**[Урок 144. Рисование. Matrix-преобразования](http://startandroid.ru/ru/uroki/vse-uroki-spiskom/317-urok-144-risovanie-matrix-preobrazovanija.html)**

В этом уроке:

- используем Matrix для геометрических преобразований фигур

Рисовать фигуры мы умеем, но бывает необходимость эти фигуры перемещать по экрану, поворачивать и менять их размеры. Тут нам поможет [Matrix](http://developer.android.com/reference/android/graphics/Matrix.html" \t "_blank). Будем называть этот объект матрицей. Тем более, что это действительно 3х3 матрица из алгебры.

Матрица умеет выполнять четыре основные операции над фигурой:

**translate** – перемещение

**scale** – изменение размера

**rotate** – поворот

**skew** - наклон

Рассмотрим эти возможности на примерах. Начнем с перемещения.

**Перемещение**

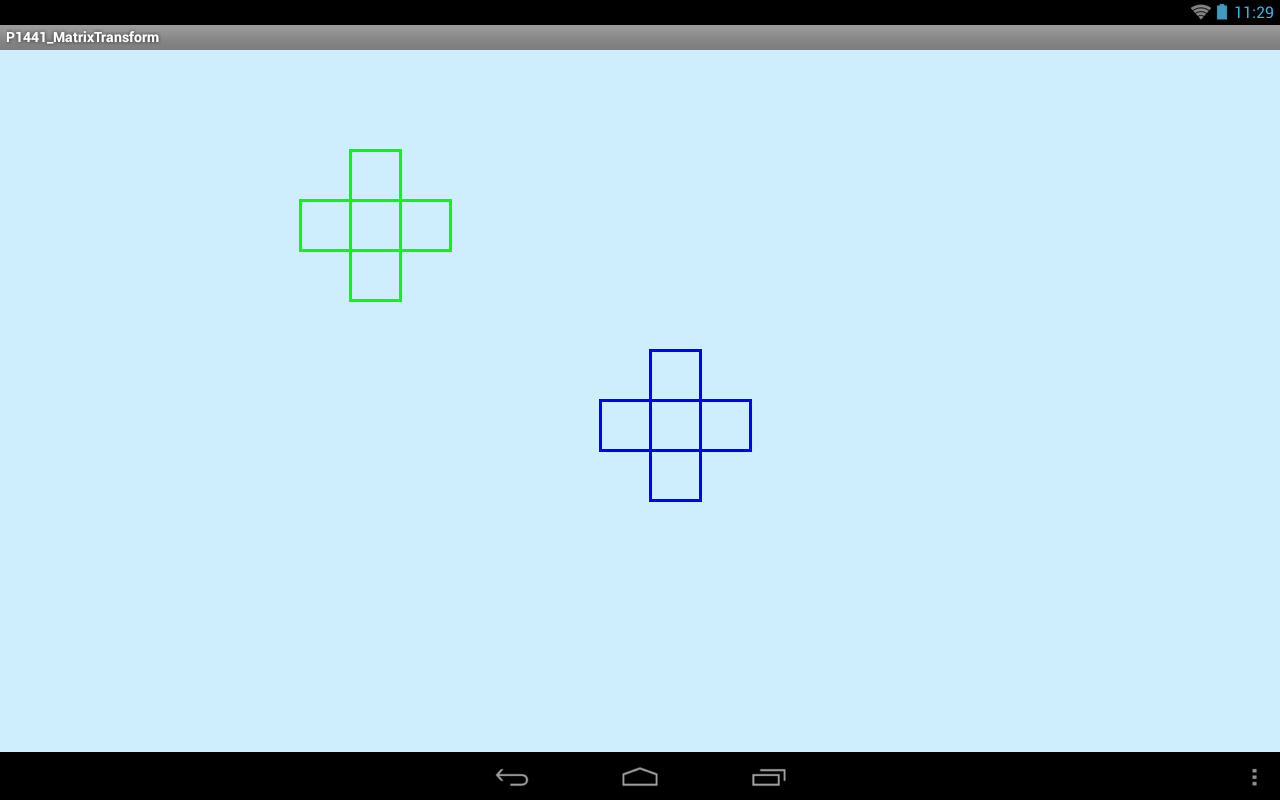
Создадим проект:

**Project name**: P1441\_MatrixTransform  
**Build Target**: Android 2.3.3   
**Application name**: MatrixTransform  
**Package name**: ru.startandroid.develop.p1441matrixtransform  
**Create Activity**: MainActivity

**MainActivity.java**:

**package** ru.startandroid.develop.p1441matrixtransform;  
  
**import** android.app.Activity;  
**import** android.content.Context;  
**import** android.graphics.Canvas;  
**import** android.graphics.Color;  
**import** android.graphics.Matrix;  
**import** android.graphics.Paint;  
**import** android.graphics.Path;  
**import** android.os.Bundle;  
**import** android.view.View;  
  
**public class** MainActivity **extends** Activity {  
  
  @Override  
  **protected void** onCreate(Bundle savedInstanceState) {  
    **super**.onCreate(savedInstanceState);  
    setContentView(**new** DrawView(**this**));  
  }  
  
  **class** DrawView **extends** View {  
  
    Paint p;  
    Path path;  
    Matrix matrix;  
  
