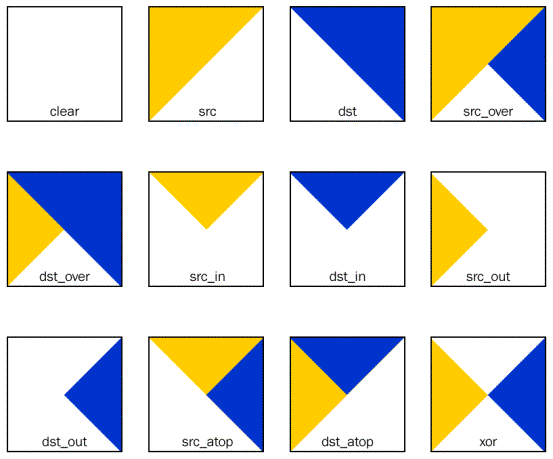
**У**[**рок 154. Рисование. PorterDuff.Mode, PorterDuffXfermode**](http://startandroid.ru/ru/uroki/vse-uroki-spiskom/356-urok-154-porterduffmode-porterduffxfermode.html)

В этом уроке:

- разбираем PorterDuff режимы используя PorterDuffXfermode

Этот урок снова будет про цвета в графике. [PorterDuff](http://developer.android.com/reference/android/graphics/PorterDuff.Mode.html" \t "_blank)-режимы позволяют нам получать различные результаты при наложении одного изображения на другое. Т.е. берутся значения цвета и прозрачности обоих изображений и по определенному алгоритму рассчитываются итоговые значения.

Для примера я взял картинку [отсюда](http://www.w3.org/TR/2002/WD-SVG11-20020215/masking.html).



Тут перечислены основные PorterDuff-режимы. Создадим пример, который будет выводить нам аналогичные результаты.

Создадим проект:

**Project name**: P1541\_PorterDuff  
**Build Target**: Android 4.4   
**Application name**: PorterDuff  
**Package name**: ru.startandroid.develop.p1541porterduff  
**Create Activity**: MainActivity

**MainActivity.java**:

**package** ru.startandroid.develop.p1541porterduff;  
  
**import** android.app.Activity;  
**import** android.content.Context;  
**import** android.graphics.Bitmap;  
**import** android.graphics.Canvas;  
**import** android.graphics.Color;  
**import** android.graphics.Paint;  
**import** android.graphics.Path;  
**import** android.graphics.PorterDuff;  
**import** android.graphics.PorterDuffXfermode;  
**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 paintSrc;  
    Paint paintDst;  
    Paint paintBorder;  
  
    Path pathSrc;  
    Path pathDst;  
  
    Bitmap bitmapSrc;  
    Bitmap bitmapDst;  
  
    // PorterDuff режим  
    PorterDuff.Mode mode = PorterDuff.Mode.SRC;  
      
    **int** colorDst = Color.BLUE;  
    **int** colorSrc = Color.YELLOW;  
  
    **public** DrawView(Context context) {  
      **super**(context);  
  
      // необходимо для корректной работы  
      **if** (android.os.Build.VERSION.SDK\_INT >= 11) {  
        setLayerType(View.LAYER\_TYPE\_SOFTWARE, **null**);  
      }  
  
      // DST фигура   
      pathDst = **new** Path();  
      pathDst.moveTo(0, 0);  
      pathDst.lineTo(500, 0);  
      pathDst.lineTo(500, 500);  
      pathDst.close();  
  
      // создание DST bitmap  
      bitmapDst = createBitmap(pathDst, colorDst);  
        
      // кисть для вывода DST bitmap  
      paintDst = **new** Paint();  
  
      // SRC фигура   
      pathSrc = **new** Path();  
      pathSrc.moveTo(0, 0);  
      pathSrc.lineTo(500, 0);  
      pathSrc.lineTo(0, 500);  
      pathSrc.close();  
  
      // создание SRC bitmap        
      bitmapSrc = createBitmap(pathSrc, colorSrc);  
  
      // кисть для вывода SRC bitmap        
      paintSrc = **new** Paint();  
      paintSrc.setXfermode(**new** PorterDuffXfermode(mode));  
  
      // кисть для рамки  
      paintBorder = **new** Paint();  
      paintBorder.setStyle(Paint.Style.STROKE);  
      paintBorder.setStrokeWidth(3);  
      paintBorder.setColor(Color.BLACK);  
    }  
  
    **private** Bitmap createBitmap(Path path, **int** color) {  
      // создание bitmap и канвы для него  
      Bitmap bitmap = Bitmap.createBitmap(500, 500,  
          Bitmap.Config.ARGB\_8888);  
      Canvas bitmapCanvas = **new** Canvas(bitmap);  
  
      // создание кисти нужного цвета  
      Paint paint = **new** Paint(Paint.ANTI\_ALIAS\_FLAG);  
      paint.setStyle(Paint.Style.FILL\_AND\_STROKE);  
      paint.setColor(color);  
  
      // рисование фигуры на канве bitmap  
      bitmapCanvas.drawPath(path, paint);  
        
      **return** bitmap;  
    }  
  
    @Override  
    **protected void** onDraw(Canvas canvas) {  
      canvas.translate(390, 80);  
        
      // DST bitmap  
      canvas.drawBitmap(bitmapDst, 0, 0, paintDst);  
  
      // SRC bitmap  
      canvas.drawBitmap(bitmapSrc, 0, 0, paintSrc);  
        
      // рамка  
      canvas.drawRect(0, 0, 500, 500, paintBorder);  
  
    }  
  }  
}

У нас будут две картинки. Одна с синим треугольником (DST), другая с желтым (SRC). В конструкторе **DrawView**создаем треугольник pathDst, и используя его и синий цвет создаем bitmapDst. Аналогично создаем треугольник pathSrc, и с ним и желтым цветом создаем bitmapSrc.

Кисть paintDst будем использовать для рисования картинки bitmapDst, а paintSrc для bitmapSrc. И для кисти paintSrc вызываем метод [setXfermode](http://developer.android.com/reference/android/graphics/Paint.html" \l "setXfermode(android.graphics.Xfermode)" \t "_blank), в который передаем[PorterDuffXfermode](http://developer.android.com/reference/android/graphics/PorterDuffXfermode.html) объект с режимом mode = PorterDuff.Mode.SRC. Далее будем менять значение переменной mode и смотреть на результат.

Кистью paintBorder будем просто рисовать черную рамку.

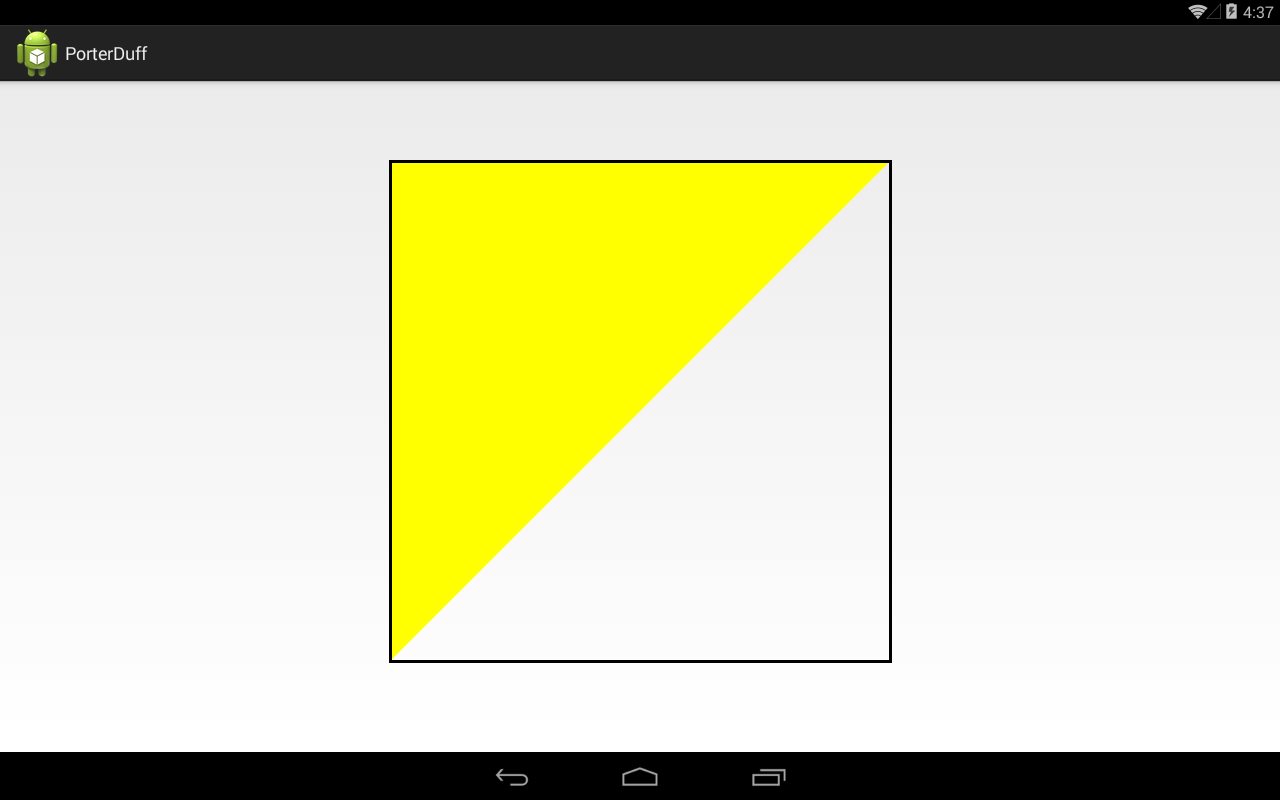
В методе **createBitmap**создаем Bitmap размерами 500х500, создаем для него персональную канву и на ней рисуем полученный path указанным цветом.

