Shader Шейдеры

3-й курс/Закрытая зона

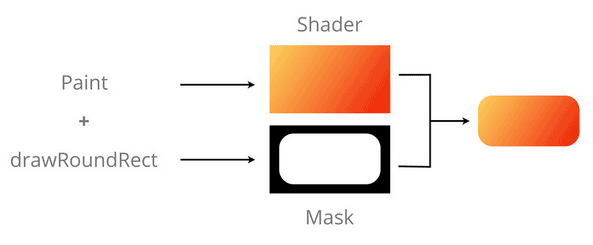
[Градиентные шейдеры](http://developer.alexanderklimov.ru/android/catshop/shader.php#gradientshader)  
[Текст на текстурной подложке](http://developer.alexanderklimov.ru/android/catshop/shader.php#pattern)  
[BitmapShader](http://developer.alexanderklimov.ru/android/catshop/shader.php#bitmapshader)  
[ComposeShader](http://developer.alexanderklimov.ru/android/catshop/shader.php#composeshader)

Расширения класса **Shader** позволяют создавать объекты [Paint](http://developer.alexanderklimov.ru/android/graphics.php#paint), которые закрашивают элементы более сложным образом, чем просто заливка сплошным цветом.

Существуют подклассы класса **android.graphics.Shader**:

* **BitmapShader**
* **ComposeShader**
* **LinearGradient**
* **RadialGradient**
* **SweepGradient**

Общая схема работы шейдера.



Чтобы использовать шейдеры при рисовании, их нужно применять к объекту **Paint** с помощью метода **setShader()**:

Paint shaderPaint = new Paint();

shaderPaint.setShader(myLinearGradient);

Все, что вы нарисуете с применением этой кисти, будет закрашиваться указанным шейдером вместо сплошного цвета.

Градиентные шейдеры

Градиентные шейдеры позволяют закрашивать контуры с помощью интерполированной цветовой гаммы. Вы можете задать градиент двумя способами.

Первый — простой переход между двумя цветами (на примере класса **LinearGradient**):

int colorFrom = Color.BLACK;

int colorTo = Color.WHITE;

LinearGradient linearGradientShader = new LinearGradient(x1, y1, x2, y2,

colorFrom, colorTo, TileMode.CLAMP);

Второй способ предусматривает описание более сложных последовательностей цветов, распределённых в заданных пропорциях.

int[] gradientColors = new int[3];

gradientColors[0] = Color.GREEN;

gradientColors[1] = Color.YELLOW;

gradientColors[2] = Color.RED;

float[] gradientPositions = new float[3];

gradientPositions[0] = 0.0f;

gradientPositions[1] = 0.5f; gradientPositions[2] = 1.0f;

RadialGradient radialGradientShader = new RadialGradient(centerX, centerY,

radius, gradientColors, gradientPositions, TileMode.CLAMP);

Каждый из градиентных шейдеров (линейный, радиальный и развёрнутый) позволяет задать градиентную заливку, используя любой из приведённых выше способов.

Использование режимов заполнения в шейдерах

Размеры кисти для градиентных шейдеров определяются с помощью явно заданных прямоугольных границ или с использованием сочетания центральной точки и длины радиуса. В растровых шейдерах размер кисти равен размерам изображения.

Если область, заданная кистью шейдера, меньше, чем область для заполнения, режим **TileMode** определяет, каким образом заполнится остальная площадь.

* CLAMP. Использует цвета по краям шейдера для заполнения дополнительного пространства.
* MIRROR. Отображает изображение шейдера по горизонтали и вертикали таким образом, чтобы каждая следующая часть соприкасалась с предыдущей.
* REPEAT. Повторяет изображение шейдера по горизонтали и вертикали, но не отображает его.

Текст с градиентом

Рассмотрим пример на **TextView**.

