Android翻页入门

Zhibin Wang[a]

[a]*School of Computer and Software Engineering, Xihua University, Chengdu, China*

Abstract

欲整理和实现Android端的翻页效果实现，并想将之整理打包成为一个成熟的第三方插件。不知道会用多少时间来实现这个功能，虽然网上已经有现成的项目，以及对之的解析，但本人从学习的角度来说，不适合直接拷贝集成别人的库来使用，应该抱着学习的心态来学习和整理，并加入自己的想法。我想这应该是接下来我应该做的事情。简单记录下今日时间：2021年4月18日 22:31:57。算是一个比较正式的自学研究历程的笔记。

Keywords: Android翻页实现, 项目源码阅读, 客户端开发

1. Introduction

在图1中，我们标明了涉及到翻页所使用的A、B、C三面，以及这三面对应的绘制区域。具体的可参考链接`https://juejin.cn/post/6844903529715335182`。根据我对它的理解，这里做一个简单的剖析。首先为了简单实现这个静态的效果，所使用到的技术有：

1. 自定义View；
2. 自定义路径；
3. canvas中的路径裁剪；
4. 二阶贝塞尔曲线；

在实现前，首先需要明白一个问题就是：翻页这个实现其实看似有这三个页面，其实本质有一个View根布局文件，也就是在一个自定义View中实现。而A、B、C三面是这个页面中的三个部分，也就是我们需要找到各面

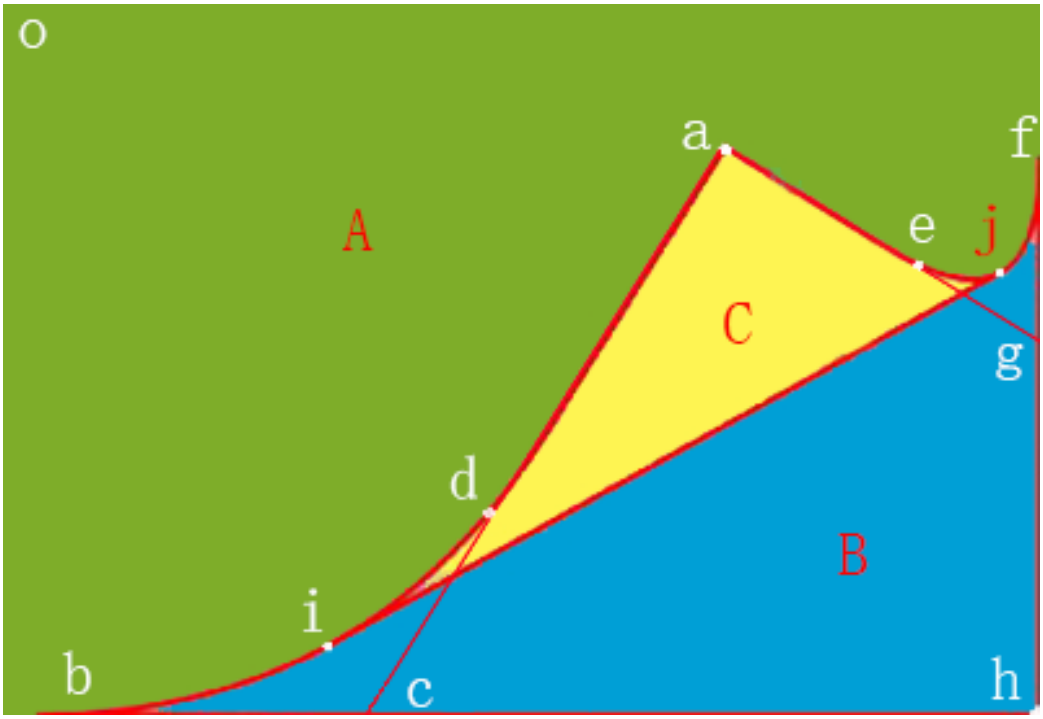

图 1: 翻页效果图，来源于互联网。

对应的绘制区域，然后使用canvas.drawBitmap绘图函数将之放置到对应的区域即可。也就是说，其实也就是需要在一个View中找到A、B、C三个区域的Path，然后将每个区域都绘制对应Path的Bitmap图像，并在这个区域中放置对应的文本和背景颜色即可。