В методе **onDraw**рисуем bitmapDst, затем на него bitmapSrc. И для наглядности рисуем черную рамку.

Т.е. мы рисуем картинку bitmapDst (DST), и поверх нее bitmapSrc (SRC). Для рисования bitmapSrc мы используем кисть paintSrc с PorterDuff-режимом, а значит будет не просто наложение одной картинки на другую, а будут использоваться определенные алгоритмы для получения результата.

Запускаем приложение

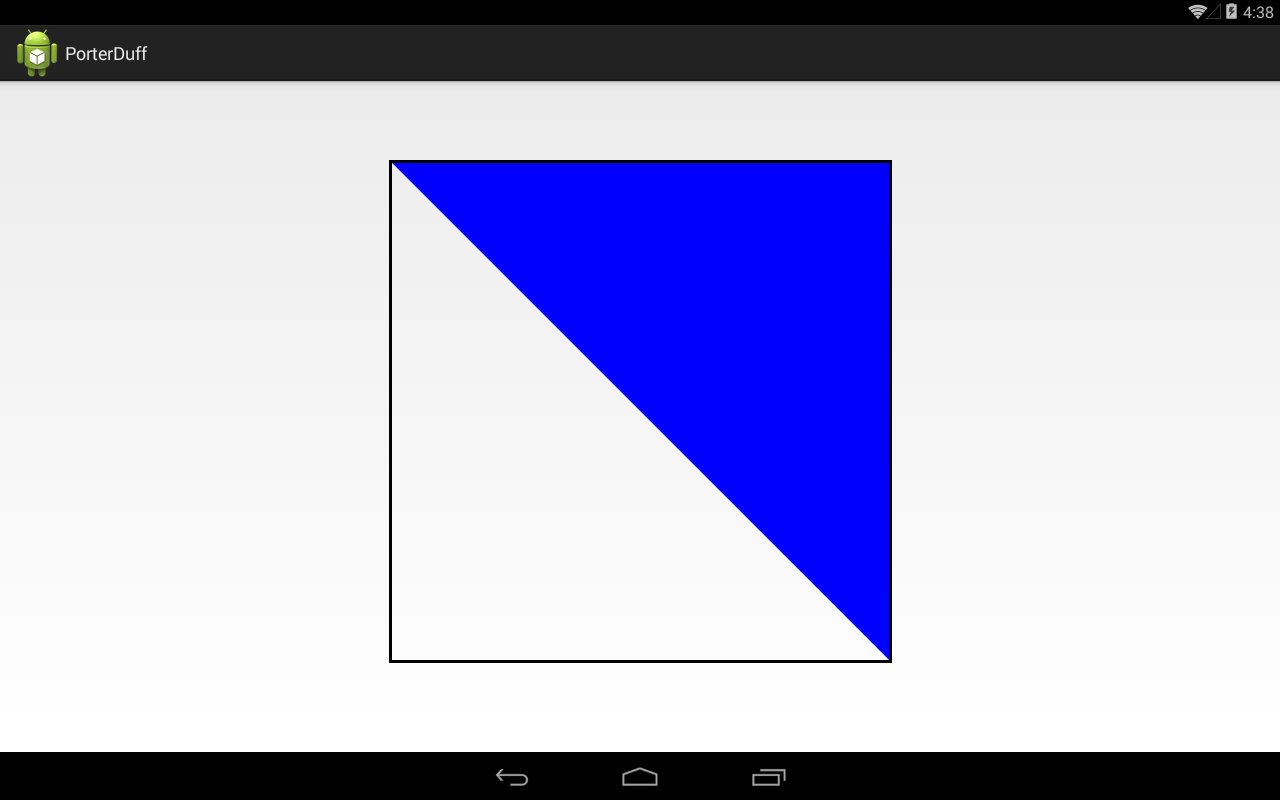
При режиме SRC результат будет такой:



отображается только SRC картинка, т.е. bitmapSrc

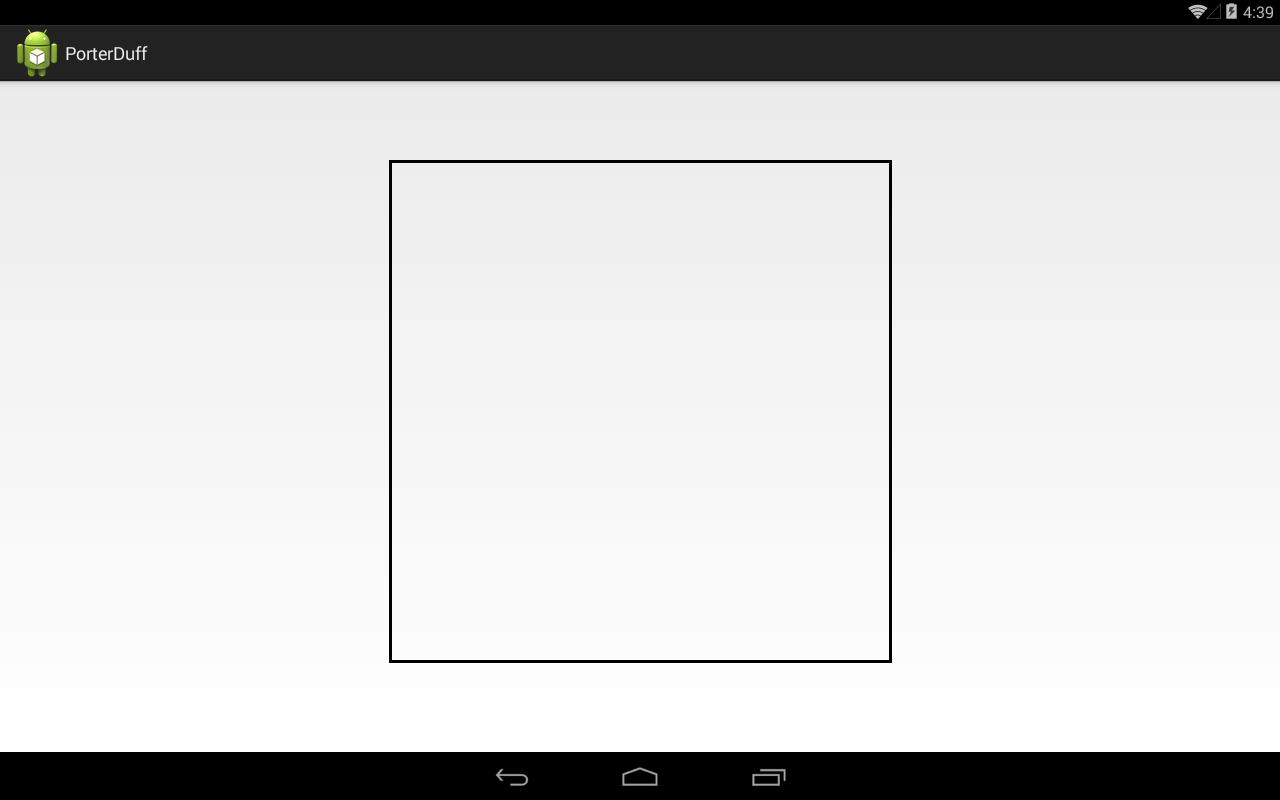
Теперь будем менять в коде значение переменной mode и получаем следующие результаты:

DST



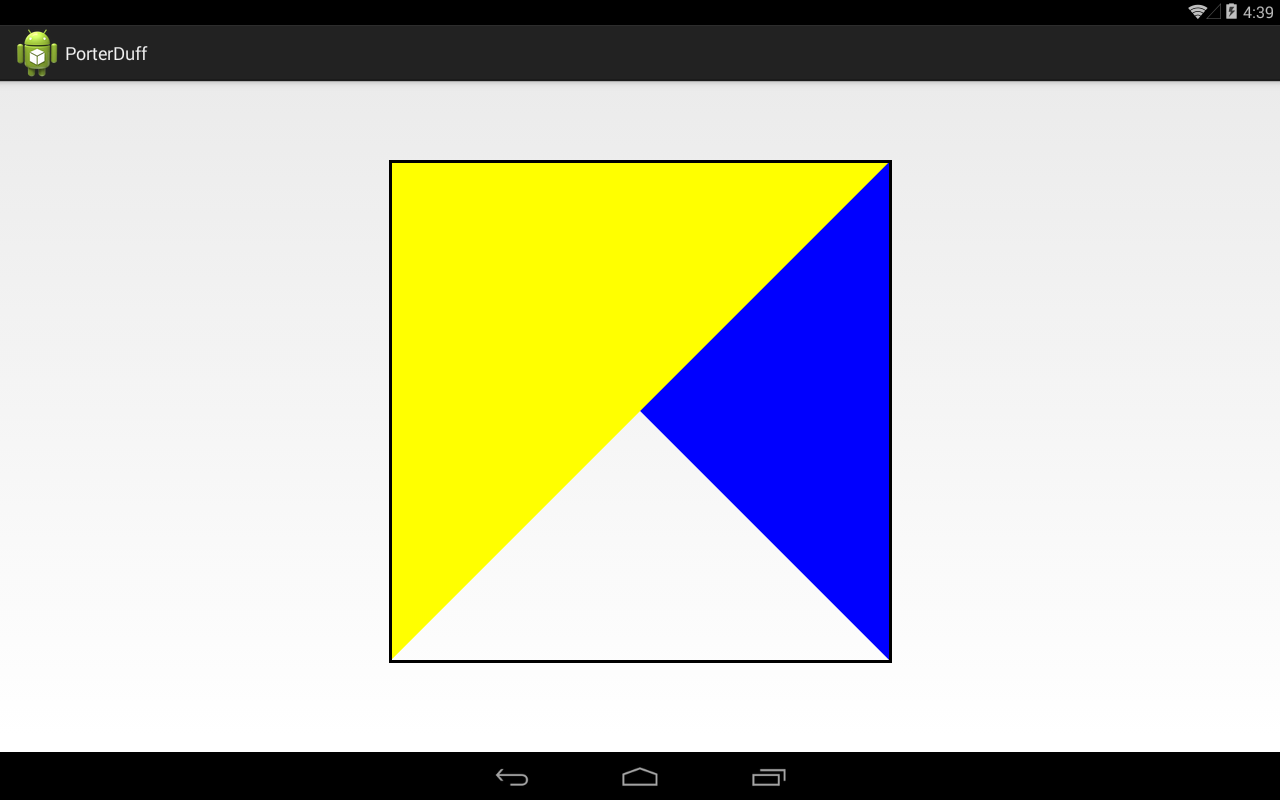
отображается только DST картинка, т.е. bitmapDst

