VizInteract

Building a large screen Interactive Data Visualization experience with Android Canvas and X86
Supratim
HCI Researcher at SIAT
@heysupratim
Wolfgang Stuerzlinger
Professor
School of Interactive arts and Technology, Vancouver
Dynspace

A VA Tool built for V4-Space. An ultra high resolution interactive display arrangement. Currently the largest research setup of its kind.

8 x 4K 80” displays synced at 60Hz

Interactive data visualization tool
For touch enabled displays which can utilize multi-touch gestures for efficient data exploration
Why android?
Flexibility to support multiple hardware configurations.

We had to support VA tasks on lower end devices.
VizInteract DEMO
What we tried
and gave up on
We required **graceful transformation** from one view type to another
In WebView
Significant decrease in performance when using scale and rotate gestures with multiple graphs on screen
Graph Surface

Canvas ───────── SurfaceView

Sense Plot ───────── SurfaceHolder

Gesture Detector ───────── DataManager

Viz Idioms
Individual drawing routines
Sharing
Learnings and Tips
Canvas **Transformations** and **Save** and **Restore** are your friends.
drawAndro()
Your new rotated canvas

Canvas save

Rotate canvas

Canvas Restore

drawAndro()

Your new rotated canvas
Save and Restore calls need to be balanced

Especially important if drawing routines are deferred to separate objects on screen
canvas.getSaveCount()

Returns #save calls minus # restore calls

canvas.restoreToCount(int count)

Pops last count save calls
Popping back save and restore stack only removes the modifications not the actual drawing
Running log of canvas operations, including transformations and addition and deletions from canvas
this.bitmap = previousDrawing.get(previousDrawing.size() - 1);
this.canvas.drawBitmap(this.bitmap, 0, 0, null);
previousDrawing.remove(previousDrawing.size() - 1);
Transformation Matrix

- Scale
- Translate
- Rotate
Map points between world and local coordinate system
transformMatrix.scale
transformMatrix.postRotate

float[] worldCoord
transformMatrix.mapPoints(worldCoord)

localCoords
transformMatrix.post Translate
transformMatrix.post Rotate
transformMatrix.post Scale
canvas.drawBitmap(bitmap, transformMatrix)

Faster

canvas.translate
canvas.rotate
canvas.scale
transformMatrix

canvas.concat(transformMatrix)

canvas = canvas * transformMatrix
Quick
Canvas Scale tip
canvas.scale

Exact Touch x,y

rect = canvas.getClipBounds()
px = touchev.getX()/scaleFactor + rect.left
py = touchev.getY()/scaleFactor + rect.top
Just a quality of life addition I did was to use a \textbf{Region} object behind the scenes to do my \texttt{overlap} detections between touch points and visualizations.

```java
region.contains(x, y)
region.quickReject(anotherRegion)
```
**Bitmap.createBitmap** (bitmap, 0, 0, bitmap.getWidth(), bitmap.getHeight(), scaleMatrix, true)

CPU Bitmap Resizing

**drawBitmap** (bitmap, scaleMatrix, paint)

GPU used to resize
Avoid creating/decoding Bitmap Resources in your draw / render function
Same goes for color resources. Do your `color.parse(colorstring)` in an `init` function.

For my Frequency histogram, I calculate the histogram bins and counts in an `init` function.
If you have multiple simple shapes to render

```java
mSelectedPath = new Path();
for(Rect rectF: listOfRects)
    mSelectedPath.addRect(rectF, Path.Direction.CW);
canvas.drawPath(mSelectedPath, paint);
```
Offloading data preprocessing on a separate thread and updating your canvas from that thread
SurfaceView implements SurfaceHolder.Callback

SurfaceHolder holder = getHolder();
holder.addCallback(this);
thread = new DataManagerThread(holder);
class DataManagerThread extends Thread {

    private SurfaceHolder mSurfaceHolder;
    public DataManagerThread(SurfaceHolder surfaceHolder) {
        mSurfaceHolder = surfaceHolder;
    }

    @Override
    public void run() {
        canvas = mSurfaceHolder.lockCanvas(null);
        // canvas drawRoutine();
        mSurfaceHolder.unlockCanvasAndPost(c);
    }
}
Quick

`SurfaceHolder` tip
Always **match** the **format** of your **SurfaceHolder** and the **bitmaps**.