根据图1，我们知道这三个区域可以使用图中的这些关键点来进行标识，也就是说只要找到了图中的这些点的坐标，那么我们就可以在一个View中绘制出对应的三个区域A、B、C。那么，我们首先需要解决的问题就是如何来找到图中标识三个区域的关键点。按照大佬的博客**Android自定义View——从零开始实现书籍翻页效果（一）**[1]以及**Android 实现书籍翻**

[1]https://juejin.cn/post/6844903529715335182

页效果—-原理篇[2] 我们知道可以通过添加一些参考线，利用已知点信息，以及相似性关系可以推测出其余点的信息。那么，这里也一样，学习下这个过程。在图2中，我们添加了一些参考线以及对应的点，接下来就来记录下这个计算过程。

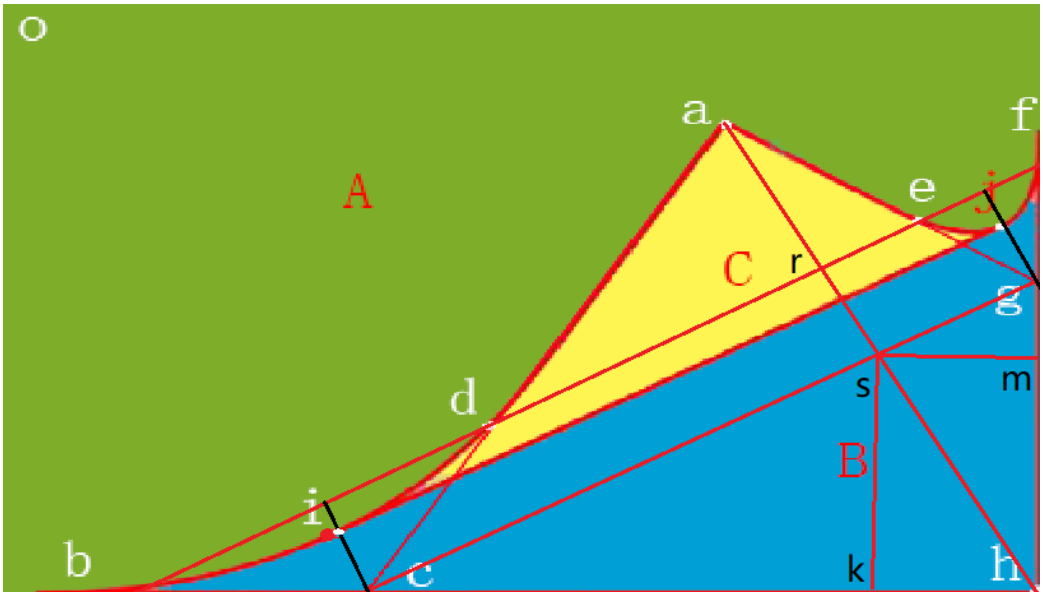

图 2: 为翻页效果图添加辅助信息，便于计算。

2. Calculate the Point Coordinate

图2中，简单的形式化任务为根据已知点和预设的信息进行目标点的坐标计算。

已知点有：o、a、h。

目标点有：b、c、d、e、f、g、i、j。

预设信息：cg垂直ah，且s为ah的中心。也就是cg是ah的垂直平分线。

记已知点的坐标表示为：$a(x, y)$，$h(W, H)$

[2] https://blog.csdn.net/hmg25/article/details/6306479

记任意两点a，b之间的直线长度表示为：$l_{a,b}$
记任意点a的坐标中，横纵坐标表示分别为：(a_x, a_y)
那么，s点的坐标为：

$$s(\frac{a_x + h_x}{2}, \frac{a_y + h_y}{2})$$

由于三角形csk相似于三角形shk，那么：

$$\frac{l_{s,k}}{l_{c,k}} = \frac{l_{k,h}}{l_{s,k}}$$

那么，$l_{c,k} = \frac{l_{s,k}^2}{l_{k,h}}$ ，且$l_{s,k} = H - s_y$，$l_{k,h} = W - s_x$，那么$l_{c,k} = \frac{(H-s_y)^2}{W-s_x}$。故而，c点的横坐标长度为$W - l_{c,k} - l_{k,h} = s_x - \frac{(H-s_y)^2}{W-s_x}$，c点坐标为：

