com.mediatek.camera.ICamera camera = mCameraDevice.getCamera();

android.hardware.Camera.Parameters params = camera.getParameters();

android.hardware.Camera.Size s = params.getPreviewSize();

}\*/

// Log.d(TAG, "TA3D -\*flipped\* ENGINE CREATE::: width: " + mPreviewFrameWidth + ", height: " + mPreviewFrameHeight);

Size paramsSize = mParameters.getPreviewSize();

Log.d(TAG, "TA3D - ENGINE CREATE::: mParameters width: " + paramsSize.width + ", height: " + paramsSize.height);

// engine = new TouchlessA3D(mPreviewFrameWidth, mPreviewFrameHeight);

// engine = new TouchlessA3D(mPreviewFrameHeight, mPreviewFrameWidth);

engine = new TouchlessA3D(paramsSize.width, paramsSize.height);

engine.registerGestureListener(TA3DThumbsUpPresence.TYPE, mCameraGestureCallback);

engine.registerGestureListener(TA3DVSignPresence.TYPE, mCameraGestureCallback);

if(mObjectStabilityTracker != null)

{

mObjectStabilityTracker.registerStabilityListener(mStabilityListener);

}

setExtendedRange(true);

} catch (LicenseNotValidException lnve) {

//Log.d(TAG, "TA3D - LicenseNotValidException");

} catch (LicenseServerUnavailableException lsue) {

//Log.d(TAG, "TA3D - LicenseServerUnavailableException");

}

}

else

{

//Log.d(TAG, "TA3D - TouchlessA3D is Cancel");

return null;

}

return engine;

}

protected void onPostExecute(TouchlessA3D engine) {

if (engine != null) {

//Log.d(TAG, "TA3D - onPostExecute");

initializeTouchlessA3D(engine);

} else {

//Log.d(TAG, "TA3D - engine == null");

}

}

}

private final TouchlessSelfTimerListener mSelfTimerListener = new TouchlessSelfTimerListener() {

@Override

public void onTimerStart() { }

@Override

public void onTimerStop() {

return;

}

@Override

public void onTimerTimeout() {

if(mCameraActor != null)

{

mPresenceType = PresenceType.Presence\_Normal;

mCameraActor.doTouchlessShutterClick();

return;

}

}

};

private final StabilityListener mStabilityListener = new StabilityListener() {

public void onStableObject(int objectType, int objectId) {

Log.i(TAG, "TA3D" + String.format("Stable object of type [%d] with id [%d]", objectType, objectId));

/\* int selfTimer = Integer.valueOf(getSelfTimer());

if(selfTimer > 3000)

{

mCameraActor.getTouchlessSelfTimerManager().setSelfTimerDuration(getSelfTimer());

}

else

{

mCameraActor.getTouchlessSelfTimerManager().setSelfTimerDuration("3000");

}

mCameraActor.getTouchlessSelfTimerManager().setTimerListener(mSelfTimerListener);\*/

/\* if (objectType== TA3DVSignPresence.TYPE && checkHandShutter() && mShowCameraAppView) {

mPresenceType = PresenceType.Presence\_TA3DVSign;

mCameraActor.getTouchlessSelfTimerManager().SetPresenceType(mPresenceType);

keepScreenOnAwhile();

//Log.d(TAG, "\*\*\*\*\*\*\*\*\*TA3D TA3DVSignPresence Do Photo Action \*\*\*\*\*\*\*\*\*\*\*\*\*");

mCameraActor.doTouchlessShutterClick();

}

else\*/ if (objectType == TA3DThumbsUpPresence.TYPE && checkHandShutter() && mShowCameraAppView) {

mPresenceType = PresenceType.Presence\_TA3DThumbsUp;

mCameraActor.getTouchlessSelfTimerManager().SetPresenceType(mPresenceType);

keepScreenOnAwhile();

mCameraActor.setTouchlessMovementValue(false);

//Log.d(TAG, "\*\*\*\*\*\*\*\*\*TA3D TA3DThumbsUpPresence Do Photo Action \*\*\*\*\*\*\*\*\*\*\*\*\*");

mCameraActor.doTouchlessShutterClick();

}

}

};

private final TA3DGestureListener mCameraGestureCallback = new TA3DGestureListener() {

public void onTA3DGesture(TA3DGesture gesture) {

//Log.i(TAG, "TA3D - onTA3DGesture");