If the surface is **16 bits, RGB565** Bitmaps will be very fast to load

```
surfaceHolder.setFormat (int format)
```
Also, if you think why your text doesn't look readable, make sure you set the `ANTI_ALIAS_FLAG` in your `paint` object.
Getting the app on the interactive large display
PQ Labs IR Touch Overlay
Android-x86 Project
AOSP derivative
Community submitted patches to make it run on x86 arch
3M 24” touchscreen monitor

Live USB bootable
Plug and play with App works
Samsung 80” 4K with PQ Labs touchscreen Overlay

Live USB bootable
Touch input system didn’t work
3M is a very popular touch screen frame manufacturer.

Kernel module is in Linux upstream.
I ventured into the complete darkness of Kernel Module Hacking
Linux 2.6.3 USB PQ Labs driver
written by Jemini for their original
single point IR touch frame
In your driver probe function

input_mt_init_slots(dev->input_dev, MAX_SUPPORT_POINTS);
Registering the driver in the probe function

```c
input_set_abs_params(dev->input_dev, ABS_MT_ORIENTATION, 0, 1, 0, 0);
input_set_abs_params(dev->input_dev, ABS_MT_TOUCH_MAJOR, 0, 0xFFFF, 0, 0);
input_set_abs_params(dev->input_dev, ABS_MT_TOUCH_MINOR, 0, 0xFFFF, 0, 0);
input_set_abs_params(dev->input_dev, ABS_MT_POSITION_X, 0, 0xFFFF, 0, 0);
input_set_abs_params(dev->input_dev, ABS_MT_POSITION_Y, 0, 0xFFFF, 0, 0);

input_set_abs_params(dev->input_dev, ABS_X, 0, 0xFFFF, 0, 0);
input_set_abs_params(dev->input_dev, ABS_Y, 0, 0xFFFF, 0, 0);
```
input_set_abs_params(dev->input_dev, ABS_MT_POSITION_X, 0, 0xFFFF, 0, 0);
input_set_abs_params(dev->input_dev, ABS_MT_POSITION_Y, 0, 0xFFFF, 0, 0);
The fuzz variable indicates the expected noise on the input device (in pixels). The input system handles this by filtering the input positions according to this value. Technically it does two things

**Hysteresis**
The cursor will not move until the difference in values is sufficient

**Smoothing**
Smoothens values immediately outside of the dead-zone.
Some software side improvements for improving large screen translation
Velocity tracking

velocityTracker = VelocityTracker.obtain();
velocityTracker.addMovement(event);
velocityTracker.computeCurrentVelocity(1000);
velocityTracker.getXVelocity(pointerId));
velocityTracker.recycle();
Velocity threshold for same action on a unnaturally large screen is a lot lower than a smaller form factor.
Accommodate for touch Jitter
Especially Long press touch events
Set a distance threshold = ViewConfiguration.getScaledTouchSlop*constant +
Waiting period threshold = 300ms
Accommodate for scaling while rotating

In our rotationGestureDetector, we find any perceived scaling and pass it on to the canvas. It feels more natural.
angle = angleBetweenLines(fX, fY, sX, sY, nfX, nfY, nsX, nsY);
span = distanceBetweenPoints(fX, fY, sX, sY);

if (listener != null && Math.abs(angle)>10) {
    listener.onRotation(angle);
}

if(listener!=null && Math.abs(angle)<10) && span>spanThreshold{
    listener.perceivedScaling(span);
}
All minor performance gains are important
Things like Double Buffering are amazing but !!
**Input: add PQ Labs IR touchscreen driver**

**author**  Supratim <supi70792@gmail.com>

**commit**  ea5eff121bd732afdbcbf78ae2adcd012a3fe5bf

**tree**  69407b84756798462048b3ae9842c0c35986c4d8  [tree | snapshot]

**parent**  b8f6e6f8a957a98a7246ef5ae64a359a638a2873  [commit | diff]

**Signed-off-by:** Chih-Wei Huang <cwhuang@linux.org.tw>

```bash
arch/x86/configs/android-x86_64_defconfig
arch/x86/configs/android-x86_defconfig
drivers/input/touchscreen/Kconfig
drivers/input/touchscreen/makefile
drivers/input/touchscreen/usb_pqlabs.c  [new file with mode: 0644]  [blob]
```

*kernel*
VizInteract large screen DEMO
Supratim
HCI Researcher at SIAT
@heysupratim
Thank you