TextView textView = (TextView) findViewById(R.id.textView);

Shader shader = new LinearGradient(

0, 0, 0, textView.getTextSize(),

Color.RED, Color.BLUE, Shader.TileMode.CLAMP);

textView.getPaint().setShader(shader);

Мы указали два цвета для градиента. Смотрим, что получилось (сделайте размер шрифта побольше)



Усложним пример, задав несколько цветов. Реализуем в отдельном классе на основе **TextView**.

package ru.alexanderklimov.shader;

import android.content.Context;

import android.graphics.LinearGradient;

import android.graphics.Matrix;

import android.graphics.Shader;

import android.util.AttributeSet;

import android.widget.TextView;

public class RainbowTextView extends TextView {

public RainbowTextView(Context context) {

super(context);

}

public RainbowTextView(Context context, AttributeSet attrs) {

super(context, attrs);

}

public RainbowTextView(Context context, AttributeSet attrs, int defStyleAttr) {

super(context, attrs, defStyleAttr);

}

protected void onSizeChanged(int w, int h, int oldw, int oldh) {

super.onSizeChanged(w, h, oldw, oldh);

int[] rainbow = getRainbowColors();

Shader shader = new LinearGradient(0, 0, 0, w, rainbow,

null, Shader.TileMode.MIRROR);

Matrix matrix = new Matrix();

matrix.setRotate(90);

shader.setLocalMatrix(matrix);

getPaint().setShader(shader);

}

@SuppressWarnings("deprecation")

private int[] getRainbowColors() {

return new int[] {

getResources().getColor(R.color.rainbow\_red),

getResources().getColor(R.color.rainbow\_yellow),

getResources().getColor(R.color.rainbow\_green),

getResources().getColor(R.color.rainbow\_blue),

getResources().getColor(R.color.rainbow\_purple)

};

}

}

В файле **res/values/colors.xml** сохраним ресурсы цветов радуги.

<color name="rainbow\_red">#bf0c43</color>

<color name="rainbow\_yellow">#f9ba15</color>

<color name="rainbow\_green">#8eac00</color>

<color name="rainbow\_blue">#127a97</color>

<color name="rainbow\_purple">#452b72</color>

Размещаем созданный компонент на экране и смотрим результат.

<ru.alexanderklimov.shader.RainbowTextView

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:id="@+id/view"

android:layout\_gravity="center\_horizontal"

android:text="Радуга"

android:textSize="82dp"

android:gravity="center\_horizontal"/>



[Другие примеры с градиентами](http://developer.alexanderklimov.ru/android/catshop/gradient.php)

BitmapShader

Текст на текстурной подложке

Если сплошной и градиентный цвет в тексте вас не устраивает, то задействуйте текстуру. Подберите подходящую текстуру и закрасьте им символы. Желательно использовать "кошачью" текстуру - шкура леопарда, тигра и т.п. Ну, хорошо, можно использовать и другую структуру, например, кирпичную кладку.



TextView textView = (TextView) findViewById(R.id.textView);

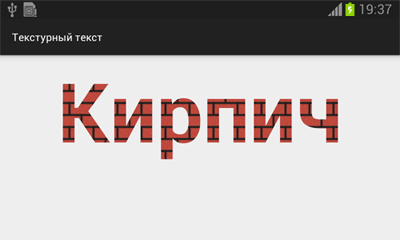
Bitmap bitmap = BitmapFactory.decodeResource(

getResources(), R.drawable.bricks); // фрагмент кирпичной кладки

Shader shader = new BitmapShader(bitmap,

Shader.TileMode.REPEAT, Shader.TileMode.REPEAT);

textView.getPaint().setShader(shader);



Другой ример использования растрового шейдера.