/\*

if (gesture.getType() == TA3DFacePresence.TYPE) {

// do face things here

TA3DFacePresence face = (TA3DFacePresence) gesture;

//Log.d(TAG, "TA3D face Up " + face.getAction());

//Log.d(TAG, "TA3D face Up face.getObjectId() = " + face.getObjectId());

keepScreenOnAwhile();

if(face.getAction() == Action.START)

{

mTouchlessFaces++;

}

else if(face.getAction() == Action.END)

{

mTouchlessFaces--;

}

}else if (gesture.getType() == TA3DThumbsUpPresence.TYPE) {

TA3DThumbsUpPresence thumb = (TA3DThumbsUpPresence) gesture;

//Log.d(TAG, "TA3D thumb Up " + thumb.getAction());

//Log.d(TAG, "TA3D thumb Up thumb.getObjectId() = " + thumb.getObjectId());

keepScreenOnAwhile();

if(thumb.getAction() == Action.START)

{

mTouchlessThumbs++;

}

else if(thumb.getAction() == Action.END)

{

mTouchlessThumbs--;

}

}

//Log.d(TAG, "TA3D Thumbs&&&faces Up mTouchlessThumbs = " + mTouchlessThumbs + " mTouchlessFaces = " + mTouchlessFaces );

\*/

//if (gesture.getType() == TA3DThumbsUpPresence.TYPE && checkHandShutter() && mShowCameraAppView) {

//if (gesture.getType() == TA3DThumbsUpPresence.TYPE && checkHandShutter() && mShowCameraAppView) {

//TA3DThumbsUpPresence thumb = (TA3DThumbsUpPresence) gesture;

//Log.d(TAG, "TA3D thumb Up thumb.getObjectId() = " + thumb.getObjectId());

/\* if (gesture.getType() == TA3DVSignPresence.TYPE && checkHandShutter() && mShowCameraAppView) {

keepScreenOnAwhile();

TA3DVSignPresence Vsign = (TA3DVSignPresence)gesture;

Log.d(TAG, "TA3D - TA3DVSignPresence, Vsign.getAction() = " + Vsign.getAction());

Log.d(TAG, "TA3D - TA3DVSignPresence, Vsign.getobjectId() = " + Vsign.getObjectId() + "++++++++++++");

mPresenceType = PresenceType.Presence\_TA3DVSign;

mClockView.SetPresenceType(mPresenceType);

if (Vsign.getAction() == Action.START){

Log.d(TAG, "TA3D \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*TA3DVSignPresence START\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

//mClockView.setupClassMembers(this);

mClockView.setCountdownCompletedListener(Camera.this);

//mClockView.setVisibility(View.VISIBLE);

// mClockView.requestLayout();

mClockView.StartCountAnimation(Camera.this);

}

}

else if (gesture.getType() == TA3DThumbsUpPresence.TYPE && checkHandShutter() && mShowCameraAppView) {

}

//Log.d(TAG, "TA3D - Thumbs the Timer is stop " );

}\*/

if(mObjectStabilityTracker != null)

{

mObjectStabilityTracker.update((TA3DObjectPresence)gesture);//xiaowen

}

int selfTimer = Integer.valueOf(getSelfTimer());

if(selfTimer > 3000)

{

mCameraActor.getTouchlessSelfTimerManager().setSelfTimerDuration(getSelfTimer());

}

else

{

mCameraActor.getTouchlessSelfTimerManager().setSelfTimerDuration("3000");

}

mCameraActor.getTouchlessSelfTimerManager().setTimerListener(mSelfTimerListener);

if (gesture.getType() == TA3DVSignPresence.TYPE && checkHandShutter() && mShowCameraAppView) {

TA3DVSignPresence Vsign = (TA3DVSignPresence)gesture;

mPresenceType = PresenceType.Presence\_TA3DVSign;

mCameraActor.getTouchlessSelfTimerManager().SetPresenceType(mPresenceType);

//Log.d(TAG, "TA3D V sign" + Vsign.getAction());

keepScreenOnAwhile();

if (Vsign.getAction() == Action.START || (mCameraId==0 && mCameraActor.getTouchlessMovementValue() && Vsign.getAction() == Action.MOVEMENT)){//|| Vsign.getAction() == Action.MOVEMENT){

