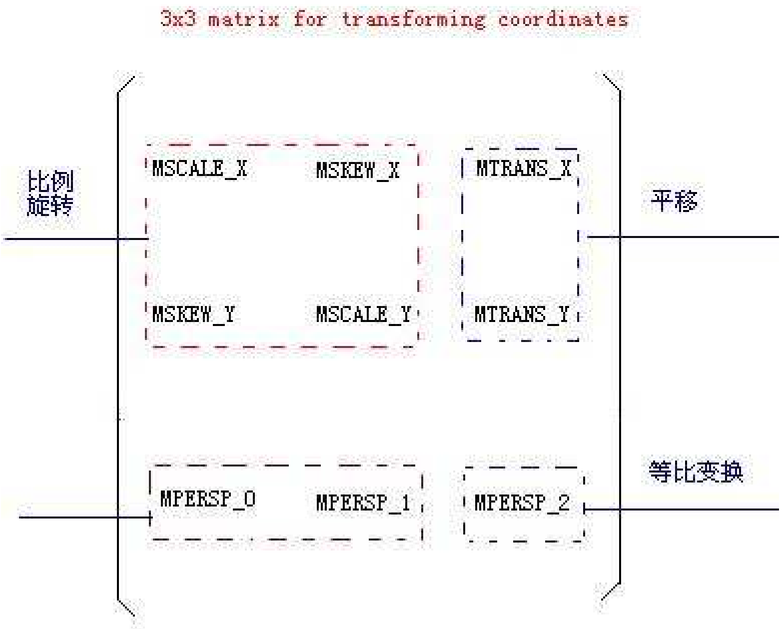
Matrix 学习——基础知识

以前在线性代数中学习了矩阵，对矩阵的基本运算有一些了解，前段时间在使用 GDI+的时候再次学习如何使用矩阵来变化图像，看了之后在这里总结说明。

首先大家看看下面这个 3 x 3 的矩阵，这个矩阵被分割成 4 部分。为什么分割成 4 部分，在后面详细说明。



首先给大家举个简单的例子：现设点 P0（x0， y0）进行平移后，移到 P（x，y），其中 x 方向的平移量为△x，y 方向的平移量为△y，那么，点 P（x，y）的坐标为：

x = x0 + △x

y = y0 + △y

采用矩阵表达上述如下：

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \triangle x \\ 0 & 1 & \triangle y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x0 \\ y0 \\ 1 \end{bmatrix}$$

上述也类似与图像的平移，通过上述矩阵我们发现，只需要修改矩阵右上角的 2 个元素就可以了。

我们回头看上述矩阵的划分：

矩阵区域	功能	
MSCALE_X　MSKEW_X MSKEW_Y　MSCALE_Y	Scale	缩放
	Skew/Shear	透视变换
	Rotate	旋转
MTRANS_X MTRANS_Y	Translate	平移
MPERSP_0　MPERSP_1		
MPERSP_2		

为了验证上面的功能划分，我们举个具体的例子：现设点 P0（x0 ，y0）进行平移后，移到 P（x，y），其中 x 放大 a 倍，y 放大 b 倍，

矩阵就是：$\begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{pmatrix}$，按照类似前面"平移"的方法就验证。

图像的旋转稍微复杂：现设点 P0（x0，　y0）旋转 θ 角后的对有点为 P（x，　y）。通过使用向量，我们得到如下：

$x0 = r \ \ \cos\alpha$

$y0 = r \ \ \sin\alpha$

$x = r \cos(\alpha-\theta) = x0 \cos\theta + y0 \sin\theta$

$y = r \text{ sia}(\alpha-\theta) = -x0 \sin\theta + y0 \cos\theta$

于是我们得到矩阵：$\begin{vmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix}$

如果图像围绕着某个点(a ，b)旋转呢？则先要将坐标平移到该点，再进行旋转，然后将旋转后的图像平移回到原来的坐标原点，在后面的篇幅中我们将详细介绍。

Matrix 学习——如何使用 Matrix

上一篇幅 Matrix 学习——基础知识，从高等数学方面给大家介绍了 Matrix，本篇幅我们就结合 Android 中的 android.graphics.Matrix 来具体说明，还记得我们前面说的图像旋转的矩阵：

$$\begin{vmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

从最简单的旋转 90 度的是：

$$\begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

在 android.graphics.Matrix 中有对应旋转的函数：

Matrix matrix = new Matrix();

matrix.setRotate(90);