BitmapShader shader = new BitmapShader(bitmap, Shader.TileMode.CLAMP, Shader.TileMode.CLAMP);

Paint paint = new Paint();

paint.setAntiAlias(true);

paint.setShader(shader);

RectF rect = new RectF(0.0f, 0.0f, width, height);

canvas.drawRoundRect(rect, radius, radius, paint);

Для удобства напишем вспомогательный метод **processImage()**, которому надо передать нужный **Bitmap** для создания нужного эффекта.

package ru.alexanderklimov.test;

import android.app.Activity;

import android.graphics.Bitmap;

import android.graphics.BitmapFactory;

import android.graphics.BitmapShader;

import android.graphics.Canvas;

import android.graphics.Paint;

import android.graphics.PorterDuff;

import android.graphics.PorterDuffXfermode;

import android.graphics.RectF;

import android.os.Bundle;

import android.view.View;

import android.widget.ImageView;

public class MainActivity extends Activity {

private static final float RADIUS\_FACTOR = 8.0f;

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

ImageView imageView = (ImageView) findViewById(R.id.imageView);

Bitmap sourceBitmap = BitmapFactory.decodeResource(getResources(),

R.drawable.mybestcat);

imageView.setImageBitmap(processImage(sourceBitmap));

sourceBitmap.recycle();

}

public Bitmap processImage(Bitmap bitmap) {

Bitmap bmp;

bmp = Bitmap.createBitmap(bitmap.getWidth(), bitmap.getHeight(),

Bitmap.Config.ARGB\_8888);

BitmapShader shader = new BitmapShader(bitmap,

BitmapShader.TileMode.CLAMP, BitmapShader.TileMode.CLAMP);

float radius = Math.min(bitmap.getWidth(), bitmap.getHeight())

/ RADIUS\_FACTOR;

Canvas canvas = new Canvas(bmp);

Paint paint = new Paint();

paint.setAntiAlias(true);

paint.setShader(shader);

RectF rect = new RectF(0, 0, bitmap.getWidth(), bitmap.getHeight());

canvas.drawRoundRect(rect, radius, radius, paint);

return bmp;

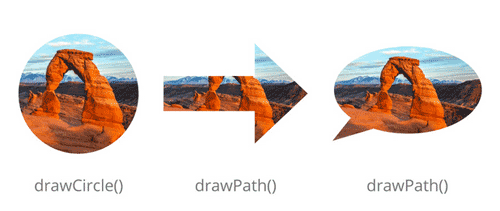
}

}

Результат будет аналогичен примеру [PorterDuffXferMode](http://developer.alexanderklimov.ru/android/catshop/porterduffxfermode.php), но при этом требует гораздо меньше памяти. Это рекомендуемый способ создания закруглённых изображений.



Шейдеры эффективно использовать для создания сложных фигур.



Если для создания овала/круга сложностей не возникнет, так как существует готовый метод, аналогичный **drawRoundRect()**, то для более сложных фигур следует использовать объект **Path**.

Комбинируя различные методы класса **Path**, например, **moveTo()** и **lineTo()**, можно создать нужную фигуру.

Например, мы можем создать фигуру, используемую в комиксах. Нам нужно создать два контура - закруглённый прямоугольник и треугольник, которые вместе создадут нужный общий контур.

package ru.alexanderklimov.test;

import android.app.Activity;

import android.graphics.Bitmap;

import android.graphics.BitmapFactory;

import android.graphics.BitmapShader;

import android.graphics.Canvas;

import android.graphics.Paint;

import android.graphics.Path;

import android.graphics.RectF;

import android.os.Bundle;

import android.view.View;

import android.widget.ImageView;

public class MainActivity extends Activity {

private static final float RADIUS\_FACTOR = 8.0f;

private static final int TRIANGLE\_WIDTH = 120;

private static final int TRIANGLE\_HEIGHT = 100;

private static final int TRIANGLE\_OFFSET = 300;

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

ImageView imageView = (ImageView) findViewById(R.id.imageView);

Bitmap sourceBitmap = BitmapFactory.decodeResource(getResources(),

R.drawable.mybestcat);

imageView.setImageBitmap(processImage(sourceBitmap));

sourceBitmap.recycle();

}

public Bitmap processImage(Bitmap bitmap) {

Bitmap bmp;

bmp = Bitmap.createBitmap(bitmap.getWidth(), bitmap.getHeight(),

Bitmap.Config.ARGB\_8888);