//Log.d(TAG, "\*\*\*\*\*\*\*\*\*TA3D TA3DVSignPresence Do Photo Action \*\*\*\*\*\*\*\*\*\*\*\*\*");

//mCameraActor.setTouchlessMovementValue(false);

//Log.d(TAG, "TA3D - Thumbs mCameraActor is NULL " );

mCameraActor.doTouchlessShutterClick();

}

}

/\*else if (gesture.getType() == TA3DThumbsUpPresence.TYPE && checkHandShutter() && mShowCameraAppView) {

TA3DThumbsUpPresence Thumbsup = (TA3DThumbsUpPresence)gesture;

//Log.d(TAG, "TA3D Thumbsup" + Thumbsup.getAction());

mPresenceType = PresenceType.Presence\_TA3DThumbsUp;

mCameraActor.getTouchlessSelfTimerManager().SetPresenceType(mPresenceType);

keepScreenOnAwhile();

if (Thumbsup.getAction() == Action.START || (mCameraId==0 && mCameraActor.getTouchlessMovementValue() && Thumbsup.getAction() == Action.MOVEMENT) ){//|| Thumbsup.getAction() == Action.MOVEMENT){

mCameraActor.setTouchlessMovementValue(false);

//Log.d(TAG, "\*\*\*\*\*\*\*\*\*TA3D TA3DThumbsUpPresence Do Photo Action \*\*\*\*\*\*\*\*\*\*\*\*\*");

mCameraActor.doTouchlessShutterClick();

}

//}

}\*/

}

} ;

public Vibrator getVibrator() {

return mVibrator;

}

//ysfeagle add for frame begin

public void tinno\_frame\_init(){

mhandler = new Handler();

circlelayout = (FrameLayout)findViewById(R.id.circle\_layout);

circle\_main = (SUI\_CircleLayout)findViewById(R.id.main\_circle\_layout);

circle\_main.SUI\_setOnItemSelectedListener(this);

first\_circle\_meun =(SUI\_firstCircleLayout)findViewById(R.id.first\_circle\_layout);

//first\_circle\_meun.SUI\_firstcircle\_setOnItemSelectedListener(this);

tinno\_Pre\_Camera\_ui\_close();

}

public void tinno\_Pre\_Camera\_ui\_open(){

Log.d(TAG, "tinno\_Pre\_Camera\_ui\_open:getCurrentMode# "+getCurrentMode());

if(is\_open\_precapture() ==true)

{

circlelayout.setVisibility(View.VISIBLE);

circle\_main.setVisibility(View.VISIBLE);

first\_circle\_meun.setVisibility(View.VISIBLE);

first\_circle\_meun.get\_SharedPreferences\_again();

mShutterManager.iscanadjust=true;

mShutterManager.tinno\_adjust\_postion(40);//Util.PREVIEW\_4\_3\_RECT\_ALIGN);

mhandler.removeCallbacks(mRunable);

mhandler.postDelayed(mRunable, 20000);

dismissProgress();

}

}

public void tinno\_Pre\_Camera\_ui\_close(){

Log.d(TAG, "tinno\_Pre\_Camera\_ui\_close:# ");

circlelayout.setVisibility(View.GONE);

circle\_main.setVisibility(View.GONE);

first\_circle\_meun.setVisibility(View.GONE);

first\_circle\_meun.Save\_SharedPreferences\_again();

mShutterManager.tinno\_reset\_postion();

mShutterManager.iscanadjust=false;

}

private void tinno\_Pre\_Camera\_reset\_state()

{

if(is\_open\_precapture()==true){

//mShutterManager.mPhotoShutter.setImageResource(R.drawable.btn\_photo);

tinno\_Pre\_Camera\_ui\_close();

first\_circle\_meun.ResetSharedPreferences();

}

}

public Runnable mRunable =new Runnable(){

public void run() {

tinno\_Pre\_Camera\_ui\_close();

}

};

public void onItemSelected(View view, int position, long id, String name) {

mhandler.removeCallbacks(mRunable);

first\_circle\_meun.show\_screen(position);

mhandler.postDelayed(mRunable, 20000);

}

public void tinno\_frame\_ui\_open(){

}

public void tinno\_frame\_ui\_close(){

}

private void tinno\_frame\_get\_key(){

}

public boolean isFullScreencheck() {

if (/\*16\_9\*/"1.7778".equals(mPreferences.

