文档编号：

**Android平台相机对焦与测光分离**

软件设计说明书

编写日期： 2014 年 12 月 10 日

文档修订记录

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 版本号 | 日期 | 修订说明 | 变更人 | 批准日期 | 批准人 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

目　　录

[Android平台相机对焦与测光分离 1](#_Toc408216868)

[1 概述 1](#_Toc408216869)

[2 任务概述程序总体设计及结构设计 1](#_Toc408216870)

[2.1 设计思想 1](#_Toc408216871)

[2.2 程序结构 2](#_Toc408216872)

[2.2.1 相机对焦与测光分离软件运行流程图： 2](#_Toc408216873)

[2.2.2 类结构图 3](#_Toc408216874)

[2.3 运行环境 3](#_Toc408216875)

[3 详细设计与实现 3](#_Toc408216876)

[3.1 外部接口 4](#_Toc408216877)

[3.2 类设计 4](#_Toc408216878)

[3.2.1 DragLayer类设计 4](#_Toc408216879)

[3.2.2 RotateLayout类设计 5](#_Toc408216880)

[3.3 MeteringLayout、FocusLayout类设计 7](#_Toc408216881)

[4 运行设计 9](#_Toc408216882)

[4.1 运行控制 9](#_Toc408216883)

[5 维护设计 9](#_Toc408216884)

1. 概述

Anroid平台移动终端同质化越来越严重，满足用户的个性化需求、开发更加炫酷的界面、使用更加简单方便的操作方式，提升用户操作体验成为破局的关键。而Camera渐渐成为各厂家必争的重要功能，开发出更好用、效果更好的相机应用，能为旗下产品增色不少。

鉴于此，提出相机对焦与测光分离项目：

1. 拖动测光框可实时看到感光效果；
2. 拖动对焦框随意选择对焦点；
3. 可分开设置对焦点和测光点（传统只能使用同一个点）
4. 任务概述程序总体设计及结构设计
   1. 设计思想

1、不同状态时的显示效果：

a、实现一个继承ViewGroup的可随着屏幕旋转而自适应旋转的RotateLayout类

b、对焦框与测光框都为RotateLayout的子类，并实现FocusIndicator接口类来显示不同状态

2、实时拖动效果：

a、实现一个继承自FrameLayout的DragLayer类，做为将要拖动的对焦框和测光框的父容器，并定义DragInterface接口，以便拖动对焦框和测光框时控制拖动时和拖动放手后需要实现的功能。

b、拖动时不移动对焦框和测光框，隐藏它们并实时地在canvas上显示通过它们获取的bitmap。放手时不显示这个bitmap，计算对焦框和测光框此时的位置，将它们移到该位置并显示。

3、拖动时实时设置对焦与测光参数：

a、拖动测光框时，实时计算测光框的位置，并使用上次成功对焦后的对焦点，然后将参数设置下去。

b、拖动对焦框时，不进行参数设置，等放手时计算对焦框的位置，然后将对焦位置参数设置下去，并记录此时的对焦位置以便下次拖动测光框时使用。

* 1. 程序结构
     1. 相机对焦与测光分离软件运行流程图：



* + 1. 类结构图



MeteringLayout（测光框）和FocusLayout（对焦框）继承自RotateLayout，并实现DragLayer定义的接口类DragInerface来响应用户的拖拽动作，它们都被包含在父容器DragLayer里面。包含在DragLayer里面的控件可以被拖拽，拖拽的过程中DragLayer通过调用接口来调用对焦框或测光框要实现的功能。

* 1. 运行环境

【简要地说明对本系统的运行环境（包括硬件环境和支持环境）的规定。】

软件环境：android2.2以上

1. 详细设计与实现
   1. 外部接口

3D解锁需要实现draglayer的拖拽接口，实时响应用户的拖拽动作：

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);

}

* 1. 类设计
     1. DragLayer类设计

public class DragLayer extends FrameLayout{

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) {

//开始拖拽

invalidate();

}

@Override

protected void dispatchDraw(Canvas canvas) {

super.dispatchDraw(canvas);

//绘制draglayer，吸附效果在这个函数中处理

}

@Override

public boolean dispatchKeyEvent(KeyEvent event) {

//发送按键事件

return mDragging || super.dispatchKeyEvent(event);