BitmapShader shader = new BitmapShader(bitmap,

BitmapShader.TileMode.CLAMP, BitmapShader.TileMode.CLAMP);

float radius = Math.min(bitmap.getWidth(), bitmap.getHeight())

/ RADIUS\_FACTOR;

Canvas canvas = new Canvas(bmp);

Paint paint = new Paint();

paint.setAntiAlias(true);

paint.setShader(shader);

RectF rect = new RectF(TRIANGLE\_WIDTH, 0, bitmap.getWidth(),

bitmap.getHeight());

canvas.drawRoundRect(rect, radius, radius, paint);

Path triangle = new Path();

triangle.moveTo(0, TRIANGLE\_OFFSET);

triangle.lineTo(TRIANGLE\_WIDTH, TRIANGLE\_OFFSET - (TRIANGLE\_HEIGHT / 2));

triangle.lineTo(TRIANGLE\_WIDTH, TRIANGLE\_OFFSET + (TRIANGLE\_HEIGHT / 2));

triangle.close();

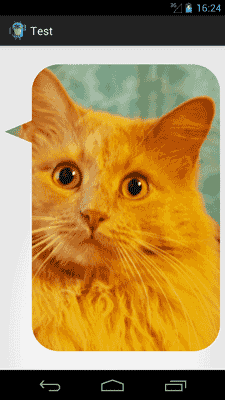
canvas.drawPath(triangle, paint);

return bmp;

}

}

Результат:



[Источник](http://blog.stylingandroid.com/archives/2686)

Создадим ещё более сложную форму в виде сердечка. Нам понадобится не только **Path**, но и **Matrix** и **Region**.

Для начала нарисуем овал.

package ru.alexanderklimov.test;

import android.app.Activity;

import android.graphics.Bitmap;

import android.graphics.BitmapFactory;

import android.graphics.BitmapShader;

import android.graphics.Canvas;

import android.graphics.Matrix;

import android.graphics.Paint;

import android.graphics.Path;

import android.graphics.RectF;

import android.graphics.Region;

import android.os.Bundle;

import android.view.View;

import android.widget.ImageView;

public class MainActivity extends Activity {

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

ImageView imageView = (ImageView) findViewById(R.id.imageView);

Bitmap sourceBitmap = BitmapFactory.decodeResource(getResources(),

R.drawable.mybestcat);

imageView.setImageBitmap(processImage(sourceBitmap));

sourceBitmap.recycle();

}

public Bitmap processImage(Bitmap bitmap) {

Bitmap bmp;

bmp = Bitmap.createBitmap(bitmap.getWidth(), bitmap.getHeight(),

Bitmap.Config.ARGB\_8888);

BitmapShader shader = new BitmapShader(bitmap,

BitmapShader.TileMode.CLAMP, BitmapShader.TileMode.CLAMP);

Canvas canvas = new Canvas(bmp);

Paint paint = new Paint();

paint.setAntiAlias(true);

paint.setShader(shader);

float width = bitmap.getWidth();

float height = bitmap.getHeight();

Path oval = new Path();

RectF ovalRect = new RectF(width / 8, 0,

width - (width / 8), height);

oval.addOval(ovalRect, Path.Direction.CW);

canvas.drawPath(oval, paint);

return bmp;

}

}



Повернём овал на 30 градусов с помощью **Matrix**:

public Bitmap processImage(Bitmap bitmap) {

Bitmap bmp;

bmp = Bitmap.createBitmap(bitmap.getWidth(), bitmap.getHeight(),

Bitmap.Config.ARGB\_8888);

BitmapShader shader = new BitmapShader(bitmap,

BitmapShader.TileMode.CLAMP, BitmapShader.TileMode.CLAMP);

Canvas canvas = new Canvas(bmp);

Paint paint = new Paint();

paint.setAntiAlias(true);

paint.setShader(shader);

float width = bitmap.getWidth();

float height = bitmap.getHeight();

