**SurfaceView**

[SurfaceView с поддержкой касаний](http://developer.alexanderklimov.ru/android/views/surfaceview.php#touch)  
[Гроза эпилептиков](http://developer.alexanderklimov.ru/android/views/surfaceview.php#epilectic)  
[Поймай меня](http://developer.alexanderklimov.ru/android/views/surfaceview.php#catchme)  
[Отскакивающие значки](http://developer.alexanderklimov.ru/android/views/surfaceview.php#touchbounce)

Класс SurfaceView предоставляет объект Surface, который поддерживает рисование в фоновом потоке и дает возможность использовать OpenGL для трехмерной графики. Это отличный вариант для насыщенных графикой элементов, которые нуждаются в частых обновлениях или должны отображать сложную графическую информацию, как в случае с играми и трехмерной визуализацией.

SurfaceView — обертка вокруг класса SurfaceHolder, который в свою очередь служит оберткой класса Surface, используемого для обновления изображения из фоновых потоков.

Найти данный элемент можно в папке **Advanced**.

В основе SurfaceView объект Surface, а не Canvas. Это важно, потому как Surface поддерживает рисование из фоновых потоков. Данное отличие особенно полезно для ресурсоемких операций или быстрых обновлений, а также когда необходимо обеспечить высокую частоту изменения кадров (использование трехмерной графики, создание игр или предпросмотр видеопотока с камеры в режиме реального времени.

Возможность рисовать вне зависимости от графического потока ведет к повышенному потреблению памяти. Таким образом, хоть это и полезный (а иногда просто необходимый) способ создания нестандартных представлений, будьте осторожны, используя его.

SurfaceView используется точно таким же образом, как любые производные от View классы. Вы можете применять анимацию и размещать их внутри разметки так же, как и другие представления.

Применяя OpenGL, вы можете рисовать на Surface любые поддерживаемые двумерные или трехмерные объекты, получая при этом все выгоды от аппаратного ускорения (если таковое имеется). Таким образом, вы значительно повышаете производительность, если сравнивать с теми же операциями, выполненными на двумерном Canvas.

Объекты SurfaceView особенно пригодятся для отображения динамических трехмерных изображений, к примеру, в интерактивных играх, их можно назвать лучшим выбором для отображения предварительного просмотра видеопотоков с камеры в режиме реального времени.

**Создание нового объекта SurfaceView**

Чтобы создать данный тип, наследуйте класс SurfaceView и реализуйте интерфейс SurfaceHolder.Callback, описывающий функцию обратного вызова. Он уведомляет представление о том, что исходный объект Surface был создан/уничтожен/модифицирован и передает в объект SurfaceHolder ссылку, содержащую допустимый экземпляр Surface.

Типичный шаблон проектирования SurfaceView предусматривает классы, производные от Thread, которые принимают ссылку на текущий объект SurfaceHolder и немедленно его обновляют.

**SurfaceView с поддержкой касаний**

Напишем простой пример использования SurfaceView, на котором можно рисовать линии.

package ru.alexanderklimov.surfacedemo;

import java.util.Random;

import android.app.Activity;

import android.content.Context;

import android.graphics.Canvas;

import android.graphics.Color;

import android.graphics.Paint;

import android.graphics.Path;

import android.os.Bundle;

import android.view.MotionEvent;

import android.view.SurfaceHolder;

import android.view.SurfaceView;

import android.view.Window;

public class TestActivity extends Activity {

@Override

public void onCreate(Bundle icicle) {

super.onCreate(icicle);

requestWindowFeature(Window.FEATURE\_NO\_TITLE);

setContentView(new MySurfaceView(this));

}

class MySurfaceView extends SurfaceView {

Path path;

Thread thread = null;

SurfaceHolder surfaceHolder;

volatile boolean running = false;

private Paint paint = new Paint(Paint.ANTI\_ALIAS\_FLAG);

Random random;

public MySurfaceView(Context context) {

super(context);

surfaceHolder = getHolder();

paint.setStyle(Paint.Style.STROKE);

paint.setStrokeWidth(3);

paint.setColor(Color.WHITE);

}

@Override

public boolean onTouchEvent(MotionEvent event) {

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

path = new Path();

path.moveTo(event.getX(), event.getY());

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

path.lineTo(event.getX(), event.getY());

} else if (event.getAction() == MotionEvent.ACTION\_UP) {

path.lineTo(event.getX(), event.getY());

}

if (path != null) {

Canvas canvas = surfaceHolder.lockCanvas();

canvas.drawPath(path, paint);

surfaceHolder.unlockCanvasAndPost(canvas);

}

return true;

}

}

}

Запустив проект, вы можете рисовать пальцем по экрану.