CLEAR



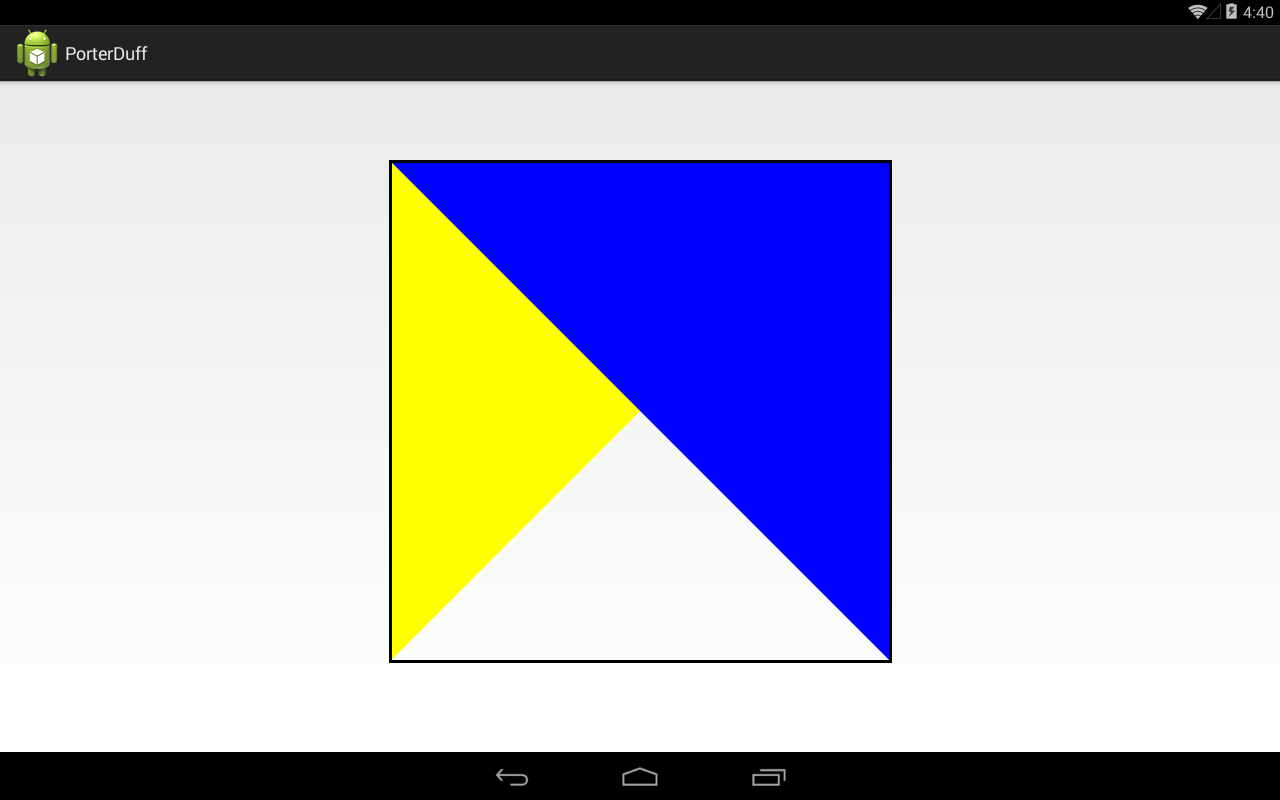
Ничего не отображается

SRC\_OVER



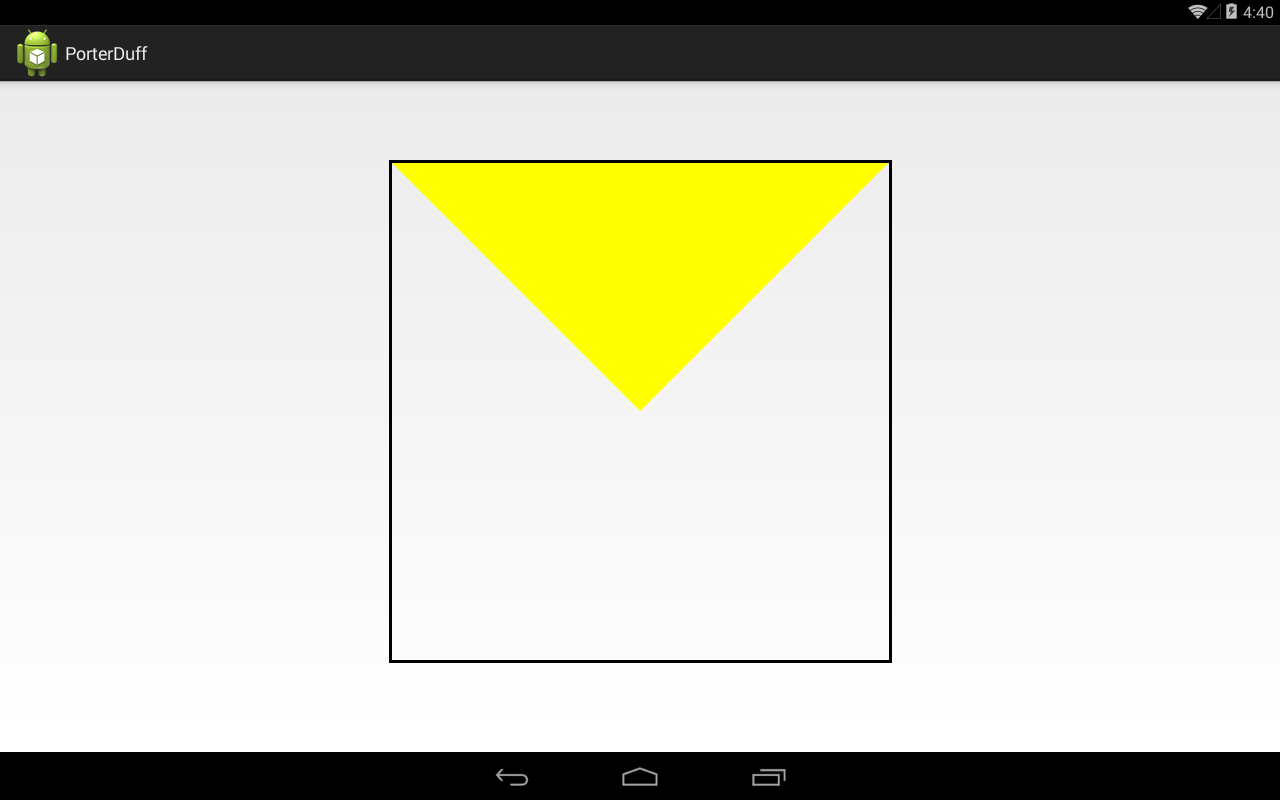
SRC отображается над DST

DST\_OVER



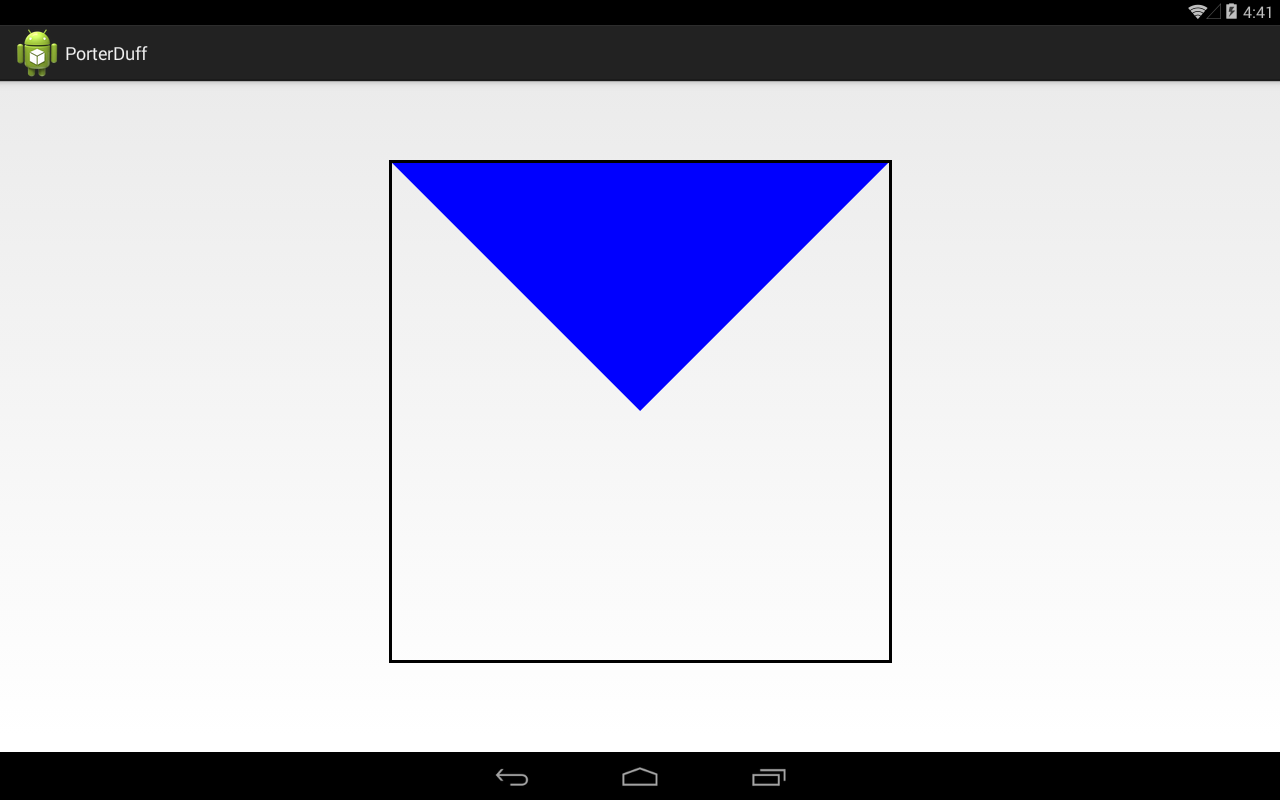
DST отображается над SRC

SRC\_IN



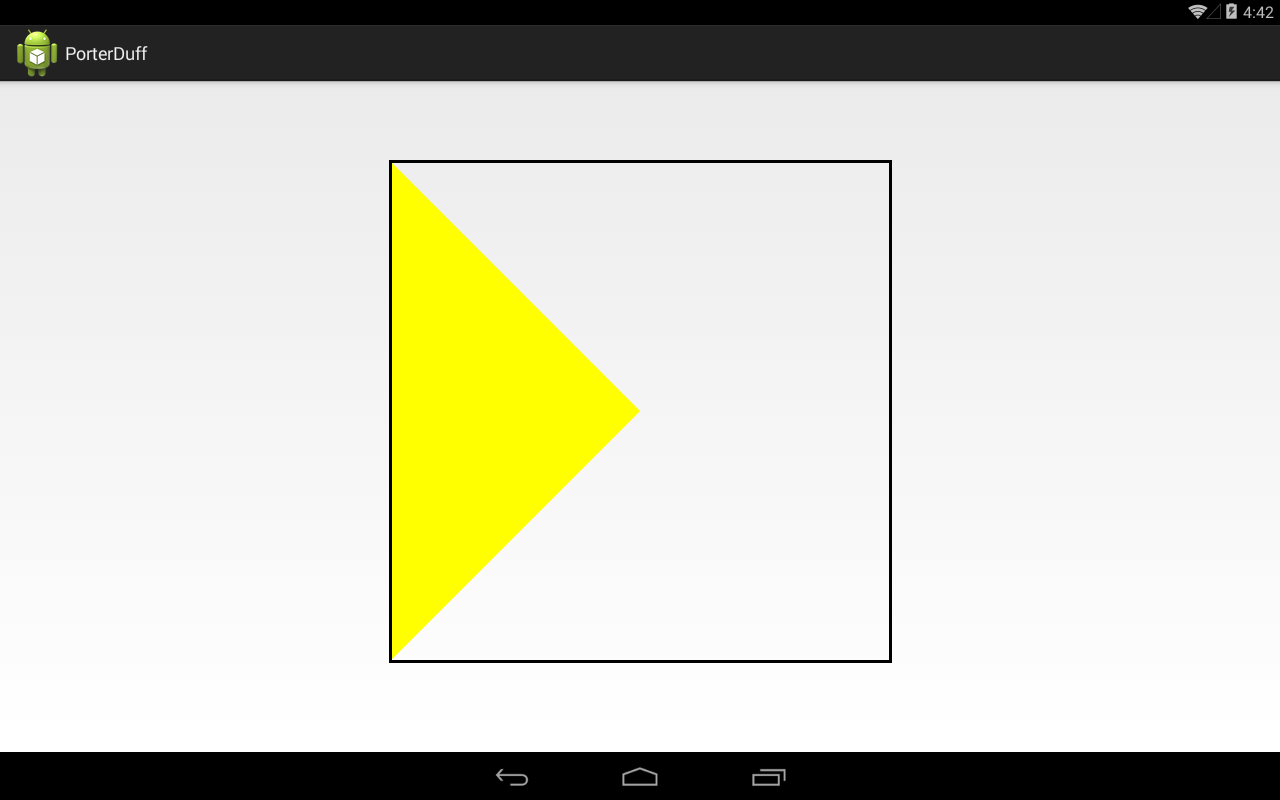
отображается часть SRC, которая пересекается с DST