Test.Log(MAXTRIX_TAG,”setRotate(90):%s” , matrix.toString());

查看运行后的矩阵的值（通过 Log 输出）：

```
dalvikvm    VM cleaning up
dalvikvm    LinearAlloc 0x0 used 637060 of 4194304 (15%)
Matrix      setRotate(90) Matrix{[0.0, -1.0, 0.0][1.0, 0.0, 0.0][0.0, 0.0, 1.0]}
dalvikvm    GC freed 3824 objects / 217344 bytes in 105ms
```

与上面的公式基本完全一样（android.graphics.Matrix 采用的是浮点数，而我们采用的整数）。

有了上面的例子，相信大家就可以亲自尝试了。通过上面的例子我们也发现，我们也可以直接来初始化矩阵，比如说要旋转 30 度：

$$A=\begin{vmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

$\sin 30 = 0.5 \quad \cos 30 = 0.866$

$$\begin{matrix} 0.866 & -0.5 & 0 \\ 0.5 & 0.866 & 0 \\ 0 & 0 & 1 \end{matrix}$$

$$\begin{pmatrix} 0.866 & -0.5 & 0 \\ 0.5 & 0.866 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

前面给大家介绍了这么多，下面我们开始介绍**图像的镜像**，分为 2 种：水平镜像、垂直镜像。先介绍如何实现垂直镜像，什么是垂直镜像就不详细说明。图像的垂直镜像变化也可以用矩阵变化的表示，设点 P0（x0 ，y0 ）进行镜像后的对应点为 P（x ，y ），图像的高度为 fHeight，宽度为 fWidth，原图像中的 P0（x0 ，y0 ）经过垂直镜像后的坐标变为（x0 ，fHeight- y0）；

x = x0

y = fHeight – y0

推导出相应的矩阵是：

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & fHeight \\ 0 & 0 & 1 \end{pmatrix}$$

```
final float f[] = {1.0F,0.0F,0.0F,0.0F,-1.0F,120.0F,0.0F,0.0F,1.0F};
Matrix matrix = new Matrix();
matrix.setValues(f);
```

按照上述方法运行后的结果：

至于水平镜像采用类似的方法，大家可以自己去试试吧。

实际上，使用下面的方式也可以实现垂直镜像：

```
Matrix matrix = new Matrix();
```

matrix.setScale (1.0，-1.0);
matrix.postTraslate(0, fHeight);

这就是我们将在后面的篇幅中详细说明。

Matrix 学习——图像的复合变化

Matrix 学习——基础知识篇幅中，我们留下一个话题：如果图像围绕着某个点 P(a，b)旋转，则先要将坐标系平移到该点，再进行旋转，然后将旋转后的图像平移回到原来的坐标原点。

我们需要 3 步：

1. **平移**——将坐标系平移到点 P(a，b)；
2. **旋转**——以原点为中心旋转图像；
3. **平移**——将旋转后的图像平移回到原来的坐标原点；

相比较前面说的图像的几何变化（基本的图像几何变化），这里需要**平移——旋转——平移**，这种需要多种图像的几何变化就叫做图像的复合变化。

设对给定的图像依次进行了基本变化 F1、F2、F3…..、Fn，它们的变化矩阵分别为 T1、T2、T3…..、Tn，图像复合变化的矩阵 T 可以表示为：T = TnTn-1…T1。

按照上面的原则，围绕着某个点(a，b)旋转 θ 的变化矩阵序列是：

T1	$\begin{matrix} 1 & 0 & -a \\ 0 & -1 & -b \\ 0 & 0 & 1 \end{matrix}$	T2	$\begin{matrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{matrix}$	T3	$\begin{matrix} 1 & 0 & a \\ 0 & -1 & b \\ 0 & 0 & 1 \end{matrix}$

$$T = T3 \cdot T2 \cdot T1 =$$

$$\begin{pmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -a \\ 0 & 1 & -b \\ 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} \cos\theta & -\sin\theta & -a\cos\theta + b\sin\theta + a \\ \sin\theta & \cos\theta & -a\sin\theta - b\cos\theta + b \\ 0 & 0 & 1 \end{pmatrix}$$