$$c(s_x - \frac{(H - s_y)^2}{W - s_x}, H)$$

同理，由于三角形hsm相似于三角形sgm，那么：

$$\frac{l_{s,m}}{l_{m,h}} = \frac{l_{g,m}}{l_{s,m}}$$

那么，$l_{g,m} = \frac{l_{s,m}^2}{l_{m,h}}$，且$l_{s,m} = W - s_x$，$l_{m,h} = l_{s,k} = H - s_y$，那么$l_{g,m} = \frac{(W-s_x)^2}{H-s_y}$，故而g点的纵坐标长度为$H - l_{g,m} - l_{m,h} = s_y - \frac{(W-s_x)^2}{H-s_y}$，g点坐标为：

$$g(W, s_y - \frac{(W - s_x)^2}{H - s_y})$$

设r为线段as的中点，那么三角形brh相似于三角形csh，可以得到相似性关系为：

$$\frac{l_{b,h}}{l_{c,h}} = \frac{l_{r,h}}{l_{s,h}} = \frac{3}{2}$$

那么，$l_{b,h} = \frac{3}{2}l_{c,h} = \frac{3}{2}(l_{c,k} + l_{k,h}) = \frac{3}{2}(l_{c,k} + l_{s,m})$，即：

$$l_{b,h} = \frac{3}{2}(\frac{(H - s_y)^2}{W - s_x} + W - s_x)$$

那么，b点坐标为：

$$b(W - l_{b,h}, H) = (\frac{1}{2}(3s_x - W - 3\frac{(H - s_y)^2}{W - s_x}), H)$$

由三角形rfh相似于三角形sgh，那么可以得到相似性关系：

$$\frac{l_{f,h}}{l_{g,h}} = \frac{l_{r,h}}{l_{s,h}} = \frac{3}{2}$$

那么，$l_{f,h} = \frac{3}{2} l_{g,h} = \frac{3}{2}(l_{g,m} + l_{m,h}) = \frac{3}{2}(l_{g,m} + l_{s,k})$，即：

$$l_{f,h} = \frac{3}{2}(\frac{(W - s_x)^2}{H - s_y} + H - s_y)$$

那么，f点坐标为：

$$f(W, H - l_{f,h}) = (W, \frac{1}{2}(3s_y - H - 3\frac{(W - s_x)^2}{H - s_y}))$$

直线ac与直线ag分别交直线bf于点d和e，故而可通过直线求交点得到d和e的坐标表示。通过构建辅助线ic和jg，由于垂直关系可以得到i。

直线方程为$y = kx + b$，设已知两点(x_1, y_1)，(x_2, y_2)，那么：

$$k = \frac{y_2 - y_1}{x_2 - x_1}$$

$$b = y_1 - \frac{y_2 - y_1}{x_2 - x_1} x_1$$

对于直线ac和ag，a的坐标为(x, y)，c的坐标为$c(s_x - \frac{(H-s_y)^2}{W-s_x}, H)$，g的坐标为$g(W, s_y - \frac{(W-s_x)^2}{H-s_y})$，记未知数为X和Y，那么可以计算到直线ac：

$$Y_1 = (\frac{H - y}{s_x - \frac{(H-s_y)^2}{W-s_x} - x})X_1 + (y - ((\frac{H - y}{s_x - \frac{(H-s_y)^2}{W-s_x} - x}))x)$$

同理，直线ag的表示为：

$$Y_2 = (\frac{s_y - \frac{(W-s_x)^2}{H-s_y} - y}{W - x})X_2 + (y - (\frac{s_y - \frac{(W-s_x)^2}{H-s_y} - y}{W - x})x)$$

但是，观察到上面式子比较复杂，那么可以直接简化为：

$$k_1 = \frac{c_y - a_y}{c_x - a_x}$$

$$b_1 = a_y - k_1 a_x$$

$$Y_1 = k_1 X_1 + b_1 = \frac{c_y - a_y}{c_x - a_x} X_1 + a_y - \frac{c_y - a_y}{c_x - a_x} a_x$$

类似的，对于Y_2有：

$$Y_2 = \frac{g_y - a_y}{g_x - a_x} X_2 + a_y - \frac{g_y - a_y}{g_x - a_x} a_x$$