    **public** DrawView(Context context) {  
      **super**(context);  
      p = **new** Paint();  
      p.setStrokeWidth(3);  
      p.setStyle(Paint.Style.STROKE);  
  
      path = **new** Path();  
      matrix = **new** Matrix();  
    }  
  
    @Override  
    **protected void** onDraw(Canvas canvas) {  
      canvas.drawARGB(80, 102, 204, 255);  
  
      // создаем крест в path  
      path.reset();  
      path.addRect(300, 150, 450, 200, Path.Direction.CW);  
      path.addRect(350, 100, 400, 250, Path.Direction.CW);  
  
      // рисуем path зеленым  
      p.setColor(Color.GREEN);  
      canvas.drawPath(path, p);  
  
      // настраиваем матрицу на перемещение на 300 вправо и 200 вниз  
      matrix.reset();  
      matrix.setTranslate(300, 200);  
        
      // применяем матрицу к path  
      path.transform(matrix);  
  
      // рисуем path синим  
      p.setColor(Color.BLUE);  
      canvas.drawPath(path, p);  
  
    }  
  
  }  
  
}

Результат:



В конструкторе DrawView создаем объекты matrix и path.

В onDraw очищаем path, добавляем к нему пару прямоугольников, чтобы получился крест, и выводим на экран зеленым цветом. Далее очищаем матрицу методом[reset](http://developer.android.com/reference/android/graphics/Matrix.html#reset()) и настраиваем смещение на 300 вправо и 200 вниз методом [setTranslate](http://developer.android.com/reference/android/graphics/Matrix.html" \l "setTranslate(float, float)" \t "_blank). Матрица настроена и готова к выполнению преобразований.

Методом [transform](http://developer.android.com/reference/android/graphics/Path.html" \l "transform(android.graphics.Matrix)" \t "_blank) объекта path мы выполняем преобразование этого объекта согласно переданной в этот метод матрице. Т.е. фигура path должна сместиться вправо на 300 и вниз на 200.

Включаем синий цвет и выводим измененную фигуру path на экран. На скриншоте видно, что синяя фигура смещена так, как мы и планировали.

Вот и весь алгоритм использования матрицы. Т.е. сначала мы матрицу настраиваем, а потом применяем к какой-либо фигуре и фигура трансформируется согласно настройкам этой матрицы.

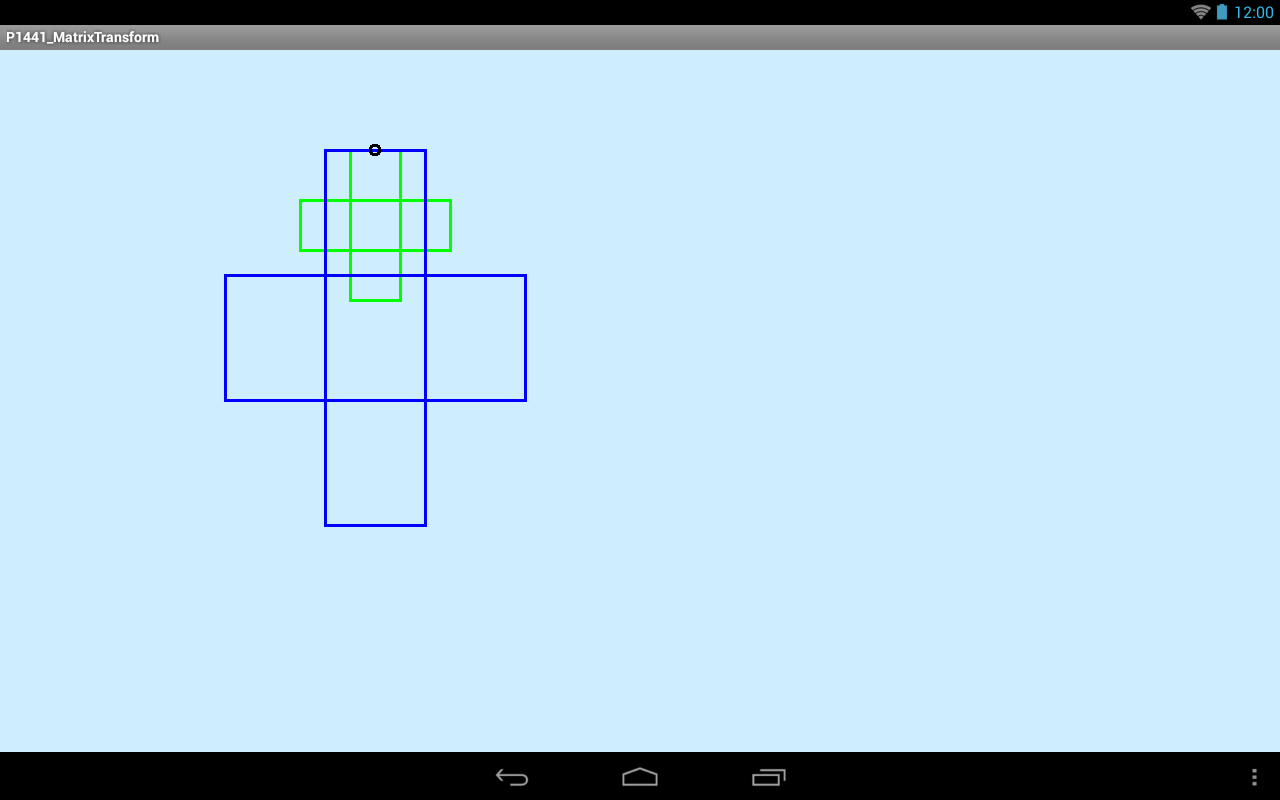
**Изменение размера**

Перепишем класс **DrawView**:

**class** DrawView **extends** View {  
      
    Paint p;  
    Path path;  
    Matrix matrix;  
  
    **public** DrawView(Context context) {  
      **super**(context);  
      p = **new** Paint();  
      p.setStrokeWidth(3);  
      p.setStyle(Paint.Style.STROKE);  
  
      path = **new** Path();  
      matrix = **new** Matrix();  
    }  
      
    @Override  
    **protected void** onDraw(Canvas canvas) {  
      canvas.drawARGB(80, 102, 204, 255);  
  
      // создаем крест в path  
      path.reset();  
      path.addRect(300,150,450,200, Path.Direction.CW);  
      path.addRect(350,100,400,250, Path.Direction.CW);  
        
      // рисуем path зеленым  
      p.setColor(Color.GREEN);  
      canvas.drawPath(path, p);  
        
      // настраиваем матрицу на изменение размера:  
      // в 2 раза по горизонтали  
      // в 2,5 по вертикали  
      // относительно точки (375, 100)  
      matrix.reset();        
      matrix.setScale(2f, 2.5f, 375, 100);  
        
      // применяем матрицу к path  
      path.transform(matrix);  
        
      // рисуем path синим  
      p.setColor(Color.BLUE);  
      canvas.drawPath(path, p);  
  
      // рисуем точку относительно которой было выполнено преобразование  
      p.setColor(Color.BLACK);  
      canvas.drawCircle(375, 100, 5, p);  
        
    }  
      
  }

Рассмотрим тот же path, что и в первом примере. Но теперь для настройки матрицы используем метод [setScale](http://developer.android.com/reference/android/graphics/Matrix.html" \l "setScale(float, float, float, float)" \t "_blank). На вход ему передаем коэффициенты изменения размера в ширину (2), в высоту (2.5) и координаты точки (375, 100), относительно которой будет выполнено изменение размера.

Результат:



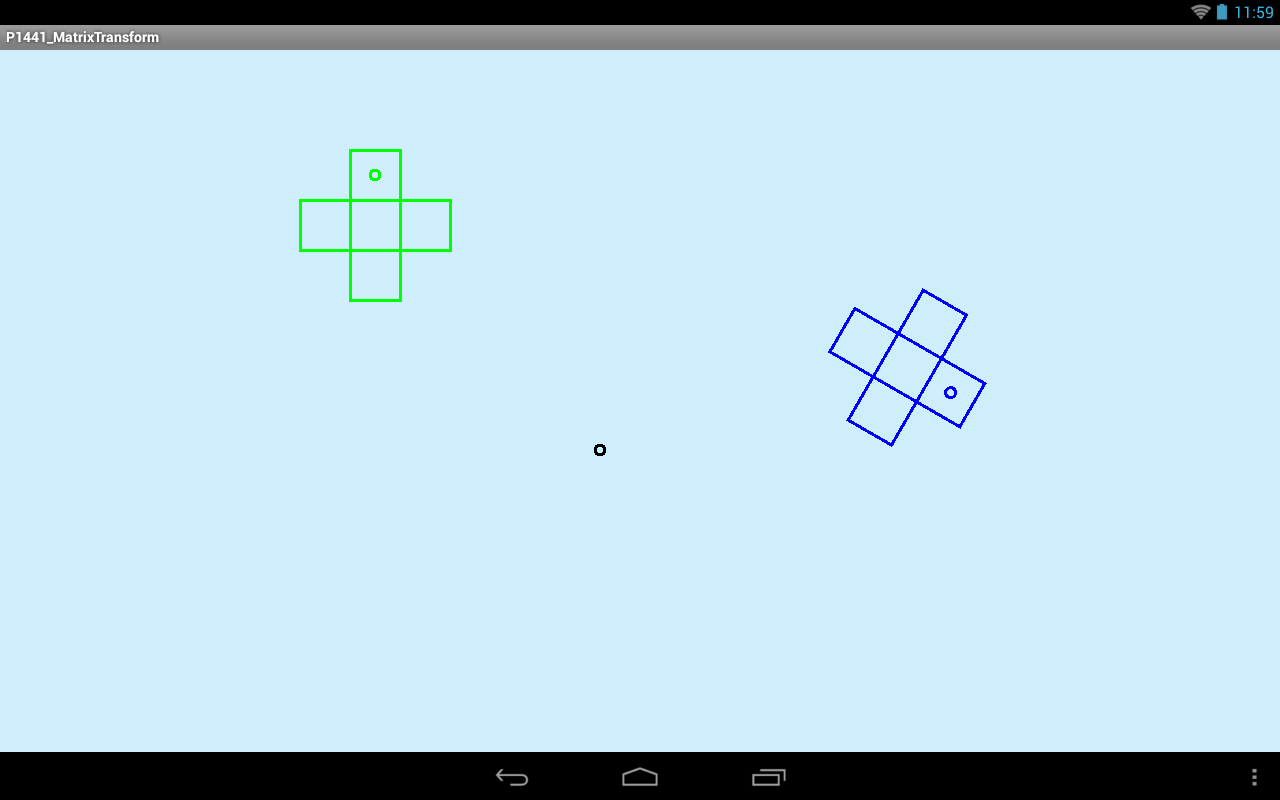
Видим, что крест стал в два раза шире и в два с половиной раза выше. Точку, относительно которой было выполнено изменение размера, мы отметили черным кругом.

