 Работа с SurfaceView в Android [из песочницы](https://habrahabr.ru/sandbox/)

При работе с 2D графикой в Android отрисовку можно выполнять используя Canvas. Проще всего это сделать с помощью своего класса, унаследованного от View. Необходимо просто описать метод onDraw(), и использовать предоставленный в качестве параметра canvas для выполнения всех необходимых действий. Однако этот подход имеет свои недостатки. Метод onDraw() вызывается системой. Вручную же можно использовать метод invalidate(), говорящий системе о необходимости перепрорисовки. Но вызов invalidate() не гарантирует незамедлительного вызова метода onDraw(). Поэтому, если нам необходимо постоянно делать отрисовку (например для какой-либо игры), вышеописанный способ вряд ли стоит считать подходящим.   
  
Существует еще один подход — с использованием класса SurfaceView. Почитав официальное руководство и изучив несколько примеров, решил написать небольшую статью на русском, которая возможно поможет кому-то быстрее освоиться с данным способом выполнения отрисовки. Статья рассчитана на новичков. Никаких сложных и хитрых приемов здесь не описано.

Класс SurfaceView

Особенность класса SurfaceView заключается в том, что он предоставляет отдельную область для рисования, действия с которой должны быть вынесены в отдельный поток приложения. Таким образом, приложению не нужно ждать, пока система будет готова к отрисовке всей иерархии view-элементов. Вспомогательный поток может использовать canvas нашего SurfaceView для отрисовки с той скоростью, которая необходима.  
  
Вся реализация сводится к двум основным моментам:

1. Создание класса, унаследованного от SurfaceView и реализующего интерфейс SurfaceHolder.Callback
2. Создание потока, который будет управлять отрисовкой.

Создание класса

Как было сказано выше, нам потребуется свой класс, расширяющий SurfaceView и реализующий интерфейс SurfaceHolder.Callback. Этот интерфейс предлагает реализовать три метода: surfaceCreated(), surfaceChanged() и surfaceDestroyed(), вызываемые соответственно при создании области для рисования, ее изменении и разрушении.  
Работа с полотном для рисования осуществляется не напрямую через созданный нами класс, а с помощью объекта SurfaceHolder. Получить его можно вызовом метода getHolder(). Именно этот объект будет предоставлять нам canvas для отрисовки.

**public** **class** **MySurfaceView** **extends** **SurfaceView** **implements** **SurfaceHolder**.**Callback** {

**public** **MySurfaceView**(Context context) {

**super**(context);

getHolder().addCallback(**this**);

}

**@Override**

**public** **void** **surfaceChanged**(SurfaceHolder holder, **int** format, **int** width,

**int** height) {

}

**@Override**

**public** **void** **surfaceCreated**(SurfaceHolder holder) {

}

**@Override**

**public** **void** **surfaceDestroyed**(SurfaceHolder holder) {

}

}

В конструкторе класса получаем объект SurfaceHolder и с помощью метода addCallback() указываем что хотим получать соответствующие обратные вызовы.   
В методе surfaceCreated(), как правило, необходимо начинать выполнять отрисовку, а в surfaceDestroyed() наоборот завершать ее. Оставим пока тела этих методов пустыми и реализуем поток, отвечающий за отрисовку.

Реализация потока отрисовки

Создадим класс, унаследованный от Thread. В конструкторе он будет принимать два параметра: SurfaceHolder и Resources для загрузки картинки, которую мы будем рисовать на экране. Также в классе нам понадобится переменная-флаг, указывающая на то, что производится отрисовка и метод для установки этой переменной. Ну и, конечно, потребуется переопределить метод run().  
В итоге получим следующий класс:

**class** **DrawThread** **extends** **Thread**{

**private** **boolean** runFlag = **false**;

**private** SurfaceHolder surfaceHolder;

**private** Bitmap picture;

**private** Matrix matrix;

**private** **long** prevTime;

**public** **DrawThread**(SurfaceHolder surfaceHolder, Resources resources){

**this**.surfaceHolder = surfaceHolder;

*// загружаем картинку, которую будем отрисовывать*

picture = BitmapFactory.decodeResource(resources, R.drawable.icon);

*// формируем матрицу преобразований для картинки*

matrix = **new** Matrix();

matrix.postScale(3.0f, 3.0f);

matrix.postTranslate(100.0f, 100.0f);

*// сохраняем текущее время*

prevTime = System.currentTimeMillis();