同理，对于直线bf，可计算到直线bf的表示：

$$Y_3 = \frac{b_y - f_y}{b_x - f_x} X_3 + f_y - \frac{b_y - f_y}{b_x - f_x} f_x$$

那么，可以计算出Y_1与Y_3以及Y_2与Y_3对应的交点d、e的坐标表示。

$$x = \frac{b_1 - b_2}{k_2 - k_1}$$

$$y = k_1 \frac{b_1 - b_2}{k_2 - k_1} + b_1$$

即d点坐标：

$$d(\frac{b_1 - b_3}{k_3 - k_1}, k_1 \frac{b_1 - b_3}{k_3 - k_1} + b_1)$$

e点坐标：

$$e(\frac{b_2 - b_3}{k_3 - k_2}, k_2 \frac{b_2 - b_3}{k_3 - k_2} + b_2)$$

对于ij两点，由于ic、gj直线和bf垂直，可以求的其斜率$k_4 = \frac{-1}{k_3}$
不妨假设i是其中点，那么我们需要求到ic与bf的交点，设交点为p_1、p_2：
直线ic的表示为：

$$Y_4 = \frac{-1}{k_3} X_4 + c_y - k_4 c_x$$

求的Y_3和Y_4的交点为：

$$p_1(\frac{b_4 - b_3}{k_3 - k_4}, k_4 \frac{b_4 - b_3}{k_3 - k_4} + b_4)$$

那么，可以得到i的坐标表示为：

$$i(\frac{p_{1x} + c_x}{2}, \frac{p_{1y} + c_y}{2})$$

同理，对于直线gj，斜率$k_5 = k_4 = \frac{-1}{k_3}$

$$Y_5 = \frac{-1}{k_3}X_5 + g_y - k_5 g_x$$

求的Y_3和Y_4的交点为：

$$p_2(\frac{b_5 - b_3}{k_3 - k_5}, k_5\frac{b_5 - b_3}{k_3 - k_5} + b_5)$$

那么，可以得到j的坐标表示为：

$$j(\frac{p_{2x} + g_x}{2}, \frac{p_{2y} + g_y}{2})$$

然后，根据这个我们可以创建一个计算的工具类。

2.1. Implementation

前面，我们对绘制过程中的点的坐标进行了计算表达的推导，这里就可以根据上面的公式进行点具体值的计算。这里写一个工具类MyPoint，如下：

```
public class MyPoint {
    public float x;
    public float y;

    public MyPoint(){}
    public MyPoint(float x, float y){
        this.x = x;
        this.y = y;
    }
    @Override
    public String toString() {
        return "MyPoint{" + "x=" + x + ", y=" + y + '}';
    }
}
```