Существует также вариант метода [setScale](http://developer.android.com/reference/android/graphics/Matrix.html" \l "setScale(float, float)" \t "_blank), который позволяет не указывать точку. В этом случае используется точка (0,0).

**Поворот**

Перепишем класс **DrawView**:

**class** DrawView **extends** View {  
      
    Paint p;  
    Path path;  
    Matrix matrix;  
  
    **public** DrawView(Context context) {  
      **super**(context);  
      p = **new** Paint();  
      p.setStrokeWidth(3);  
      p.setStyle(Paint.Style.STROKE);  
  
      path = **new** Path();  
      matrix = **new** Matrix();  
    }  
      
    @Override  
    **protected void** onDraw(Canvas canvas) {  
      canvas.drawARGB(80, 102, 204, 255);  
  
      // создаем крест в path  
      path.reset();  
      path.addRect(300,150,450,200, Path.Direction.CW);  
      path.addRect(350,100,400,250, Path.Direction.CW);  
      path.addCircle(375, 125, 5, Path.Direction.CW);  
  
      // рисуем path зеленым  
      p.setColor(Color.GREEN);  
      canvas.drawPath(path, p);  
  
      // настраиваем матрицу на поворот на 120 градусов  
      // относительно точки (600,400)  
      matrix.reset();  
      matrix.setRotate(120, 600, 400);  
        
      // применяем матрицу к path  
      path.transform(matrix);  
        
      // рисуем path синим  
      p.setColor(Color.BLUE);  
      canvas.drawPath(path, p);  
        
      // рисуем точку, относительно которой был выполнен поворот  
      p.setColor(Color.BLACK);  
      canvas.drawCircle(600, 400, 5, p);  
    }  
      
  }



Здесь мы в path добавляем два прямоугольника, чтобы получился крест, и рисуем мелкий круг в верхней части креста. Выводим path на экран зеленым цветом.

Далее настраиваем матрицу методом [setRotate](http://developer.android.com/reference/android/graphics/Matrix.html" \l "setRotate(float, float, float)" \t "_blank). На вход метод принимает угол (120) поворота и координаты точки (600,400), относительно которой будет выполнен поворот. Применяем матрицу к фигуре path и рисуем ее синим цветом.

Черным цветом рисуем мелкий круг в точке (600,400), относительно которой был выполнен поворот. Делаем это для наглядности результата. На скриншоте видно, что фигура повернута относительно черного кружка по часовой на 120 градусов.

Существует также вариант метода [setRotate](http://developer.android.com/reference/android/graphics/Matrix.html" \l "setRotate(float)" \t "_blank), который требует только угол, без координат. В этом случае, в качестве точки поворота будет использована точка (0,0).

**Порядок операций**

Методы **set**\* настраивают матрицу на выполнение одной определенной трансформации. Но матрица может содержать в себе несколько трансформаций. Если возникает такая необходимость, то необходимо использовать методы **pre**\* и **post**\*.

Методы pre\* добавляют трансформацию в самое начало списка трансформаций матрицы. Методы post\* - в самый конец.

Причем это может сыграть очень существенную роль в результате выполнения преобразований. Тут та самая ситуация, когда от перемены мест слагаемых сумма очень даже может поменяться.

Рассмотрим пример. Перепишем класс **DrawView**:

**class** DrawView **extends** View {  
      
    Paint p;  
    Path path;  
    Path pathDst;  
    Matrix matrix;  
  
    **public** DrawView(Context context) {  
      **super**(context);  
      p = **new** Paint();  
      p.setStrokeWidth(3);  
      p.setStyle(Paint.Style.STROKE);  
  
      path = **new** Path();  
      pathDst = **new** Path();  
      matrix = **new** Matrix();  
    }  
      
    @Override  
    **protected void** onDraw(Canvas canvas) {  
      canvas.drawARGB(80, 102, 204, 255);  
        
      p.setColor(Color.BLACK);  
      canvas.drawCircle(400, 200, 10, p);  
  
      // прямоугольник  
      path.reset();  
      path.addRect(300,100,500,300, Path.Direction.CW);  
      canvas.drawPath(path, p);  
        
      // перемещение после поворота  
      matrix.reset();  
      matrix.setRotate(45, 400, 200);  
      matrix.postTranslate(500, 0);  
      path.transform(matrix, pathDst);  
      p.setColor(Color.GREEN);  
      canvas.drawPath(pathDst, p);  
        
      // перемещение до поворота  
      matrix.reset();  
      matrix.setRotate(45, 400, 200);  
      matrix.preTranslate(500, 0);  
      path.transform(matrix, pathDst);  
      p.setColor(Color.RED);  
      canvas.drawPath(pathDst, p);        
    }  
  }