DST\_IN



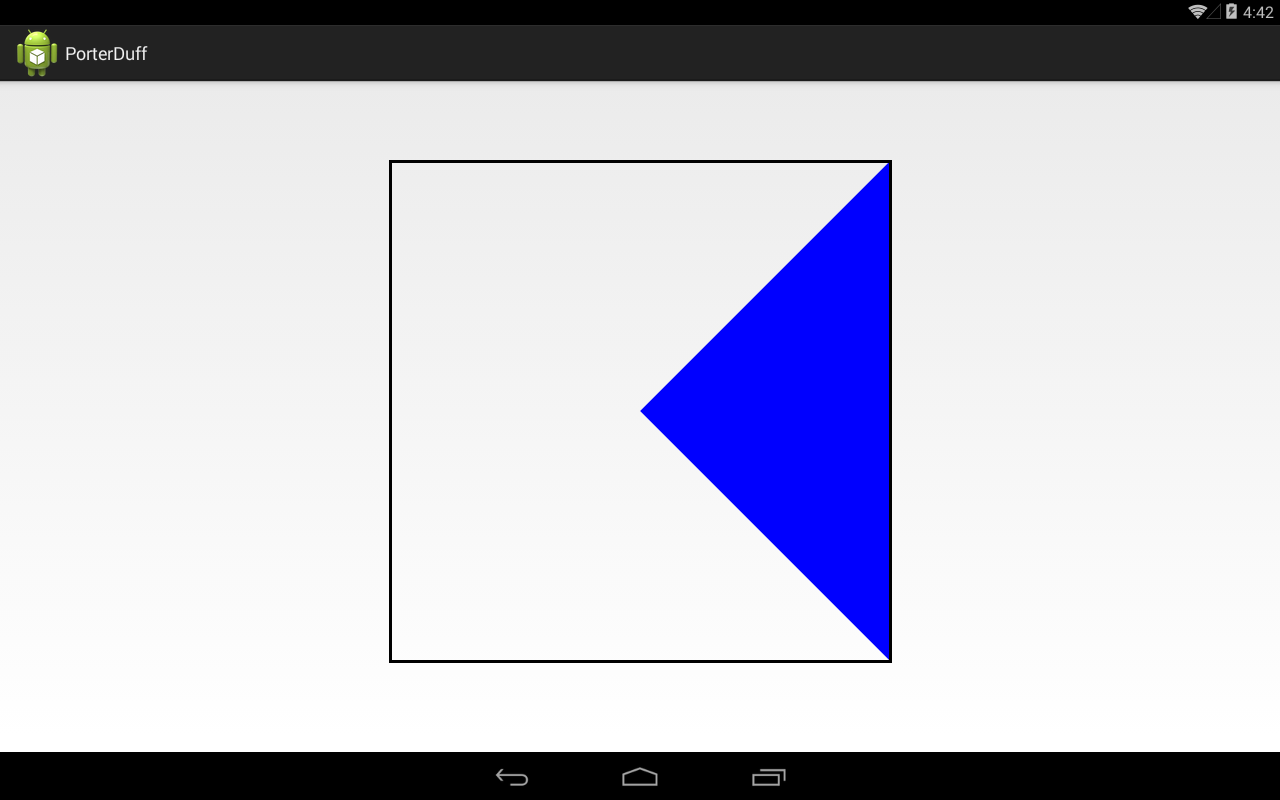
отображается часть DST, которая пересекается с SRC

SRC\_OUT



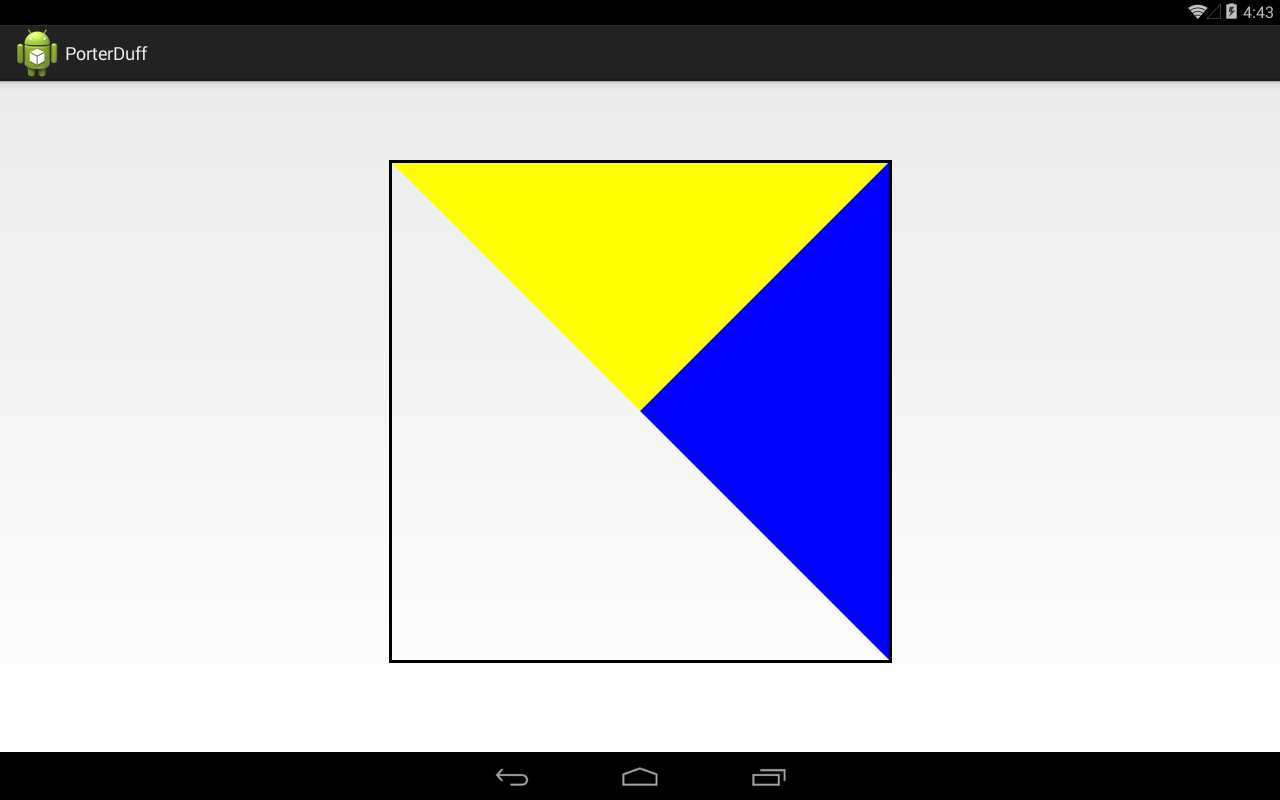
отображается часть SRC, которая не пересекается с DST

DST\_OUT



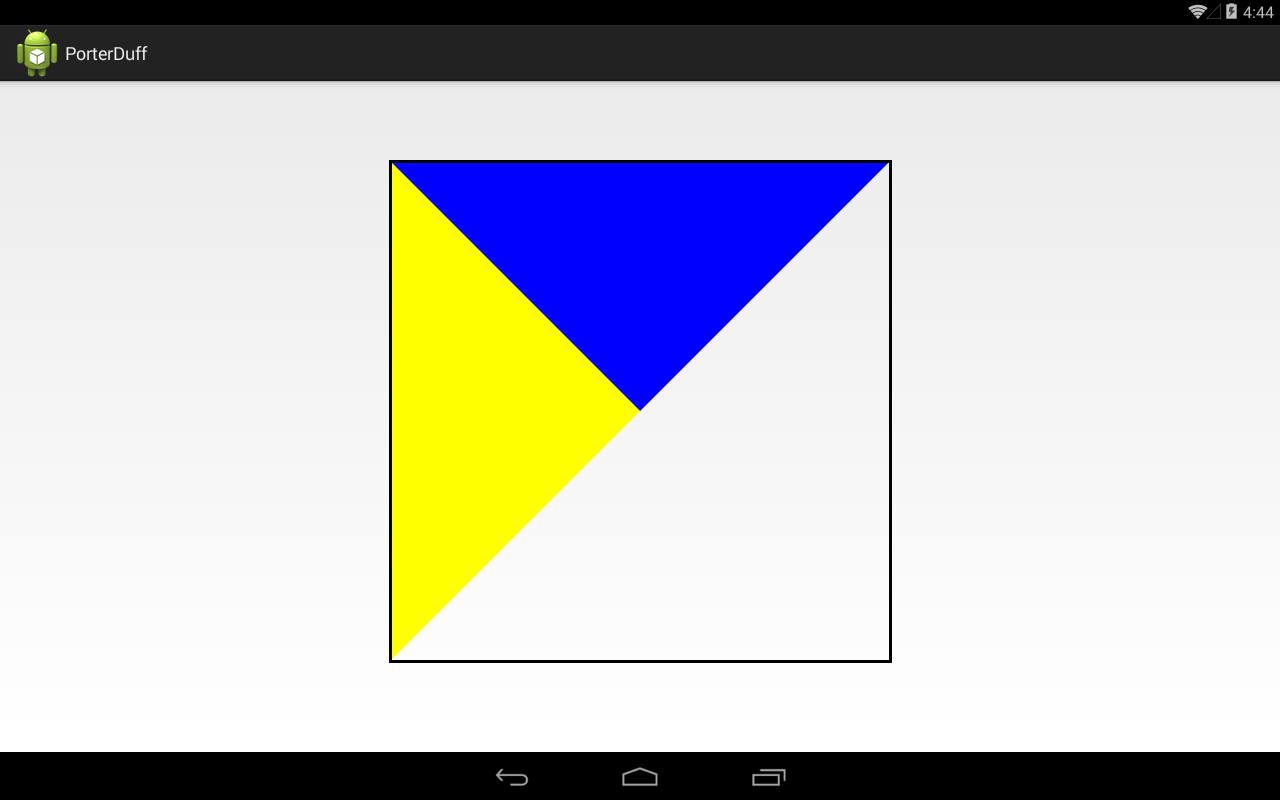
отображается часть DST, которая не пересекается с SRC