然后，对于怎么计算待求的点的坐标，不妨创建一个工具类来处理，如下：

```
import android.util.Log;
import com.example.pojo.MyPoint;
import java.util.HashMap;
import java.util.Map;

public class CalcPoints {

    // 传入数据
    private MyPoint a, h;
    // 目标
    private MyPoint b, c, d, e, f, g, i, j;

    private Map<String, MyPoint> points;

    public Map<String, MyPoint> calcuation(){
        MyPoint s = new MyPoint((a.x + h.x) / 2, (a.y +
            h.y) / 2);
        c.x = s.x - (h.y - s.y) * (h.y - s.y) / (h.x - s.x);
        c.y = h.y;
        g.x = h.x;
        g.y = s.y - (h.x - s.x) * (h.x - s.x) / (h.y - s.y);
        b.x = 0.5f * (3*s.x - h.x - 3*(h.y - s.y)*(h.y -
            s.y) / (h.x - s.x));
        if(b.x <= 0){
            return null;
        }
        b.y = h.y;
        f.x = h.x;
```

```
f.y = 0.5f * (3*s.y - h.y - 3 * (h.x - s.x) * (h.x
    - s.x) / (h.y - s.y));

float k_1 = (c.y - a.y) / (c.x - a.x);
float b_1 = a.y - k_1 * a.x;

float k_2 = (g.y - a.y) / (g.x - a.x);
float b_2 = a.y - k_2 * a.x;

float k_3 = (b.y - f.y) / (b.x - f.x);
float b_3 = f.y - k_3 * f.x;

Log.e("k_3", String.valueOf(k_3));
Log.e("k_1", String.valueOf(k_1));
d.x = (b_1 - b_3) / (k_3 - k_1);
d.y = k_1*d.x+b_1;

e.x = (b_2 - b_3) / (k_3 - k_2);
e.y = k_2 * e.x + b_2;

float k_4 = -1.0f / k_3;
float b_4 = c.y - k_4 * c.x;
MyPoint p_1 = new MyPoint();
p_1.x = (b_3 - b_4) / (k_4 - k_3);
p_1.y = k_3 * p_1.x + b_3;

i.x = (p_1.x + c.x) / 2;
i.y = (p_1.y + c.y) / 2;

float k_5 = -1.0f / k_3;
float b_5 = g.y - k_5 * g.x;
```

```
        MyPoint p_2 = new MyPoint();
        p_2.x = (b_3 - b_5) / (k_5 - k_3);
        p_2.y = k_3 * p_2.x + b_3;

        j.x = (p_2.x + g.x) / 2;
        j.y = (p_2.y + g.y) / 2;


        points.put("b", b);
        points.put("c", c);
        points.put("d", d);
        points.put("e", e);
        points.put("f", f);
        points.put("g", g);
        points.put("i", i);
        points.put("j", j);


        for (String ss : points.keySet()) {
            Log.e(ss, points.get(ss).toString());
        }

        return points;
    }

    public CalcPoints(MyPoint a, MyPoint h){
        this.h = h;
        this.a = a;
        b = new MyPoint();
        c = new MyPoint();
        d = new MyPoint();
```

```
        e = new MyPoint();
        f = new MyPoint();
        g = new MyPoint();
        i = new MyPoint();
        j = new MyPoint();
        this.points = new HashMap<>();
    }
}
```

通过上面的代码可以得到我们绘制A、B、C三面的所有点的坐标，然后就可以开始进入到对应的A、B、C三面的绘制。

3. Draw Panel

绘制这种比较复杂的路径图像，需要使用Android的Path类来解决。对于A面，我们定义路径如下：

1. 从左下角出发，沿bidaejf最后回到o；
2. 对于直线，可以简单的使用path.lineTo绘制；
3. 对于曲线，这里使用二阶贝塞尔曲线，控制点分别是c和g；

那么，对应的绘制路径为：

```
private Path getPathA(){
    path.reset();
    path.lineTo(0, viewHeight);
    path.lineTo(points.get("b").x,points.get("b").y);
    path.quadTo(points.get("c").x,points.get("c").y,
        points.get("d").x, points.get("d").y);
    path.lineTo(a.x, a.y);
    path.lineTo(points.get("e").x, points.get("e").y);
    path.quadTo(points.get("g").x,points.get("g").y,
        points.get("f").x, points.get("f").y);
```

```
    path.lineTo(viewWidth,0);
    path.close();
    return path;
}
```

如下图：

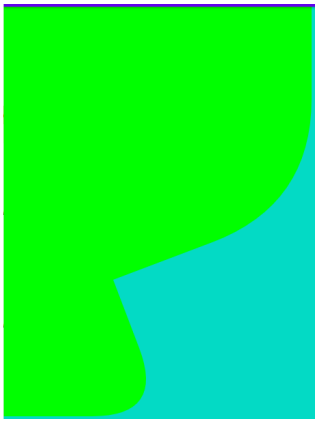

图 3: A面绘制，颜色为绿色。BC面为预设的背景色

对于B面，也就在图2中的蓝色部分的绘制。但是，这个面比较不好绘制，因为包含两段二阶贝塞尔曲线的一半，一个直线。难点在就在于曲线的绘制。故而考虑将B面暂定为图3的背景色的部分，也就是图2中的B和C两个部分。那么，此时可以用二阶贝塞尔曲线来进行绘制，绘制代码如下：

```
private Path getPathA(){
    path.reset();
    path.lineTo(0, viewHeight);
    path.lineTo(points.get("b").x,points.get("b").y);
    path.quadTo(points.get("c").x,points.get("c").y,
        points.get("d").x, points.get("d").y);
    path.lineTo(a.x,a.y);
    path.lineTo(points.get("e").x, points.get("e").y);
```

```
    path.quadTo(points.get("g").x,points.get("g").y,
        points.get("f").x, points.get("f").y);
    path.lineTo(viewWidth,viewHeight);
    path.lineTo(0, viewHeight);
    path.close();
    return path;
}
```