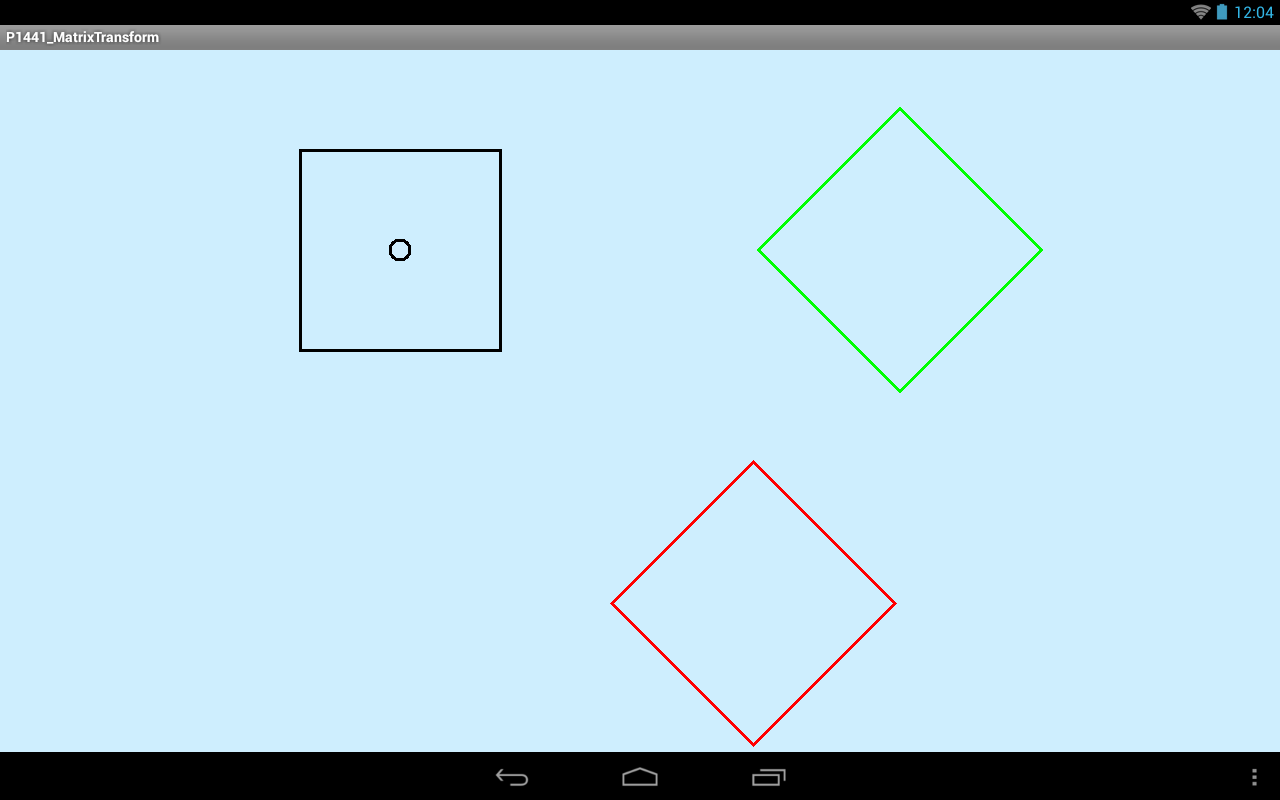
Создаем path из одного прямоугольника и рисуем его черным цветом.

Настраиваем матрицу на поворот на 45 градусов относительно точки (400,200), а затем методом [postTranslate](http://developer.android.com/reference/android/graphics/Matrix.html" \l "postTranslate(float, float)" \t "_blank) добавляем к матрице трансформу перемещения на 500 направо. Применяем результат к path методом [transform](http://developer.android.com/reference/android/graphics/Path.html" \l "transform(android.graphics.Matrix, android.graphics.Path)" \t "_blank), который полученный результат поместит в объект pathDst, а path оставит прежним. Это необходимо, т.к. нам дальше еще понадобится оригинальный path.  Выводим pathDst зеленым цветом.

Далее делаем все аналогично, только используем метод [preTranslate](http://developer.android.com/reference/android/graphics/Matrix.html" \l "preTranslate(float, float)" \t "_blank), который добавит трансформу перемещения в начало матрицы. И результат выводим красным цветом.

Для наглядности выведем мелкий круг в точке (400,200), которая у нас являлась точкой поворота в методах setRotate. Эта точка, кстати, является серединой первоначального квадрата.

Смотрим результат:



Как был получен зеленый квадрат? Матрица взяла черный квадрат, повернула (setRotate) его вокруг его же центра на 45 градусов и сместила (postTranslate) вправо на 500. Все примерно так и выглядит на скриншоте.

А вот красный квадрат уехал. Разбираемся. Мы также, как и в случае с зеленым квадратом, сначала попросили матрицу повернуть (setRotate) черный квадрат на 45 относительно своего центра. А вот трансформацию смещения мы добавили методом preTranslate. Т.е. она пошла в начало матрицы и выполнилась до поворота. Т.е. красный квадрат сначала был смещен на 500 вправо, а потом повернут относительно точки (400,200), которая уже вовсе не являлась его центром. И поэтому квадрат уехал вниз по часовой стрелке на 45 градусов относительно точки (400,200).