SRC\_ATOP



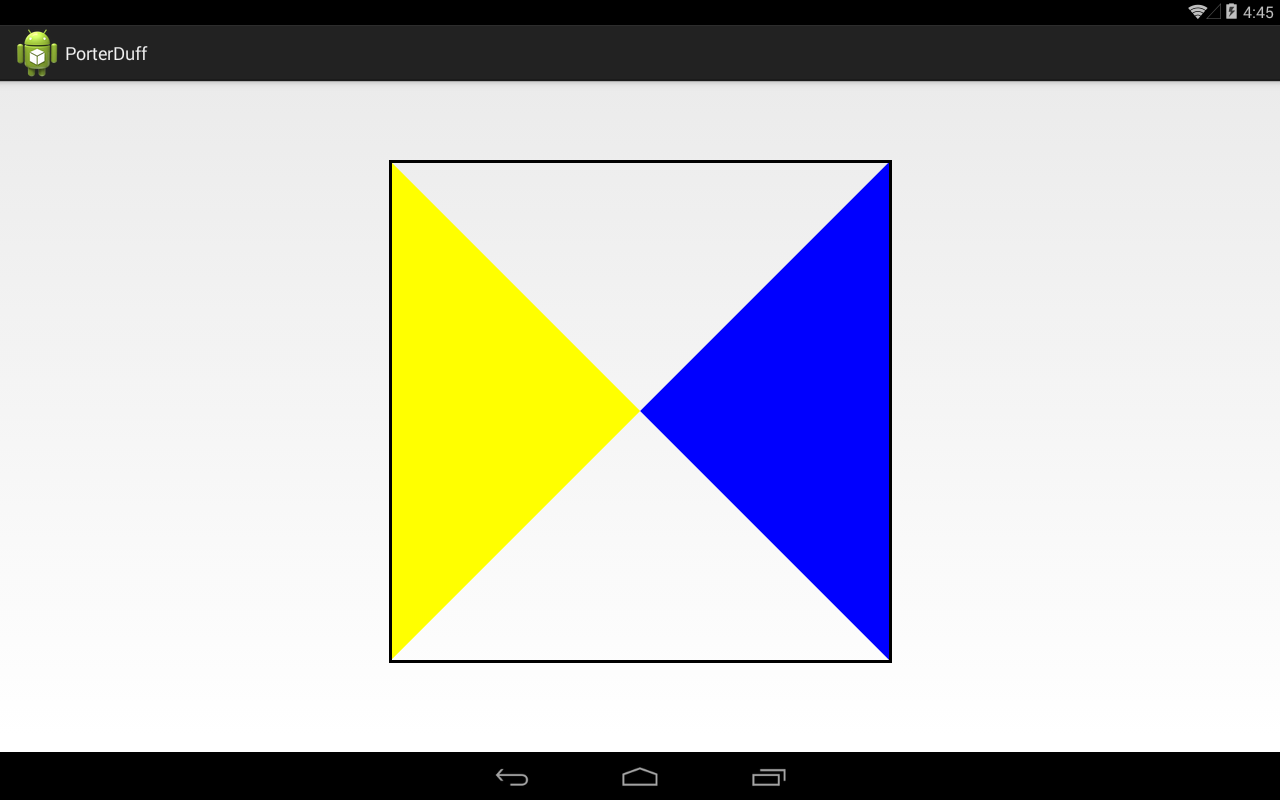
отображается DST, и поверх него часть SRC, которая пересекается с DST

DST\_ATOP



отображается SRC, и поверх него часть DST, которая пересекается с SRC

XOR

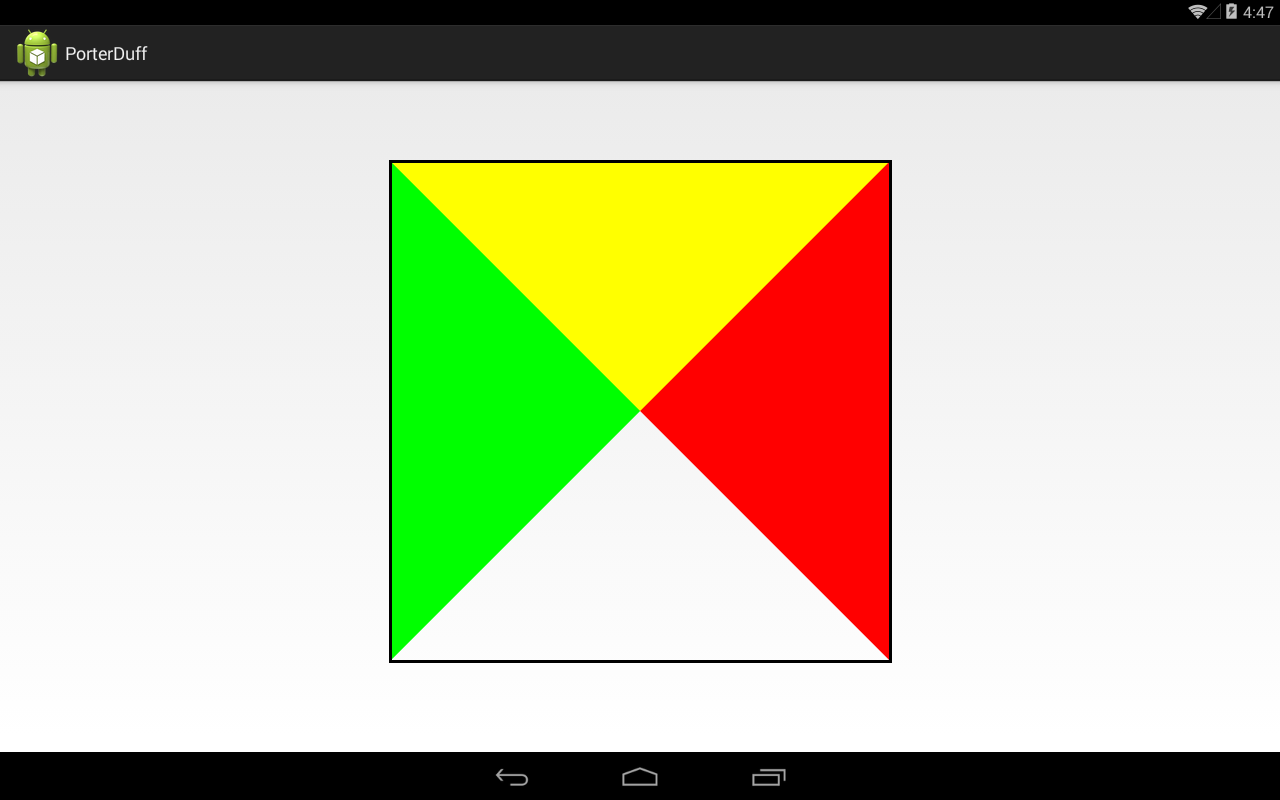


отображается SRC + DST, без части их пересечения

ADD

Для этого режима поменяем цвета следующим образом, чтобы более наглядный пример получился:

**int** colorDst = Color.RED;  
    **int** colorSrc = Color.GREEN;



Сложение цветов. Красный + зеленый при сложении дали желтый.

Мы рассмотрели режимы на самых простых случаях, когда альфа = 1. Но если поменять значения альфы, то результат будет другим. Обратите внимание на [список режимов](http://developer.android.com/reference/android/graphics/PorterDuff.Mode.html). Там для каждого режима справа показаны формулы расчета. Это две формулы, разделенные запятой. По ним рассчитываются два значения: значение прозрачности и значение цвета.  В формулах используются следующие параметры:

**Da** – альфа DST  
**Dc** – цвет DST  
**Sa** – альфа SRC  
**Sc** – цвет SRC

Т.е. используя значения цветов и прозрачностей точек изображений SRC и DST система рассчитывает значения цвета и прозрачности точек итогового изображения.