只绘制B面，如下图所示：

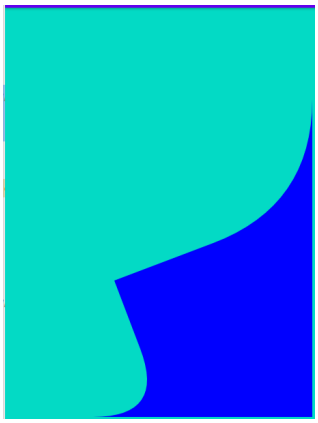

图 4: B面绘制，颜色为蓝色，AC面为预设的背景色。

对于C面，这部分内容我们在B中绘制过了，也就是在刚刚绘制的B面中包含了目标C面。且那会所说的一半的二阶贝塞尔曲线难以绘制问题依然存在。故而，这里考虑使用Android中的裁剪技术，也就是我们绘制直线构成的区域：idaeji。

绘制代码如下：

```
private Path getPathC(){
    path.reset();
    path.moveTo(points.get("i").x, points.get("i").y);
    path.lineTo(points.get("d").x, points.get("d").y);
```

```
    path.lineTo(a.x,a.y);
    path.lineTo(points.get("e").x, points.get("e").y);
    path.lineTo(points.get("j").x, points.get("j").y);
    path.close();
    return path;
}
```

然后，将这个区域和刚刚的B面进行交集计算，即可得到我们所需要的C面。由于Android的裁剪技术对应canvas.clipPath，也就是作用对象是画布对象。也就是：

```
@Override
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);
    Path path = getPathC();

    bitmap1 = Bitmap.createBitmap((int) viewWidth, (int)
        viewHeight, Bitmap.Config.ARGB_8888);
    Canvas bitmapCanvas1 = new Canvas(bitmap1);
    bitmapCanvas1.drawPath(getPathA(), mAPaint); // GREEN A
    canvas.drawBitmap(bitmap1,0,0,null);

    bitmap2 = Bitmap.createBitmap((int) viewWidth, (int)
        viewHeight, Bitmap.Config.ARGB_8888);
    Canvas bitmapCanvas2 = new Canvas(bitmap2);
    bitmapCanvas2.drawPath(getPathB(), mBPaint); // RED B
    // 指定裁剪区域为B+C
    canvas.clipPath(getPathB(), Region.Op.INTERSECT);
    canvas.drawBitmap(bitmap2,0,0,null);

    bitmap3 = Bitmap.createBitmap((int) viewWidth, (int)
```

```
        viewHeight, Bitmap.Config.ARGB_8888);
    Canvas bitmapCanvas3 = new Canvas(bitmap3);
    bitmapCanvas3.drawPath(getPathC(), mCPaint); // YELLOW C
    // 指定裁剪区域为，取和上次的裁剪区域的交集C
    canvas.clipPath(getPathC(), Region.Op.INTERSECT);
    canvas.drawBitmap(bitmap3,0,0,null);
}
```