**Гроза эпилептиков**

Попробовал [данный пример](http://android-er.blogspot.ru/2010/05/android-surfaceview.html). При рисовании рисуются окружности, которые постоянно мигают. Жуткое зрелище.

В примере реализовано:

* новый класс MySurfaceView, наследующий от SurfaceView и реализующий интерфейс SurfaceHolder.Callback
* используются методы surfaceCreated(), surfaceDestroyed(), surfaceChanged() для SurfaceHolder.Callback
* используется класс Thread для SurfaceView

package ru.alexanderklimov.test;

import android.app.Activity;

import android.content.Context;

import android.graphics.Canvas;

import android.graphics.Color;

import android.graphics.Paint;

import android.os.Bundle;

import android.util.AttributeSet;

import android.view.MotionEvent;

import android.view.SurfaceHolder;

import android.view.SurfaceView;

import android.view.Window;

public class TestActivity extends Activity {

private Paint paint = new Paint(Paint.ANTI\_ALIAS\_FLAG);

private float initX, initY, radius;

private boolean drawing = false;

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

requestWindowFeature(Window.FEATURE\_NO\_TITLE);

setContentView(new MySurfaceView(this));

}

public class MySurfaceThread extends Thread {

private SurfaceHolder myThreadSurfaceHolder;

private MySurfaceView myThreadSurfaceView;

private boolean myThreadRun = false;

public MySurfaceThread(SurfaceHolder surfaceHolder,

MySurfaceView surfaceView) {

myThreadSurfaceHolder = surfaceHolder;

myThreadSurfaceView = surfaceView;

}

public void setRunning(boolean b) {

myThreadRun = b;

}

@Override

public void run() {

// TODO Auto-generated method stub

// super.run();

while (myThreadRun) {

Canvas c = null;

try {

c = myThreadSurfaceHolder.lockCanvas(null);

synchronized (myThreadSurfaceHolder) {

myThreadSurfaceView.onDraw(c);

}

} finally {

// do this in a finally so that if an exception is thrown

// during the above, we don't leave the Surface in an

// inconsistent state

if (c != null) {

myThreadSurfaceHolder.unlockCanvasAndPost(c);

}

}

}

}

}

public class MySurfaceView extends SurfaceView implements

SurfaceHolder.Callback {

private MySurfaceThread thread;

@Override

protected void onDraw(Canvas canvas) {

// TODO Auto-generated method stub

// super.onDraw(canvas);

if (drawing) {

canvas.drawCircle(initX, initY, radius, paint);

}

}

@Override

public boolean onTouchEvent(MotionEvent event) {

// TODO Auto-generated method stub

// return super.onTouchEvent(event);

int action = event.getAction();

if (action == MotionEvent.ACTION\_MOVE) {

float x = event.getX();

float y = event.getY();

radius = (float) Math.sqrt(Math.pow(x - initX, 2)

+ Math.pow(y - initY, 2));

} else if (action == MotionEvent.ACTION\_DOWN) {

initX = event.getX();

initY = event.getY();

radius = 1;

drawing = true;

} else if (action == MotionEvent.ACTION\_UP) {

drawing = false;

}

return true;

}

public MySurfaceView(Context context) {

super(context);

// TODO Auto-generated constructor stub

init();

}

public MySurfaceView(Context context, AttributeSet attrs) {

super(context, attrs);

// TODO Auto-generated constructor stub

init();

}

public MySurfaceView(Context context, AttributeSet attrs, int defStyle) {

super(context, attrs, defStyle);

// TODO Auto-generated constructor stub

init();

}

private void init() {

getHolder().addCallback(this);

setFocusable(true); // make sure we get key events

paint.setStyle(Paint.Style.STROKE);

paint.setStrokeWidth(3);

paint.setColor(Color.WHITE);

}

@Override

public void surfaceChanged(SurfaceHolder arg0, int arg1, int arg2,

int arg3) {

// TODO Auto-generated method stub

}

@Override

public void surfaceCreated(SurfaceHolder holder) {

// TODO Auto-generated method stub

thread = new MySurfaceThread(getHolder(), this);

thread.setRunning(true);

thread.start();

}

@Override

public void surfaceDestroyed(SurfaceHolder holder) {

// TODO Auto-generated method stub

boolean retry = true;

thread.setRunning(false);

while (retry) {

try {

thread.join();

retry = false;

} catch (InterruptedException e) {

}

}

}

}

}

**Поймай меня**

Тот автор на основе предыдущего примера сделал [новый пример](http://android-er.blogspot.ru/2010/05/android-surfaceview-run-in-thread-with.html). На экране рисуется белая точка, которая начинает двигаться в место касания экрана пальцем пользователя.

package ru.alexanderklimov.test;

import ...

