[**Android图片处理（Matrix,ColorMatrix）**](http://www.cnblogs.com/leon19870907/archive/2011/03/09/1978065.html)

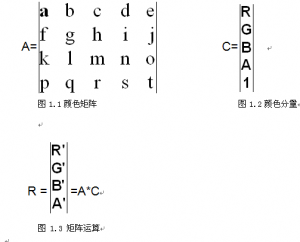
在编程中有时候需要对图片做特殊的处理，比如将图片做出黑白的，或者老照片的效果，有时候还要对图片进行变换，以拉伸，扭曲等等。

这些效果在android中有很好的支持，通过颜色矩阵（ColorMatrix）和坐标变换矩阵（Matrix）可以完美的做出上面的所说的效果。

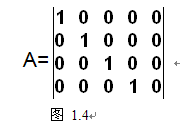
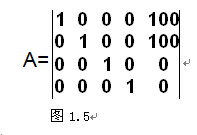
下面将分别介绍这两个矩阵的用法和相关的函数。

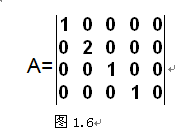
**颜色矩阵**  
android中可以通过颜色矩阵（ColorMatrix类）方面的操作颜色，颜色矩阵是一个5x4 的矩阵（如图1.1）

可以用来方面的修改图片中RGBA各分量的值，颜色矩阵以一维数组的方式存储如下：  
 [ a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t ]  
他通过RGBA四个通道来直接操作对应颜色，如果会使用Photoshop就会知道有时处理图片通过控制RGBA各颜色通道来做出特殊的效果。

这个矩阵对颜色的作用计算方式如1.3示：  
  
[](http://dev.chinamobile.com/cmdn/wiki/uploads/201004/12713816310wSz1syZ.png)  
矩阵的运算规则是矩阵A的一行乘以矩阵C的一列作为矩阵R的一行，

C矩阵是图片中包含的ARGB信息，R矩阵是用颜色矩阵应用于C之后的新的颜色分量，运算结果如下：  
   
R' = a\*R + b\*G + c\*B + d\*A + e;  
G' = f\*R + g\*G + h\*B + i\*A + j;  
B' = k\*R + l\*G + m\*B + n\*A + o;  
A' = p\*R + q\*G + r\*B + s\*A + t;  
   
颜色矩阵并不是看上去那么深奥，其实需要使用的参数很少，而且很有规律第一行决定红色第二行决定绿色

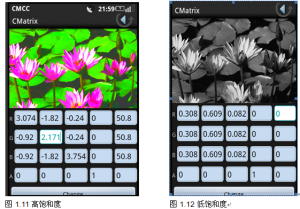
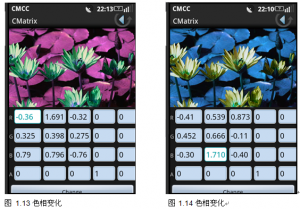
第三行决定蓝色，第四行决定了透明度，第五列是颜色的偏移量。下面是一个实际中使用的颜色矩阵。  
  
[](http://dev.chinamobile.com/cmdn/wiki/uploads/201004/1271381666y9yeN31S.png)  
如果把这个矩阵作用于各颜色分量的话，R=A\*C，计算后会发现，各个颜色分量实际上没有任何的改变(R'=R G'=G B'=B A'=A)。  
[](http://dev.chinamobile.com/cmdn/wiki/uploads/201004/1271381693ZrrIsDUa.png)  
图1.5所示矩阵计算后会发现红色分量增加100，绿色分量增加100，

这样的效果就是图片偏黄，因为红色和绿色混合后得到黄色，黄色增加了100，图片当然就偏黄了。  
[](http://dev.chinamobile.com/cmdn/wiki/uploads/201004/1271381725l0fGeSkP.png)  
改变各颜色分量不仅可以通过修改第5列的颜色偏移量也可如上面矩阵所示将对应的颜色值乘以一个倍数，直接放大。