效果：

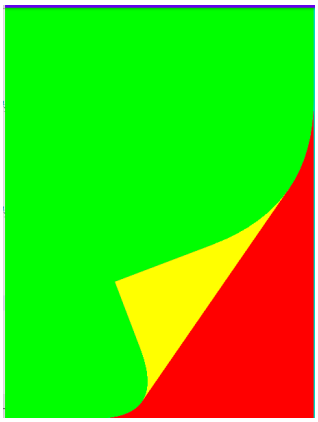

图 5: 全部面绘制

这部分自定义View的完整代码如下：

```
public class Learn extends View {

    private Paint mAPaint;
    private Paint mBPaint;
    private Paint mCPaint;

    private MyPoint a, h;
    private Map<String, MyPoint> points;
```

```
private Path path;
private Bitmap bitmap1;
private Bitmap bitmap2;
private Bitmap bitmap3;

private int viewWidth = 450;
private int viewHeight = 600;

public Learn(Context context, @Nullable AttributeSet
    attrs) {
    super(context, attrs);
    init();
}

private void init(){
    mAPaint = new Paint();
    mAPaint.setColor(Color.GREEN);
    mAPaint.setAntiAlias(true);//设置抗锯齿

    mBPaint = new Paint();
    mBPaint.setColor(Color.RED);
    mBPaint.setAntiAlias(true);//设置抗锯齿

    mCPaint = new Paint();
    mCPaint.setColor(Color.YELLOW);
    mCPaint.setAntiAlias(true);//设置抗锯齿

    a = new MyPoint(160, 400);
    h = new MyPoint(viewWidth, viewHeight);
```

```
        points = new CalcPoints(a, h).calcuation();


        path = new Path();
    }



    @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        Path path = getPathC();

        bitmap1 = Bitmap.createBitmap((int) viewWidth,
            (int) viewHeight, Bitmap.Config.ARGB_8888);
        Canvas bitmapCanvas1 = new Canvas(bitmap1);
        bitmapCanvas1.drawPath(getPathA(), mAPaint); //
            GREEN A
        canvas.drawBitmap(bitmap1,0,0,null);

        bitmap2 = Bitmap.createBitmap((int) viewWidth,
            (int) viewHeight, Bitmap.Config.ARGB_8888);
        Canvas bitmapCanvas2 = new Canvas(bitmap2);
        bitmapCanvas2.drawPath(getPathB(), mBPaint); // RED
            B
        canvas.clipPath(getPathB(), Region.Op.INTERSECT);
            // 指定裁剪区域
            为B+C
        canvas.drawBitmap(bitmap2,0,0,null);

        bitmap3 = Bitmap.createBitmap((int) viewWidth,
```

```
            (int) viewHeight, Bitmap.Config.ARGB_8888);
        Canvas bitmapCanvas3 = new Canvas(bitmap3);
        bitmapCanvas3.drawPath(getPathC(), mCPaint); //
            YELLOW C
        canvas.clipPath(getPathC(), Region.Op.INTERSECT);
            // 指定裁剪区域为, 取和上次的裁剪区域的交
            集C
        canvas.drawBitmap(bitmap3,0,0,null);
    }

    private Path getPathA(){
        path.reset();
        path.lineTo(0, viewHeight);
        path.lineTo(points.get("b").x,points.get("b").y);
        path.quadTo(points.get("c").x,points.get("c").y,
            points.get("d").x, points.get("d").y);
        path.lineTo(a.x, a.y);
        path.lineTo(points.get("e").x, points.get("e").y);
        path.quadTo(points.get("g").x,points.get("g").y,
            points.get("f").x, points.get("f").y);
        path.lineTo(viewWidth,0);
        path.close();
        return path;
    }


    /**
    * 绘制默认的界面
    * @return
    */
    private Path getPathDefault(){
```

```
        path.reset();
        path.lineTo(0, viewHeight);
        path.lineTo(viewWidth,viewHeight);
        path.lineTo(viewWidth,0);
        path.close();
        return path;
    }

    private Path getPathB(){
        path.reset();
        path.lineTo(0, viewHeight);
        path.lineTo(points.get("b").x,points.get("b").y);
        path.quadTo(points.get("c").x,points.get("c").y,
            points.get("d").x, points.get("d").y);
        path.lineTo(a.x,a.y);
        path.lineTo(points.get("e").x, points.get("e").y);
        path.quadTo(points.get("g").x,points.get("g").y,
            points.get("f").x, points.get("f").y);
        path.lineTo(viewWidth,viewHeight);
        path.lineTo(0, viewHeight);
        path.close();
        return path;
    }

    /**
    * 绘制区域C
    * @return
    */
    private Path getPathC(){
        path.reset();
        path.moveTo(points.get("i").x, points.get("i").y);
```

```
        path.lineTo(points.get("d").x, points.get("d").y);
        path.lineTo(a.x,a.y);
        path.lineTo(points.get("e").x, points.get("e").y);
        path.lineTo(points.get("j").x, points.get("j").y);
        path.close();
        return path;
    }

}
```

至此，基础的原理和基础的实现已经呈上了。那么接下来就是实现一些判断和对应的时间处理函数。由于主要功能这里已经讲解完毕，故而就考虑不再继续写这些后续的文档。故而考虑将这个文档命名问Android翻页入门。