}

@Override

public boolean onInterceptTouchEvent(MotionEvent ev) {

//拦截触摸事件处理

return mDragging;

}

private void endDrag(float x, float y) {

//放弃触摸停止拖拽时的动作处理

}

@Override

public boolean onTouchEvent(MotionEvent ev) {//处理触摸响应事件

return true;

}

}

DragLayer类是一个FrameLayout，它用来内嵌在一个需要实现拖拽动作的父布局中来实现拖拽动作。

* + 1. RotateLayout类设计

public class RotateLayout extends ViewGroup implements Rotatable {

private static final String TAG = "RotateLayout";

private OnSizeChangedListener mListener;

private int mOrientation;

protected View mChild;

public RotateLayout(Context context, AttributeSet attrs) {

super(context, attrs);

setBackgroundResource(android.R.color.transparent);

//构造函数

}

/\*\* A callback to be invoked when the preview frame's size changes. \*/

public interface OnSizeChangedListener {

void onSizeChanged(int width, int height);

}

@Override

protected void onFinishInflate() {

mChild = getChildAt(0);

mChild.setPivotX(0);

mChild.setPivotY(0);

}

@Override

protected void onLayout(

boolean change, int left, int top, int right, int bottom) {

}

@Override

protected void onMeasure(int widthSpec, int heightSpec) {

//

}

@Override

public boolean shouldDelayChildPressedState() {

return false;

}

// Rotate the view counter-clockwise

@Override

public void setOrientation(int orientation, boolean animation) {

//设置方向

}

public int getOrientation() {

return mOrientation;

}

public void setOnSizeChangedListener(OnSizeChangedListener listener) {

mListener = listener;

}

@Override

protected void onSizeChanged(int w, int h, int oldw, int oldh) {

}

RotateLayout继承自ViewGroup类，根据手机屏幕旋转自动跟随旋转适配。FocusLayout和MeteringLayout继承这个类并实现FocusIndicator的接口函数来实现对状态的控制。

* 1. MeteringLayout、FocusLayout类设计

public class MeteringIndicatorRotateLayout extends RotateLayout implements FocusIndicator {

private Camera mContext;

public MeteringIndicatorRotateLayout(Context context, AttributeSet attrs) {

super(context, attrs);

//构造函数

}

private void setDrawable(int resid) {

//设置背景

}

@Override

public void showStart() {

// FocusIndicator接口函数，开始显示

}

@Override

public void showSuccess(boolean timeout) {

// FocusIndicator接口函数，显示成功

}

@Override

public void showFail(boolean timeout) {

// FocusIndicator接口函数，显示失败

}

@Override

public void clear() {

// FocusIndicator接口函数，清除状态

}

private class EndAction implements Runnable {//内部线程类，控制定时不可见

@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;

}

}

@Override

public boolean gatherTransparentRegion(Region region) {

return true;

}

}

public class FocusIndicatorRotateLayout extends RotateLayout implements FocusIndicator {

private Camera mContext;

public FocusIndicatorRotateLayout(Context context, AttributeSet attrs) {

super(context, attrs);

//构造函数

}

private void setDrawable(int resid) {

//设置背景

}

@Override

public void showStart() {

// FocusIndicator接口函数，开始显示

}

@Override

public void showSuccess(boolean timeout) {

// FocusIndicator接口函数，显示成功

}

@Override

public void showFail(boolean timeout) {

// FocusIndicator接口函数，显示失败

}

@Override

public void clear() {

// FocusIndicator接口函数，清除状态

}

private class EndAction implements Runnable {//内部线程类，控制定时不可见

@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;

}

}

@Override

public boolean gatherTransparentRegion(Region region) {

return true;

}

}

1. 运行设计
   1. 运行控制

软件运行时界面友好，做到界面运行流畅，对焦状态一目了然，测光效果实时可见，用户操作方便简单。

1. 维护设计

软件设计时已考虑到后期多项目共用的移植性问题，界面显示与逻辑控制分开处理，方便后续该功能的移植及维护。

软件设计考虑到后期多项目公用的移植性问题，界面在各自类中处理，同时逻辑控制中使用上层公开的api接口函数，剥离了与硬件相关耦合，方便后续功能的移植及维护。