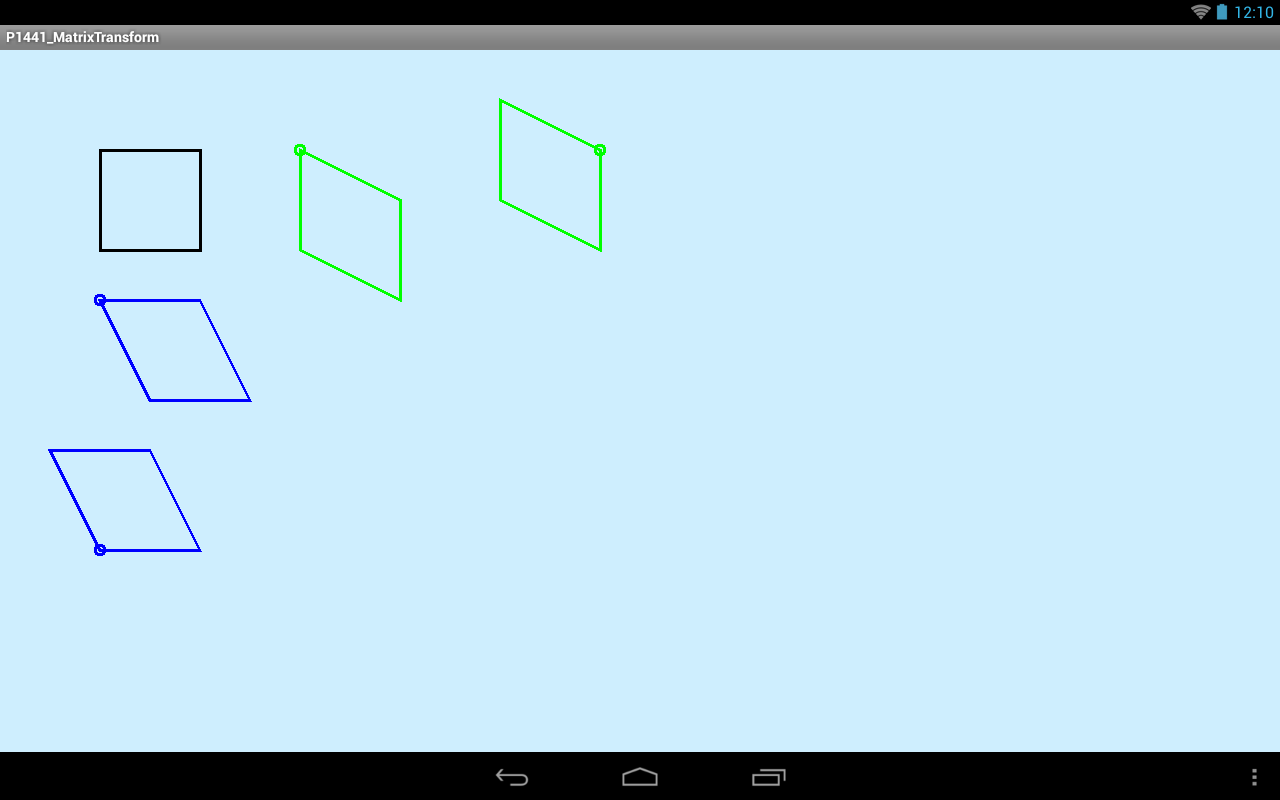
Этот простой пример позволяет увидеть насколько важно соблюдать порядок трансформаций в матрице.  Операции были одни и те же, а разный порядок дал разный результат.

**Наклон**

Перепишем класс **DrawView**:

**class** DrawView **extends** View {  
  
    Paint p;  
    Path path;  
    Path pathDst;  
    Matrix matrix;  
  
    **public** DrawView(Context context) {  
      **super**(context);  
      p = **new** Paint();  
      p.setStrokeWidth(3);  
      p.setStyle(Paint.Style.STROKE);  
  
      path = **new** Path();  
      pathDst = **new** Path();  
      path.addRect(100, 100, 200, 200, Path.Direction.CW);  
  
      matrix = **new** Matrix();  
    }  
  
    @Override  
    **protected void** onDraw(Canvas canvas) {  
      canvas.drawARGB(80, 102, 204, 255);  
  
      p.setColor(Color.BLACK);  
      canvas.drawPath(path, p);  
  
      p.setColor(Color.GREEN);  
  
      // перемещение на 200 вправо  
      // и наклон по вертикали на 0.5  
      // точка наклона - слева  
      matrix.reset();  
      matrix.setTranslate(200, 0);  
      matrix.postSkew(0.0f, 0.5f, 300, 100);  
      path.transform(matrix, pathDst);  
      canvas.drawPath(pathDst, p);  
      canvas.drawCircle(300, 100, 5, p);  
  
      // перемещение на 400 вправо  
      // и наклон по вертикали на 0.5  
      // точка наклона - справа  
      matrix.reset();  
      matrix.setTranslate(400, 0);  
      matrix.postSkew(0.0f, 0.5f, 600, 100);  
      path.transform(matrix, pathDst);  
      canvas.drawPath(pathDst, p);  
      canvas.drawCircle(600, 100, 5, p);  
  
      p.setColor(Color.BLUE);  
  