getString(CameraSettings.KEY\_PICTURE\_RATIO, Util.DEFAULT\_PREVIEW\_RATIO))) {

return true;

}

return false;

}

@Override

public View makeView() {

ImageView i = new ImageView(this);

//i.setBackgroundColor(0xFF000000);

i.setScaleType(ImageView.ScaleType.FIT\_CENTER);

i.setLayoutParams(new ImageSwitcher.LayoutParams(

LayoutParams.WRAP\_CONTENT, LayoutParams.WRAP\_CONTENT));

return i;

}

@Override

public void onPanelClosed(Panel panel) {

// TODO Auto-generated method stub

Log.d(TAG, "onPanelClosed:# "+current\_index);

SharedPreferences.Editor editor = mSharedPreference.edit();

editor.putInt(CAMERA\_FRAME\_KEY, current\_index);

editor.commit();

}

@Override

public void onPanelOpened(Panel panel) {

}

public boolean is\_open\_precapture()

{

Log.d(TAG, "is\_open\_precapture:getCurrentMode# "+getCurrentMode());

if(getCurrentMode()==ModePicker.MODE\_PROFESSIONAL){

return true;

}else{

return false;

}

}

public boolean is\_open\_frame() {

String preframe =mSettingChecker.getSettingCurrentValue(SettingChecker.ROW\_SETTING\_CAMERA\_PIC\_FRAME);

if("on".equals(preframe)){

return true;

}else{

return false;

}

}

//ysfeagle add end

private void initializeAfterPreview() {

long start = System.currentTimeMillis();

callResumableBegin();

mModePicker.setCurrentMode(mCameraActor.getMode());

//Here we don't use setViewState(VIEW\_STATE\_NORMAL) for that:

//We will show remaining first time and setViewState() do more thing than we need.

mShutterManager.show();

//mSettingManager.show();

mPickerManager.show();

if (isNonePickIntent() || isStereo3DImageCaptureIntent()) {

mModePicker.show(); //will do when parameters ready

mThumbnailManager.show();

View cover = findViewById(com.android.gallery3d.R.id.gl\_root\_cover);

cover.setVisibility(View.GONE);

addIdleHandler();//why no flag to disable it after checked.

long stop = System.currentTimeMillis();

Log.v(TAG, "initializeAfterPreview() consume:" + (stop - start));

}

/\*

private IAppGuideExt.OnGuideFinishListener onFinishListener = new IAppGuideExt.OnGuideFinishListener() {

public void onGuideFinish() {

setAppGuideFinished(true);

// show toast after appGuide finish

if (isNonePickIntent() || isImageCaptureIntent()) {

//mVoiceManager.startUpdateVoiceState();

showRemainingAways();

}

}

};

\*/

//Here should be lightweight functions!!!

private void initializeCommonManagers() {

mSettingChecker = new SettingChecker(this);

mReviewManager = new ReviewManager(this);

mShutterManager = new ShutterManager(this);

mModePicker = new ModePicker(this);

mSettingManager = new SettingManager(this);

mLittleIManager = new LittleIManager(this);

//For tablet

mThumbnailManager = new ThumbnailManager(this);

mPickerManager = new PickerManager(this);

mIndicatorManager = new IndicatorManager(Camera.this);

//mRemainingManager = new RemainingManager(this);

mInfoManager = new InfoManager(this);

mTouchlessGuideManager = new TouchlessGuideManager(this);//wenguangyu

mCsSpeedManager = new ShowCSSpeedManager(this);//add for CS Speed

mZoomManager = new ZoomManager(this);

mFileSaver = new FileSaver(this);

mRotateDialog = new RotateDialog(this);

mRotateProgress = new RotateProgress(this);

mVoiceManager = new VoiceManager(this);

mWfdLocal = new WfdManagerLocal(this);

mFrameManager = new FrameManager(this);

//mRecorderSwitcher.setEnabled(false);

recordCommonManagers();

mModePicker.setListener(mModeChangedListener);

//mRecorderSwitcher.setListener(mModeChangedListener);

mSettingManager.setListener(mSettingListener);

mPickerManager.setListener(mPickerListener);

mThumbnailManager.setFileSaver(mFileSaver);

mWfdLocal.addListener(mWfdListener);

mPowerManager = (PowerManager) getSystemService(Context.POWER\_SERVICE);

Log.v(TAG, "getSystemService,mPowerManager =" + mPowerManager);

}

}