}

**public** **void** **setRunning**(**boolean** run) {

runFlag = run;

}

**@Override**

**public** **void** **run**() {

Canvas canvas;

**while** (runFlag) {

*// получаем текущее время и вычисляем разницу с предыдущим*

*// сохраненным моментом времени*

**long** now = System.currentTimeMillis();

**long** elapsedTime = now - prevTime;

**if** (elapsedTime > 30){

*// если прошло больше 30 миллисекунд - сохраним текущее время*

*// и повернем картинку на 2 градуса.*

*// точка вращения - центр картинки*

prevTime = now;

matrix.preRotate(2.0f, picture.getWidth() / 2, picture.getHeight() / 2);

}

canvas = **null**;

**try** {

*// получаем объект Canvas и выполняем отрисовку*

canvas = surfaceHolder.lockCanvas(**null**);

**synchronized** (surfaceHolder) {

canvas.drawColor(Color.BLACK);

canvas.drawBitmap(picture, matrix, **null**);

}

}

**finally** {

**if** (canvas != **null**) {

*// отрисовка выполнена. выводим результат на экран*

surfaceHolder.unlockCanvasAndPost(canvas);

}

}

}

}

}

Для того, чтобы результат не выглядел совсем уж пресно, загруженную картинку увеличим в три раза, и немного сместим к центру экрана. Сделаем это при помощи матрицы преобразований. Также, не чаще чем раз в 30 миллисекунд, будем поворачивать картинку на 2 градуса вокруг ее центра. Само рисование на canvas лучше, конечно, вынести в отдельный метод, но в данном случае мы всего лишь очищаем экран и рисуем изображение. Так что можно оставить как есть.

Начало и завершение отрисовки

Теперь, после того как мы написали поток, управляющий отрисовкой, вернемся к редактированию нашего класса SurfaceView. В методе surfaceCreated() создадим поток и запустим его. А в методе surfaceDestroyed() завершим его работу. В итоге класс MySurfaceView примет следующий вид:

**public** **class** **MySurfaceView** **extends** **SurfaceView** **implements** **SurfaceHolder**.**Callback** {

**private** DrawThread drawThread;

**public** **MySurfaceView**(Context context) {

**super**(context);

getHolder().addCallback(**this**);

}

**@Override**

**public** **void** **surfaceChanged**(SurfaceHolder holder, **int** format, **int** width,

**int** height) {

}

**@Override**

**public** **void** **surfaceCreated**(SurfaceHolder holder) {

drawThread = **new** DrawThread(getHolder(), getResources());

drawThread.setRunning(**true**);

drawThread.start();

}

**@Override**

**public** **void** **surfaceDestroyed**(SurfaceHolder holder) {

**boolean** retry = **true**;

*// завершаем работу потока*

drawThread.setRunning(**false**);

**while** (retry) {

**try** {

drawThread.join();

retry = **false**;

} **catch** (InterruptedException e) {

*// если не получилось, то будем пытаться еще и еще*

}

}

}

}

Следует отметить, что создание потока должно выполняться в методе surfaceCreated(). В примере LunarLander из официальной документации создание потока отрисовки происходит в конструкторе класса, унаследованного от SurfaceView. Но при таком подходе может возникнуть ошибка. Если свернуть приложение нажатием клавиши Home на устройстве, а затем снова открыть его, то возникнет исключительная ситуация IllegalThreadStateException.  
  
Activity приложения может выглядеть следующим образом:

**public** **class** **SurfaceViewActivity** **extends** **Activity** {

*/\*\* Called when the activity is first created. \*/*

**@Override**

**public** **void** **onCreate**(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(**new** MySurfaceView(**this**));

}

}

Результат работы программы выглядит так (из за вращения изображение немного размазано, но на устройстве выглядит вполне приемлемо):  


Заключение

Вышеописанный способ отрисовки графики хоть и несколько сложнее чем рисование на canvas простого View, но в некоторых случаях является более предпочтительным. Разницу можно заметить, если в приложении необходимо очень часто перерисовывать графику. Вспомогательный поток помогает лучше контролировать этот процесс.  
И в заключении хотелось бы привести ссылки на некоторые из ресурсов, которыми пользовался при создании статьи:

1. [Пример игры LunarLander из официальной документации](http://developer.android.com/resources/samples/LunarLander/index.html)
2. [Статья, описывающая использование SurfaceView](http://www.droidnova.com/playing-with-graphics-in-android-part-ii,160.html)