      // перемещение на 150 вниз  
      // и наклон по горизонтали на 0.5  
      // точка наклона - сверху  
      matrix.reset();  
      matrix.setTranslate(0, 150);  
      matrix.postSkew(0.5f, 0.0f, 100, 250);  
      path.transform(matrix, pathDst);  
      canvas.drawPath(pathDst, p);  
      canvas.drawCircle(100, 250, 5, p);  
  
      // перемещение на 300 вниз  
      // и наклон по горизонтали на 0.5  
      // точка наклона - снизу  
      matrix.reset();  
      matrix.setTranslate(0, 300);  
      matrix.postSkew(0.5f, 0.0f, 100, 500);  
      path.transform(matrix, pathDst);  
      canvas.drawPath(pathDst, p);  
      canvas.drawCircle(100, 500, 5, p);  
    }  
  }

Результат:



path содержит один прямоугольник. Выводим его на экран черным цветом.

Далее настраиваем матрицу на смещение на 200 вправо. И методом [postSkew](http://developer.android.com/reference/android/graphics/Matrix.html" \l "postSkew(float, float, float, float)" \t "_blank) настраиваем наклон. На вход ему передаем значения наклона по оси X (0), наклона по оси Y (0.5) и координаты точки, относительно которой будет выполнен наклон (300,100). Применяем матрицу к path и рисуем результат. Также для наглядности рисуем круг в точке, относительно которой был наклон.

На скриншоте это левый зеленый наклоненный квадрат. Он наклонен по оси Y с коэффициентом 0.5. Видим, что его правая (относительно точки наклона) часть ушла вниз.

Аналогично рисуем второй зеленый квадрат, но точку наклона используем другую. Тут уже видно, что при том же наклоне (0.5 по оси Y) левая часть уходит вверх.

Далее аналогично выводим два синих квадрата, но у них уже наклон идет по оси X. Видно, что нижняя часть при этом наклоняется вправо, а верхняя влево.

Несколько дополнений к наклону.

Если задавать отрицательные значения наклона по осям, то направления наклона поменяются.

Разумеется, можно сразу задавать наклон по обоим осям. Просто в этом примере для упрощения мы наклоняли каждый квадрат только по какой-то одной оси.

Точка наклона может быть любой, она не должна принадлежать фигуре. Есть аналог метода setSkew, который принимает только коэффициенты наклона, а в качестве точки использует (0,0).

Советую вам поэкспериментировать и позадавать различные значения наклонов по осям, если еще не до конца понятен механизм наклона.

**Map методы**

[mapRadius](http://developer.android.com/reference/android/graphics/Matrix.html#mapRadius(float)) – даете на вход радиус (если собираетесь трансформировать круг), а метод вам вернет значение радиуса после трансформаци

[mapPoints](http://developer.android.com/reference/android/graphics/Matrix.html#mapPoints(float[], float[])) – даете массив точек, матрица выполняет над ними свои операции преобразования и возвращает вам в результате новый массив точек

[mapVectors](http://developer.android.com/reference/android/graphics/Matrix.html#mapVectors(float[], float[])) – то же, что и mapPoints, но преобразования перемещения выполнены не будут

[mapRect](http://developer.android.com/reference/android/graphics/Matrix.html#mapRect(android.graphics.RectF, android.graphics.RectF)) – возьмет на вход прямоугольник, выполнит для него преобразование и вернет прямоугольник, составляющий границы получившейся фигуры. Рассмотрим этот метод на примере.

Перепишем класс **DrawView**:

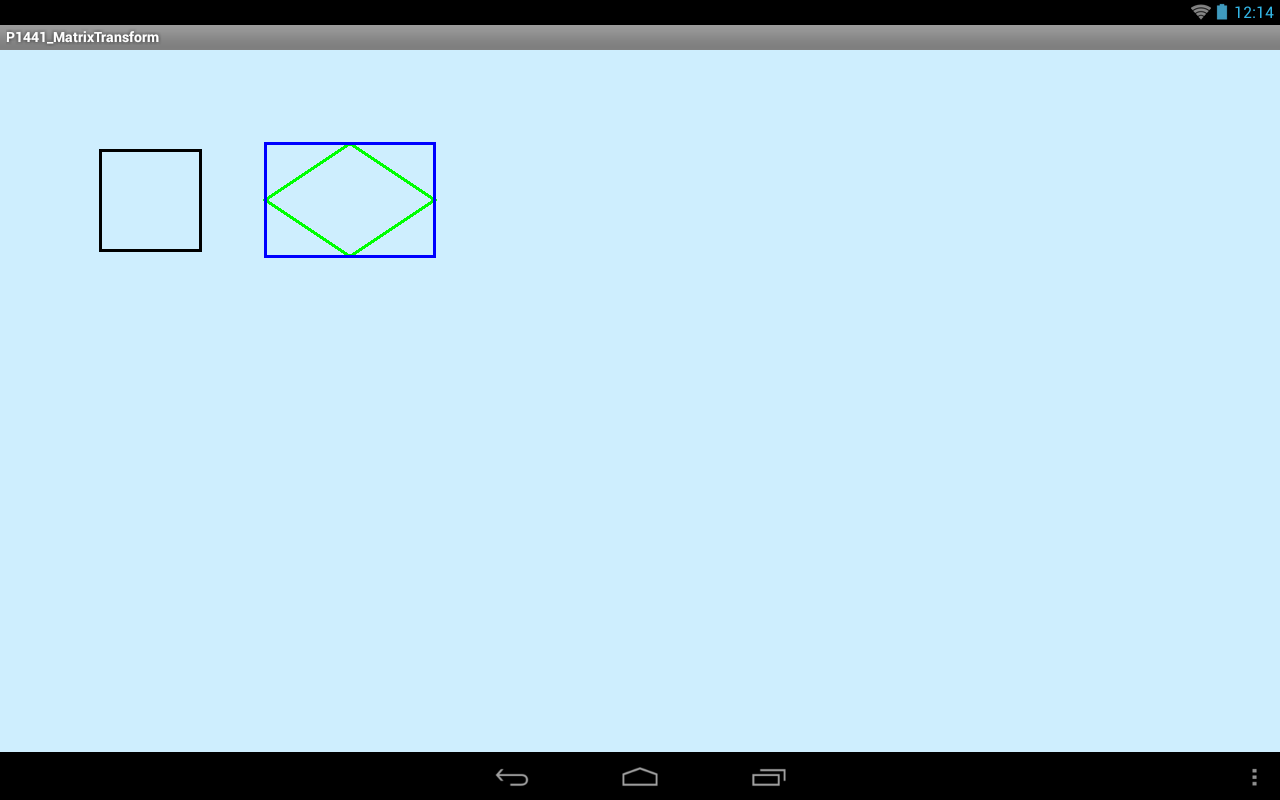
**class** DrawView **extends** View {  
  
    Paint p;  
    Matrix matrix;  
    RectF rectf;  
    RectF rectfDst;  
    Path path;  
  
    **public** DrawView(Context context) {  
      **super**(context);  
      p = **new** Paint();  
      p.setStrokeWidth(3);  
      p.setStyle(Paint.Style.STROKE);  
      rectf = **new** RectF(100, 100, 200, 200);  
      rectfDst = **new** RectF();  
      matrix = **new** Matrix();  
      path = **new** Path();  
    }  
  
    @Override  
    **protected void** onDraw(Canvas canvas) {  
      canvas.drawARGB(80, 102, 204, 255);  
  
      // прямоугольник  
      path.reset();  
      path.addRect(rectf, Path.Direction.CW);  
      p.setColor(Color.BLACK);  
      canvas.drawPath(path, p);  
  
      // поворот, размер, пермещение  
      matrix.setRotate(45, 150, 150);  
      matrix.postScale(1.2f, 0.8f, 150, 150);  
      matrix.postTranslate(200, 0);  
      path.transform(matrix);  
        
      // итоговая фигура зеленым цветом  
      p.setColor(Color.GREEN);  
      canvas.drawPath(path, p);  
  
      // границы, полученные от изначального прямоугольника   
      matrix.mapRect(rectfDst, rectf);  
      p.setColor(Color.BLUE);  
      canvas.drawRect(rectfDst, p);  
  
    }  
  }

К path добавляем квадрат и выводим черным цветом.

Настраиваем преобразования матрицы. Это поворот на 45 градусов относительно центра квадрата (150,150). Затем растянем его по горизонтали на 1.2 и сожмем по вертикали на 0.8. И переместим на 200 вправо.

Применяем матрицу к path и выводим результат зеленым цветом.

Далее выполним метод mapRect, который возьмет прямоугольник rectf, выполнит над ним все преобразования матрицы, и в rectfDst запишет границы получившейся фигуры. Выведем эти границы синим цветом.



Видим, что зеленым нарисована фигура, которая получилась из прямоугольника после преобразований, а синим – границы фигуры, которые были получены из прямоугольника методом mapRect. Т.е. mapRect сообщает нам, какие границы будут у прямоугольника после преобразований.

**Операции над матрицами**

[set](http://developer.android.com/reference/android/graphics/Matrix.html#set(android.graphics.Matrix)) – считает настройки с переданной матрицы в нашу матрицу

[setConcat](http://developer.android.com/reference/android/graphics/Matrix.html#setConcat(android.graphics.Matrix, android.graphics.Matrix)) – соберет преобразования из двух матриц в вашу матрицу

[preConcat](http://developer.android.com/reference/android/graphics/Matrix.html#preConcat(android.graphics.Matrix)) – добавит преобразования из переданной матрицы в начало вашей матрицы

[postConcat](http://developer.android.com/reference/android/graphics/Matrix.html#postConcat(android.graphics.Matrix)) – добавит преобразования из переданной матрицы в конец вашей матрицы

[invert](http://developer.android.com/reference/android/graphics/Matrix.html#invert(android.graphics.Matrix)) – настроит вашу матрицу на преобразования, обратные тем, на которые настроена переданная матрица

Мы рассмотрели почти все методы матрицы. Есть еще два интересных и полезных метода: setRectToRect и setPolyToPoly. Но по ним получилась целая простыня объяснений и я вынес их в следующий урок.