上图1.6是将绿色分量乘以2变为原来的2倍。相信读者至此已经明白了如何通过颜色矩阵来改变各颜色分量。

下面编写一段代码来，通过调整颜色矩阵来获得不同的颜色效果，JavaCode如下：  
复制到剪贴板  Java代码

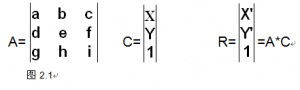
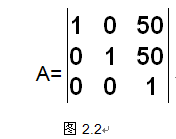
1 CMatrix类：   
2  public class CMatrix extends Activity {   
3   
4 private Button change;   
5 private EditText [] et=new EditText[20];   
6 private float []carray=new float[20];   
7 private MyImage sv;   
8 @Override   
9 public void onCreate(Bundle savedInstanceState) {   
10 super.onCreate(savedInstanceState);   
11 setContentView(R.layout.main);   
12   
13 change=(Button)findViewById(R.id.set);   
14 sv=(MyImage)findViewById(R.id.MyImage);   
15   
16 for(int i=0;i<20;i++){   
17   
18 et[i]=(EditText)findViewById(R.id.indexa+i);   
19 carray[i]=Float.valueOf(et[i].getText().toString());   
20 }   
21   
22 change.setOnClickListener(l);   
23 }   
24   
25 private Button.OnClickListener l=new Button.OnClickListener(){   
26   
27 @Override   
28 public void onClick(View arg0) {   
29 // TODO Auto-generated method stub   
30   getValues();   
31 sv.setValues(carray);   
32 sv.invalidate();   
33 }   
34   
35 };   
36 public void getValues(){   
37 for(int i=0;i<20;i++){   
38   
39 carray[i]=Float.valueOf(et[i].getText().toString());   
40 }   
41   
42 }   
43   
44   
45 }   
46 MyImage类继承自View类：   
47 public class MyImage extends View {   
48 private Paint mPaint = new Paint(Paint.ANTI\_ALIAS\_FLAG);   
49 private Bitmap mBitmap;   
50 private float [] array=new float[20];   
51   
52 private float mAngle;   
53   
54 public MyImage(Context context,AttributeSet attrs) {   
55 super(context,attrs);   
56   
57 mBitmap = BitmapFactory.decodeResource(context.getResources(),   
58 R.drawable.test);   
59 invalidate();   
60 }   
61   
62   
63 public void setValues(float [] a){   
64 for(int i=0;i<20;i++){   
65 array[i]=a[i];   
66 }   
67   
68 }   
69   
70 @Override protected void onDraw(Canvas canvas) {   
71 Paint paint = mPaint;   
72   
73   
74   
75 paint.setColorFilter(null);   
76 canvas.drawBitmap(mBitmap, 0, 0, paint);   
77   
78 ColorMatrix cm = new ColorMatrix();   
79 //设置颜色矩阵   
80 cm.set(array);   
81 //颜色滤镜，将颜色矩阵应用于图片   
82 paint.setColorFilter(new ColorMatrixColorFilter(cm));   
83 //绘图   
84 canvas.drawBitmap(mBitmap, 0, 0, paint);   
85 Log.i("CMatrix", "--------->onDraw");   
86   
87   
88 }   
89   
90 }

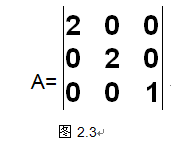
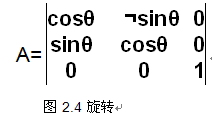
CMatrix类主要负责接收颜色矩阵的设置和重绘，没有要说的。MyImage类中进行绘图工作，首先设置颜色矩阵cm.set(..)从一维数组中读取数据20个数据给颜色矩阵赋值，paint.setColorFilter(..)设置颜色滤镜，然后绘图，效果就出来了（这个过程和PS差不多）如下： *[](http://dev.chinamobile.com/cmdn/wiki/uploads/201004/12713818099NMQz4Nl.png)  
[](http://dev.chinamobile.com/cmdn/wiki/uploads/201004/1271381825XJGiUUGn.png)  
[](http://dev.chinamobile.com/cmdn/wiki/uploads/201004/1271381842U0m4uTUx.png)  
[](http://dev.chinamobile.com/cmdn/wiki/uploads/201004/12713818647kb99802.png)*看到这里，相信大家对颜色矩阵的作用已经有了一个直观的感受，现在也可以尝试做一个照片特效的软件。

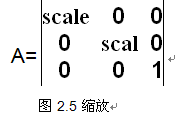
但是各种效果并不能让用户手动调节颜色矩阵，这里需要计算公式，由于本人并不是做图形软件的也不能提供，可以参考这个链接：  
<http://www.adobe.com/devnet/flash/articles/matrix_transformations/ColorMatrixDemo.swf>