按照上面的公式，我们列举一个简单的例子：围绕（100，100）旋转 30 度(sin 30 = 0.5 ，cos 30 = 0.866)

```
float f[]= { 0.866F,  -0.5F, 63.4F,0.5F, 0.866F,-36.6F,0.0F,    0.0F,  1.0F };
matrix = new Matrix();
matrix.setValues(f);
```

旋转后的图像如下：

Android 为我们提供了更加简单的方法，如下：

```
Matrix matrix = new Matrix();
matrix.setRotate(30，100，100);
```

矩阵运行后的实际结果：

```
setRotate(30,100.000000,100.000000):Matrix{[0.8660254, -0.5, 63.39746]
                                           [0.5, 0.8660254, -36.60254]
                                           [0.0, 0.0, 1.0]}
```

与我们前面通过公式获取得到的矩阵完全一样。

在这里我们提供另外一种方法，也可以达到同样的效果：

```
float a = 100.0F,b = 100.0F;
matrix = new Matrix();
matrix.setTranslate(a,b);
matrix.preRotate(30);
matrix.preTranslate(-a,-b);
```

将在后面的篇幅中为大家详细解析

通过类似的方法，我们还可以得到：**相对点 P(a，b)的比例[sx,sy]变化矩阵**

$$\begin{pmatrix} sx & 0 & (1-sx)*a \\ 0 & sy & (1-sy)*b \\ 0 & 0 & 1 \end{pmatrix}$$

Matrix 学习——Preconcats or Postconcats?

从最基本的高等数学开始，Matrix 的基本操作包括：+、*。Matrix 的乘法不满足交换律，也就是说 A*B ≠B*A。

还有 2 种常见的矩阵：

Identity Matrix	单位矩阵 E（The elements on the main diagonal of the identity matrix are 1. All other elements of the identity matrix are 0.）
Inverse Matrix	逆矩阵 A*A-1=E

有了上面的基础，下面我们开始进入主题。由于矩阵不满足交换律，所以用矩阵 B 乘以矩阵 A，需要考虑是左乘（B*A），还是右乘（A*B）。在 Android 的 android.graphics.Matrix 中为我们提供了类似的方法，也就是我们本篇幅要说明的 Preconcats matrix 与 Postconcats matrix。下面我们还是通过具体的例子还说明：

```
final float f3[] = { 1.0F, 2.0F,3.0F,
                     4.0F, 5.0F,6.0F,
                     0.0F, 0.0F,1.0F
                   };
Matrix matrix1 = new Matrix();
matrix1.setValues(f3);
Test.Log(MAXTRIX_TAG,"matrix1:%s" , matrix1.toString());
matrix1.preRotate(90);
Test.Log(MAXTRIX_TAG,"matrix1.preRotate(90):%s" ,
                                            matrix1.toString());
Matrix matrix2 = new Matrix();
matrix2.setValues(f3);
Test.Log(MAXTRIX_TAG,"matrix2:%s" , matrix2.toString());
matrix2.postRotate(90);
Test.Log(MAXTRIX_TAG,"matrix2.postRotate(90):%s" ,
                                             matrix2.toString());
```

通过输出的信息，我们分析其运行过程如下：

```
1461   Matrix      matrix1:Matrix{[1.0, 2.0, 3.0]
                                  [4.0, 5.0, 6.0]
                                  [0.0, 0.0, 1.0]}
1461   Matrix      matrix1.preRotate(90):Matrix{[2.0, -1.0, 3.0]
                                                [5.0, -4.0, 6.0]
                                                [0.0, 0.0, 1.0]}
```