Path oval = new Path();

**Matrix matrix = new Matrix();**

RectF ovalRect = new RectF(width / 8, 0,

width - (width / 8), height);

oval.addOval(ovalRect, Path.Direction.CW);

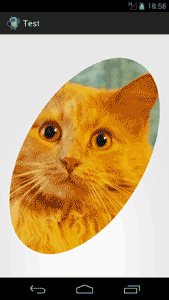
**matrix.postRotate(30, width / 2, height / 2);**

**oval.transform(matrix, oval);**

canvas.drawPath(oval, paint);

return bmp;

}



Но нам не нужен весь овал, а только его половинка. С помощью объекта **Region** отсекаем лишнее.

public Bitmap processImage(Bitmap bitmap) {

Bitmap bmp;

bmp = Bitmap.createBitmap(bitmap.getWidth(), bitmap.getHeight(),

Bitmap.Config.ARGB\_8888);

BitmapShader shader = new BitmapShader(bitmap,

BitmapShader.TileMode.CLAMP, BitmapShader.TileMode.CLAMP);

Canvas canvas = new Canvas(bmp);

Paint paint = new Paint();

paint.setAntiAlias(true);

paint.setShader(shader);

float width = bitmap.getWidth();

float height = bitmap.getHeight();

Path oval = new Path();

Matrix matrix = new Matrix();

**Region region = new Region();**

RectF ovalRect = new RectF(width / 8, 0, width - (width / 8), height);

oval.addOval(ovalRect, Path.Direction.CW);

matrix.postRotate(30, width / 2, height / 2);

oval.transform(matrix, oval);

**region.setPath(oval, new Region((int) width / 2, 0, (int) width,**

**(int) height));**

**canvas.drawPath(region.getBoundaryPath(), paint);**

return bmp;

}

У нас получилась половинка сердца.



Далее нужно повторить те же операции, повернув овал в другую сторону и отрезав половинку. Но перед этим нужно сбросить информацию у **Matrix** и **Path** с помощью метода **reset()**.

public Bitmap processImage(Bitmap bitmap) {

Bitmap bmp;

bmp = Bitmap.createBitmap(bitmap.getWidth(), bitmap.getHeight(),

Bitmap.Config.ARGB\_8888);

BitmapShader shader = new BitmapShader(bitmap,

BitmapShader.TileMode.CLAMP, BitmapShader.TileMode.CLAMP);

Canvas canvas = new Canvas(bmp);

Paint paint = new Paint();

paint.setAntiAlias(true);

paint.setShader(shader);

float width = bitmap.getWidth();

float height = bitmap.getHeight();

Path oval = new Path();

Matrix matrix = new Matrix();

Region region = new Region();

RectF ovalRect = new RectF(width / 8, 0, width - (width / 8), height);

oval.addOval(ovalRect, Path.Direction.CW);

matrix.postRotate(30, width / 2, height / 2);

oval.transform(matrix, oval);

region.setPath(oval, new Region((int) width / 2, 0, (int) width,

(int) height));

canvas.drawPath(region.getBoundaryPath(), paint);

**matrix.reset();**

**oval.reset();**

**oval.addOval(ovalRect, Path.Direction.CW);**

**matrix.postRotate(-30, width / 2, height / 2);**

**oval.transform(matrix, oval);**

**region.setPath(oval, new Region(0, 0, (int) width / 2, (int) height));**

**canvas.drawPath(region.getBoundaryPath(), paint);**

return bmp;

}



[Источник](http://blog.stylingandroid.com/archives/2692)

"Живой" шейдер

[Здесь](http://www.41post.com/4794/programming/android-rendering-a-path-with-a-bitmap-fill) нашёл интересный пример шейдера, который движется за пальцем. Получается эффект фонаря, который освещает кирпичную стенку. Попробуйте пример вживую. Не забудьте разместить в ресурсах фрагмент кирпичной стенки.