**坐标变换矩阵**  
坐标变换矩阵是一个3\*3的矩阵如图2.1，用来对图形进行坐标变化，将原来的坐标点转移到新的坐标点，

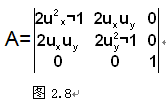
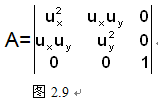
因为一个图片是有点阵和每一点上的颜色信息组成的，所以对坐标的变换，就是对每一点进行搬移形成新的图片。

具体的说图形的放大缩小，移动，旋转，透视，扭曲这些效果都可以用此矩阵来完成。 *[](http://dev.chinamobile.com/cmdn/wiki/uploads/201004/1271381907wVeuzTD2.png)*这个矩阵的作用是对坐标x,y进行变换计算结果如下：  
x'=a\*x+b\*y+c  
y'=d\*x+e\*y+f  
通常情况下g=h=0,这样使1=0\*x+0\*y+1恒成立。和颜色矩阵一样，坐标变换矩阵真正使用的参数很少也很有规律。 *[](http://dev.chinamobile.com/cmdn/wiki/uploads/201004/1271381929fvyejlxl.png)*上图就是一个坐标变换矩阵的简单例子，计算后发现x'=x+50,y'=y+50.

可见图片的每一点都在x和y方向上平移到了（50，50）点处，这种效果就是平移效果，将图片转移到了（50，50）处。 *[](http://dev.chinamobile.com/cmdn/wiki/uploads/201004/1271381954eNEbYoEI.png)*计算上面得矩阵x'=2\*x,y‘=2\*y.经过颜色矩阵和上面转移效果学习，相信读者可以明白这个矩阵的作用了，这个矩阵对图片进行了放大，具体的说是放大了二倍。  
下面将介绍几种常用的变换矩阵：  
1.      旋转 *[](http://dev.chinamobile.com/cmdn/wiki/uploads/201004/1271381984t6X7xcj1.png)*

绕原点逆时针旋转θ度角的变换公式是 x' = xcosθ − ysinθ 与 y。' = xsinθ +    ycosθ  
2.      缩放 *[](http://dev.chinamobile.com/cmdn/wiki/uploads/201004/1271382016TeOFkZ5i.png)*

变换后长宽分别放大x'=scale\*x;y'=scale\*y.  
3.         切变  
[](http://dev.chinamobile.com/cmdn/wiki/uploads/201004/1271382046nPrSJD3C.png)

4.         反射  
[](http://dev.chinamobile.com/cmdn/wiki/uploads/201004/1271382180iZqxVK85.png)  
( , )单位向量  
5.         正投影  
[](http://dev.chinamobile.com/cmdn/wiki/uploads/201004/12713822147vGAWixg.png)  
( , )单位向量  
   
上面的各种效果也可以叠加在一起，既矩阵的组合变换，可以用矩阵乘法实现之，如：R=B(A\*C)=(B\*A)C,注意一点就是B\*A和A\*B一般是不等的。

下面将编一个小程序，通过控制坐标变换矩阵来达到控制图形的目的，JavaCode如下：  
  
复制到剪贴板  Java代码

1

CooMatrix类：   
2   
3public class CooMatrix extends Activity {   
4   
5 private Button change;   
6 private EditText [] et=new EditText[9];   
7 private float []carray=new float[9];   
8 private MyImage sv;   
9 /\*\* Called when the activity is first created. \*/   
10 @Override   
11 public void onCreate(Bundle savedInstanceState) {   
12 super.onCreate(savedInstanceState);   
13 setContentView(R.layout.main);   
14   
15 change=(Button)findViewById(R.id.set);   
16 sv=(MyImage)findViewById(R.id.MyImage);   
17   
18 for(int i=0;i<9;i++){   
19   
20 et[i]=(EditText)findViewById(R.id.indexa+i);   
21 carray[i]=Float.valueOf(et[i].getText().toString());   
22   
23 }   
24   
25 change.setOnClickListener(l);   
26   
27   
28 }   
29   
30 private Button.OnClickListener l=new Button.OnClickListener(){   
31   
32 @Override   
33 public void onClick(View arg0) {   
34 // TODO Auto-generated method stub   
35 getValues();   
36 sv.setValues(carray);   
37 sv.invalidate();   
38 }   
39   
40 };   
41 public void getValues(){   
42 for(int i=0;i<9;i++){   
43   
44 carray[i]=Float.valueOf(et[i].getText().toString());   
45 }   
46   
47 }   
48   
49   
50}   
51MyImage类继承自View类：   
52public class MyImage extends View {   
53 private Paint mPaint = new Paint(Paint.ANTI\_ALIAS\_FLAG);   
54 private Bitmap mBitmap;   
55 private float [] array=new float[9];   
56   
57   
58   
59 public MyImage(Context context,AttributeSet attrs) {   
60 super(context,attrs);   
61   
62 mBitmap = BitmapFactory.decodeResource(context.getResources(),   
63 R.drawable.ic\_launcher\_android);   
64 invalidate();   
65 }   
66   
67   
68 public void setValues(float [] a){   
69 for(int i=0;i<9;i++){   
70 array[i]=a[i];   
71 }   
72   
73 }   
74   
75 @Override protected void onDraw(Canvas canvas) {   
76 Paint paint = mPaint;   
77 canvas.drawBitmap(mBitmap, 0, 0, paint);   
78 //new 一个坐标变换矩阵   
79 Matrix cm = new Matrix();   
80//为坐标变换矩阵设置响应的值   
81 cm.setValues(array);   
82//按照坐标变换矩阵的描述绘图   
83 canvas.drawBitmap(mBitmap, cm, paint);   
84 Log.i("CMatrix", "--------->onDraw");   
85   
86   
87 }   
88   
89