$$\overset{\text{matrix}}{\begin{pmatrix} 1.0F, & 2.0F, & 3.0F, \\ 4.0F, & 5.0F, & 6.0F, \\ 0.0F, & 0.0F, & 1.0F \end{pmatrix}} \overset{\text{Rotate(90)}}{\begin{pmatrix} 0.0F, & -1.0F, & 0.0F, \\ 1.0F, & 0.0F, & 0.0F, \\ 0.0F, & 0.0F, & 1.0F \end{pmatrix}} = \overset{\text{matrix1.preRotate(90)}}{\begin{pmatrix} 2.0F, & -1.0F, & 3.0F, \\ 5.0F, & -4.0F, & 6.0F, \\ 0.0F, & 0.0F, & 1.0F \end{pmatrix}}$$

```
1461   Matrix      matrix2:Matrix{[1.0, 2.0, 3.0]
                                  [4.0, 5.0, 6.0]
                                  [0.0, 0.0, 1.0]}
1461   Matrix      matrix2.postRotate(90):Matrix{[-4.0, -5.0, -6.0]
                                                 [1.0, 2.0, 3.0]
                                                 [0.0, 0.0, 1.0]}
```

$$\overset{\text{Rotate(90)}}{\begin{pmatrix} 0.0F, & -1.0F, & 0.0F, \\ 1.0F, & 0.0F, & 0.0F, \\ 0.0F, & 0.0F, & 1.0F \end{pmatrix}} \overset{\text{matrix}}{\begin{pmatrix} 1.0F, & 2.0F, & 3.0F, \\ 4.0F, & 5.0F, & 6.0F, \\ 0.0F, & 0.0F, & 1.0F \end{pmatrix}} = \overset{\text{matrix2.postRotate(90)}}{\begin{pmatrix} -4.0F, & -5.0F, & -6.0F, \\ 1.0F, & 2.0F, & 3.0F, \\ 0.0F, & 0.0F, & 1.0F \end{pmatrix}}$$

看了上面的输出信息。我们得出结论：**Preconcats matrix 相当于右乘矩阵，Postconcats matrix 相当于左乘矩阵。**

上一篇副中，我们说到：

Matrix matrix = new Matrix();
matrix.setRotate(30, 100, 100);
	等价	float a = 100.0F,b = 100.0F;
		matrix = new Matrix();
		matrix.setTranslate(a,b);
		matrix.preRotate(30);
		matrix.preTranslate(-a,-b);

其晕死过程的详细分析就不在这里多说了。

Matrix 学习——错切变换

什么是**图像的错切变换**（Shear transformation）？我们还是直接看图片错切变换后是的效果：


```
matrix = new Matrix();
matrix.setSkew(0.0f,0.5f);
Test.Log(MAXTRIX_TAG,"setValues():%s",matrix.toString());
```

当前矩阵运行后的结果是：

```
Matrix  setValues():Matrix{[1.0, 0.0, 0.0]
                           [0.5, 1.0, 0.0]
                           [0.0, 0.0, 1.0]}
```

对图像的错切变换做个总结：

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & b & 0 \\ d & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x0 \\ y0 \\ 1 \end{pmatrix}$$

x = x0 + b*y0;

y = d*x0 + y0;

当 d=0 时 x=x0+b*y0， y=y0	图形的 Y 坐标不变，X 坐标随初始值[x0，y0]及变化系数 b 而作线性变换；如 b>0，图形沿+X 方向作切换变换；如 b<0，图形沿-X 方向作切换变换。
当 b=0 时 x=x0， y=d*x0+y0	图形的 X 坐标不变，Y 坐标随初始值[x0，y0]及变化系数 d 而作线性变换；如 d>0，图形沿+Y 方向作切换变换；如 d<0，图形沿-Y 方向作切换变换。
当 b≠0，且 d≠0 时 x=x0+b*y0， y=d*x0+y0	图形沿 X，Y 两个方向作切换变换。

这里再次给大家介绍一个需要注意的地方：

matrix = new Matrix();
matrix.setSkew(0.2f,0.2f);
matrix.preTranslate(20,20);
	matrix = new Matrix();
	matrix.setSkew(0.2f,0.2f);
	matrix.setTranslate(20,20);

Test.Log(MAXTRIX_TAG,"setValues():%s",matrix.toString());	
Matrix{[1.0, 0.2, 24.0] [0.2, 1.0, 24.0] [0.0, 0.0, 1.0]}	Matrix{[1.0, 0.0, 20.0] [0.0, 1.0, 20.0] [0.0, 0.0, 1.0]}
	