Рассмотрим один пример и сами посчитаем вручную эти значения, чтобы был понятен механизм. Для примера возьмем режим DST\_OUT. Для него формулы будут такие: [Da \* (1 - Sa), Dc \* (1 - Sa)].

Для прошлых примеров мы использовали желтый и синий цвета. Давайте немного изменим их, добавив прозрачности.

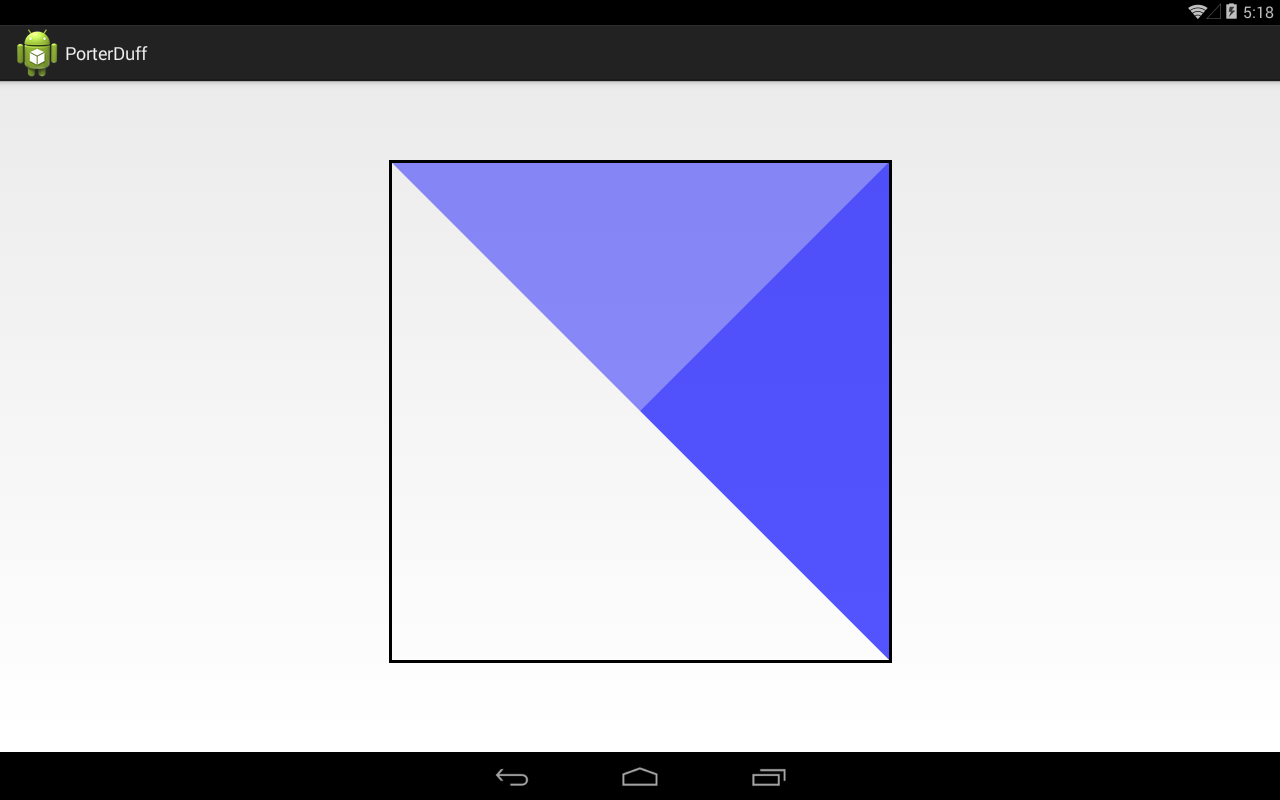
**int** colorDst = Color.argb(170, 0, 0, 255);  
    **int** colorSrc = Color.argb(85, 255, 255, 0);

Для синего (DST) поставим уровень альфа 170, а для желтого (SRC) – 85.

Режим ставим DST\_OUT.

    PorterDuff.Mode mode = PorterDuff.Mode.DST\_OUT;

Запускаем приложение, видим такую картину



Видим отличия от DST\_OUT, который у нас получался ранее. Это из-за прозрачности.

Система применяет формулы расчета попиксельно. Т.е. для каждого пиксела она берет значения Da, Dc, Sa, Sc и применяет формулу PorterDuff режима, чтобы получить результат наложения. У нас изображения одноцветные и состоят из набора одинаковых пикселов, SRC - из желтых, DST - из синих. Поэтому нам нет необходимости считать попиксельно, мы можем разделить все изображение на 4 области, в которых будет различаться набор значений Da, Dc, Sa, Sc.

Области получатся такие:

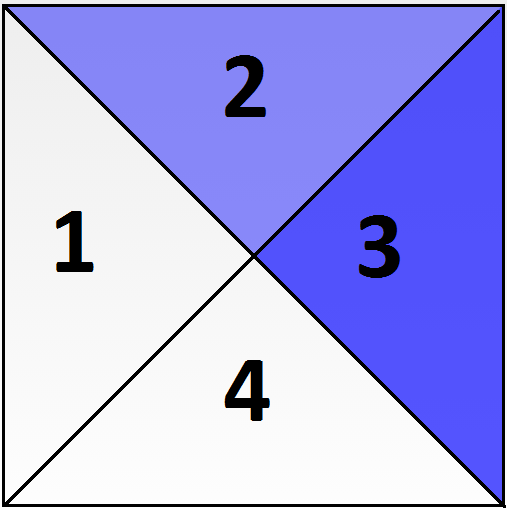
1) область SRC без пересечения с DST

2) область пересечения SRC с DST

3) область DST без пересечения с SRC

4) область вне SRC и DST

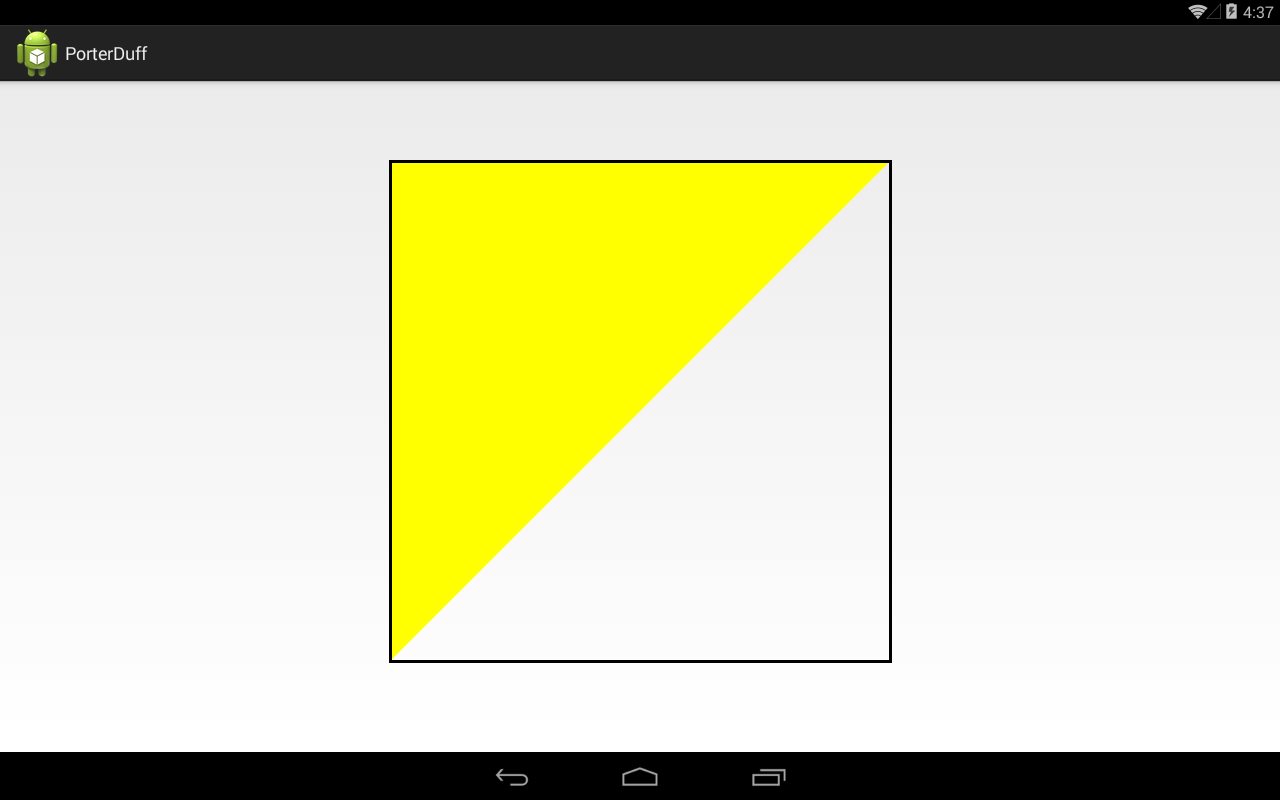
Я отобразил эти области на рисунке



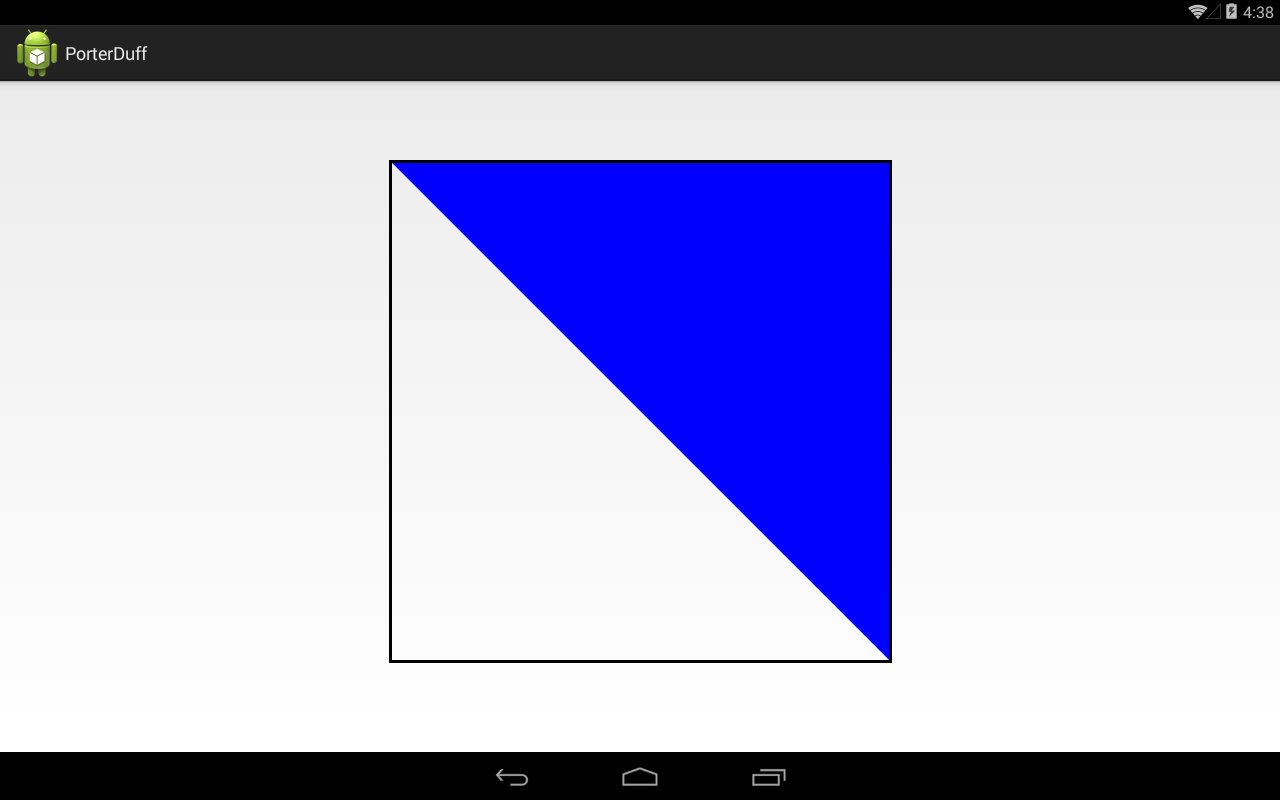
Давайте определим значения цветов и альфы для этих областей.

Напомню изображения SRC и DST, чтобы было нагляднее

SRC



DST



Используем значения colorDst (170, 0, 0, 255) и colorSrc (85, 255, 255, 0).

Da = 170, Dc = (0,0,255) - значения параметров для DST

Sa = 80, Sc = (255,255,0) - значения параметров для SRC

Сделаю небольшое отступление. В формулах мы будем значения в диапазоне от 0 до 255 приводить к диапазону от 0 до 1. Т.е., например 170 будет равно 170/255, т.е. 2/3. А 85 будет равно 85/255, т.е. 1/3. Ноль остается нолем. Ну а 255 будет равно 1.

Т.е. получим

Da = 2/3, Dc = (0,0,1)

Sa = 1/3, Sc = (1,1,0)

Итак, определяем параметры для областей:

1) Da = 0, Dc = (0,0,0), Sa = 1/3, Sc = (1,1,0)

В этой области есть только SRC. А DST тут по нулям.

2) Da = 2/3, Dc = (0,0,1), Sa = 1/3, Sc = (1,1,0)

В этой области оба изображения присутствуют

3) Da = 2/3, Dc = (0,0,1), Sa = 0, Sc = (0,0,0)

В этой области есть только DST. А SRC - по нулям.

4) Da = 0, Dc = (0,0,0), Sa = 0, Sc = (0,0,0)

В этой области оба изображения отсутствуют.

Теперь берем формулы

Ta = Da \* (1 - Sa)

Tc = Dc \* (1 - Sa)

и начинаем считать для каждой зоны Ta (итоговое значение альфа) и Tc (итоговое значение цвета)

1)

Da = 0, Dc = (0,0,0), Sa = 1/3, Sc = (1,1,0)

Ta = Da \* (1 - Sa) = 0 \* (1 – 1/3) = 0 \* 2/3 = 0

Tc = Dc \* (1 - Sa) = (0,0,0) \* (1 – 1/3) = (0,0,0) \* 2/3 = (0,0,0)

Т.е. и альфа и цвет нулевые. Итоговый ARGB будет (0,0,0,0)

2)

Da = 2/3, Dc = (0,0,1), Sa = 1/3, Sc = (1,1,0)

Ta = Da \* (1 - Sa) = 2/3 \* (1 – 1/3) = 2/3 \* 2/3 = 4/9

Tc = Dc \* (1 - Sa) = (0,0,1) \* (1 – 1/3) = (0,0,1) \* 2/3 = (0,0,2/3)

Возвращаясь к диапазону 0-255, получаем

4/9 = 4/9 \* 255 = 113

(0,0,2/3) = (0,0,170)

Итоговый ARGB будет (113,0,0,170)

3)

Da = 2/3, Dc = (0,0,1), Sa = 0, Sc = (0,0,0)

Ta = Da \* (1 - Sa) = 2/3 \* (1 - 0) = 2/3

Tc = Dc \* (1 - Sa) = (0,0,1) \* (1 - 0) = (0,0,1)

Итоговый ARGB будет (170,0,0,255)

4)

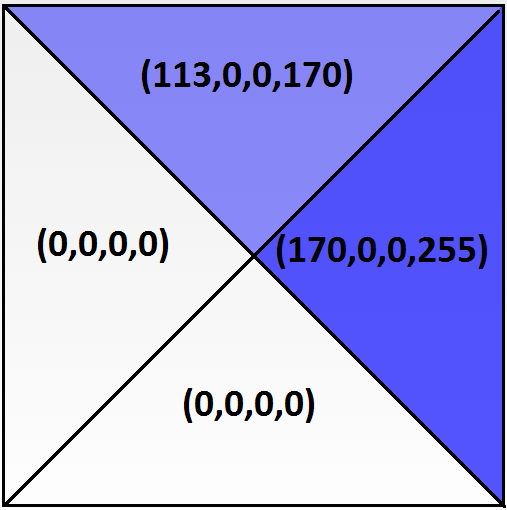
Da = 0, Dc = (0,0,0), Sa = 0, Sc = (0,0,0)

Ta = Da \* (1 - Sa) = 0 \* (1 - 0) = 0

Tc = Dc \* (1 - Sa) = (0,0,0) \* (1 - 0) = (0,0,0)

Итоговый ARGB будет (0,0,0,0)

Т.е. мы получили такую картину



Попробуйте где-нить рядом на канве нарисовать простые фигуры с полученными значениями цвета, чтобы убедиться, что все верно и цвета на картинке соответствуют значениям, которые мы рассчитали.

У нас осталось еще несколько нерассмотренных PorterDuff-режимов, их мы задействуем в следующем уроке.