public class TestActivity extends Activity {

private Paint paint = new Paint(Paint.ANTI\_ALIAS\_FLAG);

private float initX, initY;

private float targetX, targetY;

private boolean drawing = true;

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

requestWindowFeature(Window.FEATURE\_NO\_TITLE);

setContentView(new MySurfaceView(this));

}

public class MySurfaceThread extends Thread {

private SurfaceHolder myThreadSurfaceHolder;

private MySurfaceView myThreadSurfaceView;

private boolean myThreadRun = false;

public MySurfaceThread(SurfaceHolder surfaceHolder,

MySurfaceView surfaceView) {

myThreadSurfaceHolder = surfaceHolder;

myThreadSurfaceView = surfaceView;

}

public void setRunning(boolean b) {

myThreadRun = b;

}

@Override

public void run() {

// TODO Auto-generated method stub

// super.run();

while (myThreadRun) {

Canvas c = null;

try {

c = myThreadSurfaceHolder.lockCanvas(null);

synchronized (myThreadSurfaceHolder) {

myThreadSurfaceView.onDraw(c);

}

sleep(500);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} finally {

// do this in a finally so that if an exception is thrown

// during the above, we don't leave the Surface in an

// inconsistent state

if (c != null) {

myThreadSurfaceHolder.unlockCanvasAndPost(c);

}

}

}

}

}

public class MySurfaceView extends SurfaceView implements

SurfaceHolder.Callback {

private MySurfaceThread thread;

@Override

protected void onDraw(Canvas canvas) {

// TODO Auto-generated method stub

// super.onDraw(canvas);

if (drawing) {

canvas.drawRGB(0, 0, 0);

canvas.drawCircle(initX, initY, 3, paint);

if ((initX == targetX) && (initY == targetY)) {

drawing = false;

} else {

initX = (initX + targetX) / 2;

initY = (initY + targetY) / 2;

}

}

}

@Override

public boolean onTouchEvent(MotionEvent event) {

// TODO Auto-generated method stub

// return super.onTouchEvent(event);

int action = event.getAction();

if (action == MotionEvent.ACTION\_DOWN) {

targetX = event.getX();

targetY = event.getY();

drawing = true;

}

return true;

}

public MySurfaceView(Context context) {

super(context);

// TODO Auto-generated constructor stub

init();

}

public MySurfaceView(Context context, AttributeSet attrs) {

super(context, attrs);

// TODO Auto-generated constructor stub

init();

}

public MySurfaceView(Context context, AttributeSet attrs, int defStyle) {

super(context, attrs, defStyle);

// TODO Auto-generated constructor stub

init();

}

private void init() {

getHolder().addCallback(this);

thread = new MySurfaceThread(getHolder(), this);

setFocusable(true); // make sure we get key events

paint.setStyle(Paint.Style.STROKE);

paint.setStrokeWidth(3);

paint.setColor(Color.WHITE);

initX = targetX = 0;

initY = targetY = 0;

}

@Override

public void surfaceChanged(SurfaceHolder arg0, int arg1, int arg2,

int arg3) {

// TODO Auto-generated method stub

}

@Override

public void surfaceCreated(SurfaceHolder holder) {

// TODO Auto-generated method stub

thread.setRunning(true);

thread.start();

}

@Override

public void surfaceDestroyed(SurfaceHolder holder) {

// TODO Auto-generated method stub

boolean retry = true;

thread.setRunning(false);

while (retry) {

try {

thread.join();

retry = false;

} catch (InterruptedException e) {

}

}

}

}

}

Этот же пример, но с использованием разметки FrameLayout, внутри которой находится SurfaceView, находится [здесь](http://android-er.blogspot.ru/2010/05/another-exercise-of-surfaceview-in.html). Не буду приводить код.

Продолжение серии примеров - [Exercise of SurfaceView: SurfaceView overlap with a LinearLayout](http://android-er.blogspot.ru/2010/06/exercise-of-surfaceview-surfaceview.html). Здесь появляется возможность выбрать цвет точки.

**Отскакивающие значки**

Ещё один пример - при касании экрана в области SurfaceView будет появляться новая картинка, которая будет двигаться внутри контейнера, отскакивая от стенок.