通过以上，我们发现 Matrix 的 setXXXX()函数，在调用时调用了一次 reset()，这个在复合变换时需要注意。

Matrix 学习——对称变换（反射）

什么是对称变换？具体的理论就不详细说明了，图像的镜像就是对称变换中的一种。

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & 0 \\ d & e & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x0 \\ y0 \\ 1 \end{pmatrix}$$

x = a*x0 + b*y0;

y = d*x0 + e*y0;

当 b=d=0，a=-1，e=1 时 有 x=-x0，y=y0	产生与 Y 轴对称的反射图形（水平镜像）
当 b=d=0，a=1，e=-1 时 有 x=x0，y=-y0	产生与 X 轴对称的反射图形（垂直镜像）
当 b=d=0，a=e=-1 时 有 x=-x0，y=-y0	产生与原点对称的反射图形
当 b=d=1，a=e=0 时 有 x=y0，y=x0	产生与直线 y=x 对称的反射图形
当 b=d=-1，a=e=0 时 有 x=-y0，y=-x0	产生与直线 y=-x 对称的反射图形

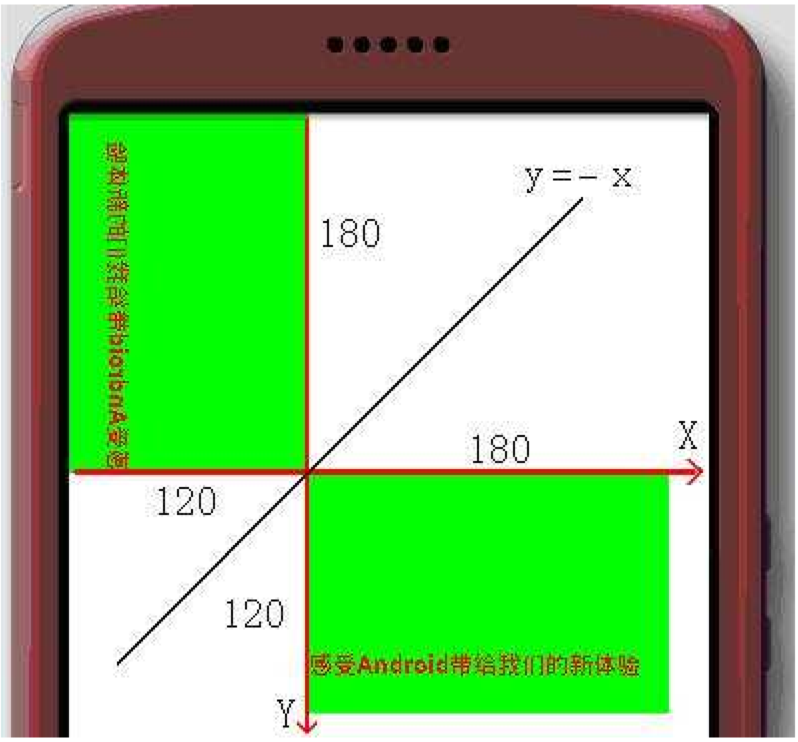
利用上面的总结做个具体的例子，产生与直线 y= － x 对称的反射图形，代码片段如下：

```
final float f4[] = {  0.0F, -1.0F,0.0F,
                     -1.0F,  0.0F,0.0F,
                      0.0F,  0.0F,1.0F
                   };
matrix = new Matrix();
matrix.setValues(f4);
matrix.preTranslate(-180,-120);
Test.Log(MAXTRIX_TAG,"setValues():%s",matrix.toString());
//原图与变换后的图像的对比
canvas.drawBitmap(mbmpTest, 120,180, null);
canvas.drawBitmap(mbmpTest, matrix, null);
```

当前矩阵输出是：

```
Matrix setValues():Matrix{[0.0, -1.0, 120.0]
                         [-1.0, 0.0, 180.0]
                         [0.0, 0.0, 1.0]}
```

图像变换的效果如下：


y = - x
180
180
X
120
120
感受Android带给我们的新体验
Y