}

上面的代码中类CooMatrix用于接收用户输入的坐标变换矩阵参数，类MyImage接收参数，通过setValues()设置矩阵参数，然后Canvas调用drawBitmap绘图。效果如下：

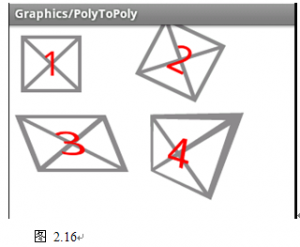
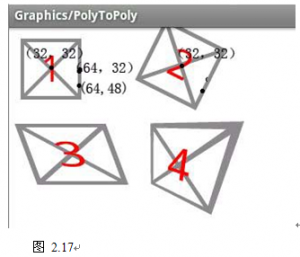
*[](http://dev.chinamobile.com/cmdn/wiki/uploads/201004/1271382361vPYddmGF.png)  
[](http://dev.chinamobile.com/cmdn/wiki/uploads/201004/12713823723z3c370V.png)  
[](http://dev.chinamobile.com/cmdn/wiki/uploads/201004/1271382385uV1mf9rz.png)*上面给出了用坐标变换矩阵做出的各种效果，用坐标变换矩阵可以方面的调节图形的各种效果，

但是我们看看Matrix类就可以发现，实际上，matrix类本身已经提供了许多类似的方法，我们只要调用，就可以了。  
   
setScale(float sx, float sy, float px, float py) 放大   
setSkew(float kx, float ky, float px, float py) 斜切   
setTranslate(float dx, float dy)                       平移   
setRotate(float degrees, float px, float py)    旋转   
   
上面的函数提供了基本的变换平移，放大，旋转，斜切。为了做出更复杂的变换，同时不必亲手去改动坐标变换矩阵，

Matrix类提供了许多Map方法，将原图形映射到目标点构成新的图形，

下面简述setPolyToPoly(float[] src, int srcIndex, float[] dst, int dstIndex, int pointCount) 的用法，希望起到举一反三的作用。

参数src和dst是分别存储了原图像的点和和指定的目标点的一维数组，数组中存储的坐标格式如下：  
 [x0, y0, x1, y1, x2,y2,...]  
   
这个个函数将src中的坐标映射到dst中的坐标，实现图像的变换。

具体的例子可以参考APIDemos里的PolyToPoly，我在这里就不再贴代码了，只讲一下函数是怎么变换图片的。下面是效果：  
 *[](http://dev.chinamobile.com/cmdn/wiki/uploads/201004/127138243999fcMBcL.png)*图中写1的是原图，写有2，3，4的是变换后的图形。现在分析2是怎么变换来的，变换的原坐标点和目的坐标点如下：  
src=new float[] { 32, 32, 64, 32 }  
dst=new float[] { 32, 32, 64, 48 } *[](http://dev.chinamobile.com/cmdn/wiki/uploads/201004/1271382482XgYceLnf.png)*从上图标示出的坐标看出原图的（32，32）映射到原图的（32，32），（64，32）映射到原图（64，48）这样的效果是图像放大了而且发生了旋转。这样的过程相当于（32，32）点不动，然后拉住图形（64，32）点并拉到（64，48）点处，这样图形必然会被拉伸放大并且发生旋转。最后用一个平移将图形移动到右边现在的位置。希望能够好好理解这一过程，下面的3，4图是同样的道理。