Создадим отдельный класс.

package ru.alexanderklimov.testapplication;

import android.content.Context;

import android.graphics.Bitmap;

import android.graphics.BitmapFactory;

import android.graphics.BitmapShader;

import android.graphics.Canvas;

import android.graphics.Matrix;

import android.graphics.Paint;

import android.graphics.Shader;

import android.util.AttributeSet;

import android.view.MotionEvent;

import android.view.View;

public class PathRendererView extends View {

// Кисть для заполнения компонента

private Paint mFillPaint;

// Кисть для обводки

private Paint mStrokePaint;

// Шейдер

private BitmapShader mFillBitmapShader;

private Matrix mMatrix = new Matrix();

// Координаты касания на экране

private float mPosX = 105;

private float mPosY = 105;

public PathRendererView(Context context, AttributeSet attrs) {

super(context, attrs);

this.setFocusable(true);

mStrokePaint = new Paint();

mStrokePaint.setDither(true);

mStrokePaint.setColor(0xFFFFFF00);

mStrokePaint.setStyle(Paint.Style.STROKE);

mStrokePaint.setAntiAlias(true);

mStrokePaint.setStrokeWidth(3);

// Загружаем картинку из ресурсов

Bitmap brickBitmap = BitmapFactory.decodeResource(context.getResources(), R.drawable.bricks);

// Инициализируем шейдер из картинки и используем как текстуру

mFillBitmapShader = new BitmapShader(brickBitmap, Shader.TileMode.REPEAT, Shader.TileMode.REPEAT);

mFillPaint = new Paint();

mFillPaint.setColor(0xFFFFFFFF);

mFillPaint.setStyle(Paint.Style.FILL);

mFillPaint.setShader(mFillBitmapShader);

}

@Override

public boolean onTouchEvent(MotionEvent event) {

if (event.getAction() == MotionEvent.ACTION\_MOVE) {

mPosX = event.getX();

mPosY = event.getY();

invalidate();

}

return true;

}

@Override

protected void onDraw(Canvas canvas) {

//Invert the current matrix, so that the Bitmap background stays at the same position.

//Only necessary if the canvas matrix is being transformed in some way.

canvas.getMatrix().invert(mMatrix);

//Assign the matrix to the BitmapShader. Again, not required to make this example work.

mFillBitmapShader.setLocalMatrix(mMatrix);

//Draw the fill

canvas.drawCircle(mPosX, mPosY, 100, mFillPaint);

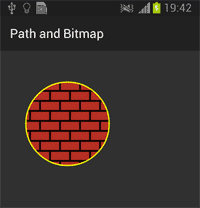
//Afterwards, draw the circle again, using the stroke paint

canvas.drawCircle(mPosX, mPosY, 100, mStrokePaint);

}

}

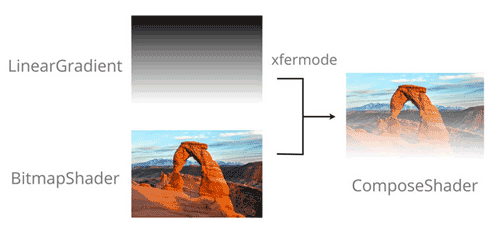
Подключаем компонент в разметке активности и смотрим на пример.



ComposeShader

Наиболее часто шейдеры используются для описания градиентной заливки. При помощи градиентов вы можете легко придать двумерному рисунку глубину и фактуру. Android поддерживает три градиентных шейдера (помимо растровых и композитных).

Общая схема.



Эффект виньетки

RadialGradient vignette = new RadialGradient(mRect.centerX(), mRect.centerY(), radius,

new int[] {0, 0, 0x7f000000},

new float[] {0.0f, 0.7f, 1.0f},

Shader.TileMode.CLAMP);

Matrix oval = new Matrix();

oval.setScale(1.0f, 0.7f);

vignette.setLocalMatrix(oval);

mPaint.setShader(new ComposeShader(mBitmapShader, vignette, PorterDuff.Mode.SRC\_OVER));