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:id="@+id/LinearLayout1"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical" >

<Button

android:id="@+id/button\_erase"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="center\_horizontal"

android:onClick="onClick"

android:text="Очистить" />

<SurfaceView

android:id="@+id/surface"

android:layout\_width="350dp"

android:layout\_height="250dp"

android:layout\_gravity="center" />

</LinearLayout>

Код:

package ru.alexanderklimov.test;

import ...

public class TestActivity extends Activity implements View.OnTouchListener,

SurfaceHolder.Callback {

private SurfaceView mSurface;

private DrawingThread mThread;

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

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

mSurface = (SurfaceView) findViewById(R.id.surface);

mSurface.setOnTouchListener(this);

mSurface.getHolder().addCallback(this);

}

public void onClick(View v) {

mThread.clearItems();

}

public boolean onTouch(View v, MotionEvent event) {

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

mThread.addItem((int) event.getX(), (int) event.getY());

}

return true;

}

@Override

public void surfaceCreated(SurfaceHolder holder) {

mThread = new DrawingThread(holder, BitmapFactory.decodeResource(

getResources(), R.drawable.ic\_launcher));

mThread.start();

}

@Override

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

int height) {

mThread.updateSize(width, height);

}

@Override

public void surfaceDestroyed(SurfaceHolder holder) {

mThread.quit();

mThread = null;

}

private static class DrawingThread extends HandlerThread implements

Handler.Callback {

private static final int MSG\_ADD = 100;

private static final int MSG\_MOVE = 101;

private static final int MSG\_CLEAR = 102;

private int mDrawingWidth, mDrawingHeight;

private SurfaceHolder mDrawingSurface;

private Paint mPaint;

private Handler mReceiver;

private Bitmap mIcon;

private ArrayList<DrawingItem> mLocations;

private class DrawingItem {

// Current location marker

int x, y;

// Direction markers for motion

boolean horizontal, vertical;

public DrawingItem(int x, int y, boolean horizontal,

boolean vertical) {

this.x = x;

this.y = y;

this.horizontal = horizontal;

this.vertical = vertical;

}

}

public DrawingThread(SurfaceHolder holder, Bitmap icon) {

super("DrawingThread");

mDrawingSurface = holder;

mLocations = new ArrayList<DrawingItem>();

mPaint = new Paint(Paint.ANTI\_ALIAS\_FLAG);

mIcon = icon;

}

@Override

protected void onLooperPrepared() {

mReceiver = new Handler(getLooper(), this);

// Start the rendering

mReceiver.sendEmptyMessage(MSG\_MOVE);

}

@Override

public boolean quit() {

// Clear all messages before dying

mReceiver.removeCallbacksAndMessages(null);

return super.quit();

}

@Override

public boolean handleMessage(Message msg) {

switch (msg.what) {

case MSG\_ADD:

// Create a new item at the touch location,

// with a randomized start direction

DrawingItem newItem = new DrawingItem(msg.arg1, msg.arg2,

Math.round(Math.random()) == 0, Math.round(Math

.random()) == 0);

mLocations.add(newItem);

break;

case MSG\_CLEAR:

// Remove all objects

mLocations.clear();

break;

case MSG\_MOVE:

// Render a frame

Canvas c = mDrawingSurface.lockCanvas();

if (c == null) {

break;

}

// Clear Canvas first

c.drawColor(Color.BLACK);

// Draw each item

for (DrawingItem item : mLocations) {

// Update location

item.x += (item.horizontal ? 5 : -5);

if (item.x >= (mDrawingWidth - mIcon.getWidth())) {

item.horizontal = false;

} else if (item.x <= 0) {

item.horizontal = true;

}

item.y += (item.vertical ? 5 : -5);

if (item.y >= (mDrawingHeight - mIcon.getHeight())) {

item.vertical = false;

} else if (item.y <= 0) {

item.vertical = true;

}

// Draw to the Canvas

c.drawBitmap(mIcon, item.x, item.y, mPaint);

}

// Release to be rendered to the screen

mDrawingSurface.unlockCanvasAndPost(c);

break;

}

// Post the next frame

mReceiver.sendEmptyMessage(MSG\_MOVE);

return true;

}

public void updateSize(int width, int height) {

mDrawingWidth = width;

mDrawingHeight = height;

}

public void addItem(int x, int y) {

// Pass the location into the Handler using Message arguments

Message msg = Message.obtain(mReceiver, MSG\_ADD, x, y);

mReceiver.sendMessage(msg);

}

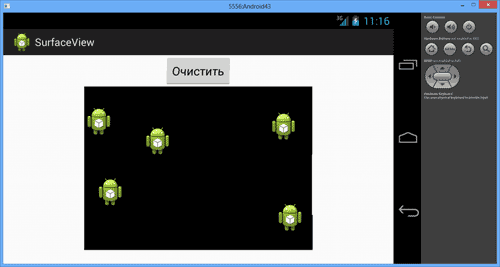
public void clearItems() {

mReceiver.sendEmptyMessage(MSG\_CLEAR);

}

}

}



**Реклама**

**Примеры**

[Класс MediaPlayer - воспроизведение видео](http://developer.alexanderklimov.ru/android/video.php#mediaplayer).

[Предварительный просмотр записываемого видео](http://developer.alexanderklimov.ru/android/videorecorder.php#previewrecord).