

Touch UI Application Launcher S60 5th

--stenlik

1. Introduction

This article shows how to use the touch screen functionality for the semi-3D application launcher. The idea behind this example is to put together the OpenGL functionality (rotating cube with texture) with the finger touch screen trajectory detection algorithm. The picture below shows how the full example will look like.

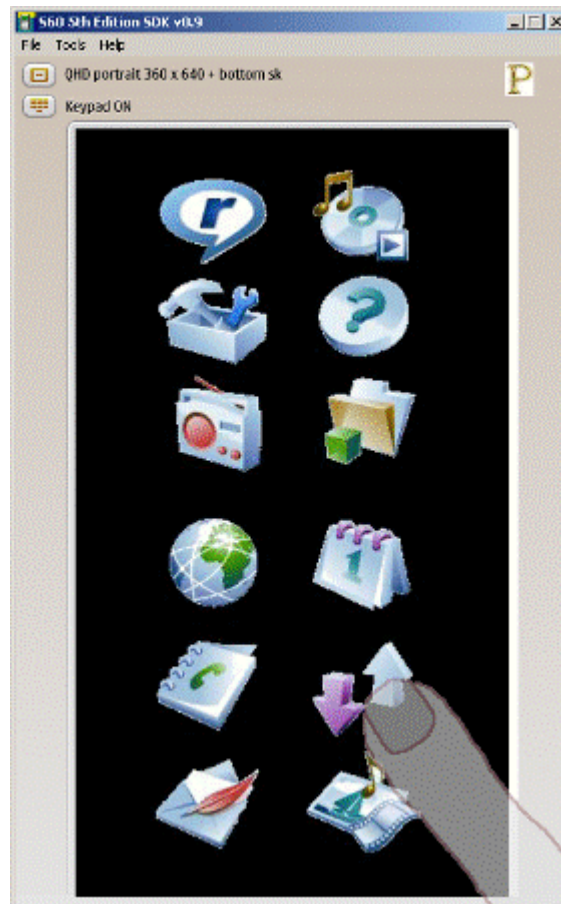


Figure 1 Animation

2. Movement detection

First I will show how to use the touch screen to rotate the cube. The issue to be solved is the detection of the stylus (or finger) movement trajectory. The only support we get from the API is the classic `HandlePointerEvent(const TPointerEvent &aPointerEvent)` method (well known from the UIQ platform), which gives us only the information about the type of the generated event and the position on the screen, where the event occurs.

There is no native support for more detailed event like e.g. “the movement of the pixel length x from the point a to point b was detected”.

In our example we will be interested in three types of pointer events (TPointerEvent::Type):

- EButton1Down: It is raised when the user touches the screen.
- EButton1Up: It is raised when the user removes the stylus.
- EDrag: It is raised during the movement of the stylus over the screen. Note that to be able to receive drag events, the EnableDragEvents() method on the observed control must be called in advance.

Following piece of code shows events we will use:

```
void HandlePointerEventL(const TPointerEvent& aPointerEvent)
{
    switch( aPointerEvent.iType )
    {
        //////////////////////////////////////
        // Stylus tap
        case TPointerEvent::EButton1Down:
        {
            ...
            break;
        };

        //////////////////////////////////////
        // Stylus removed away
        case TPointerEvent::EButton1Up:
        {
            ...
            break;
        };

        //////////////////////////////////////
        // Stylus movement
        case TPointerEvent::EDrag:
        {
            ...
            break;
        };
    }
}
```

The picture below shows the order, in which pointer events are generated. Immediately after user touches the screen the EButton1Down event is generated. During the period when the stylus slides over the screen EDrag events are generated in the defined interval (see SetMinInterDragInterval() method) and finally when the user removes the stylus or finger the EButton1Up event is generated.

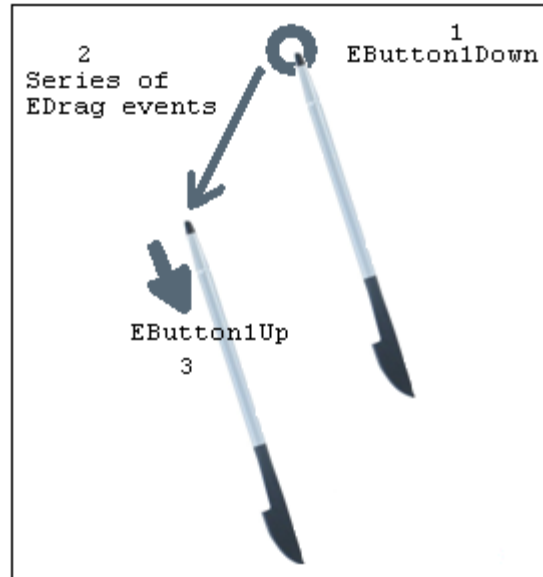


Figure 2 Pointer events

In the body of the `HandlePointerEventL()` method we are collecting all pointer event positions. When user removes the stylus I use the linear regression least square method to detect the direction and the angle of the movement trajectory. Picture below shows few recorded trajectories - red color strokes represents series of points captured in the `HandlePointerEventL()` method during sliding with stylus over the screen and with green color are drawn lines approximated by least square method.

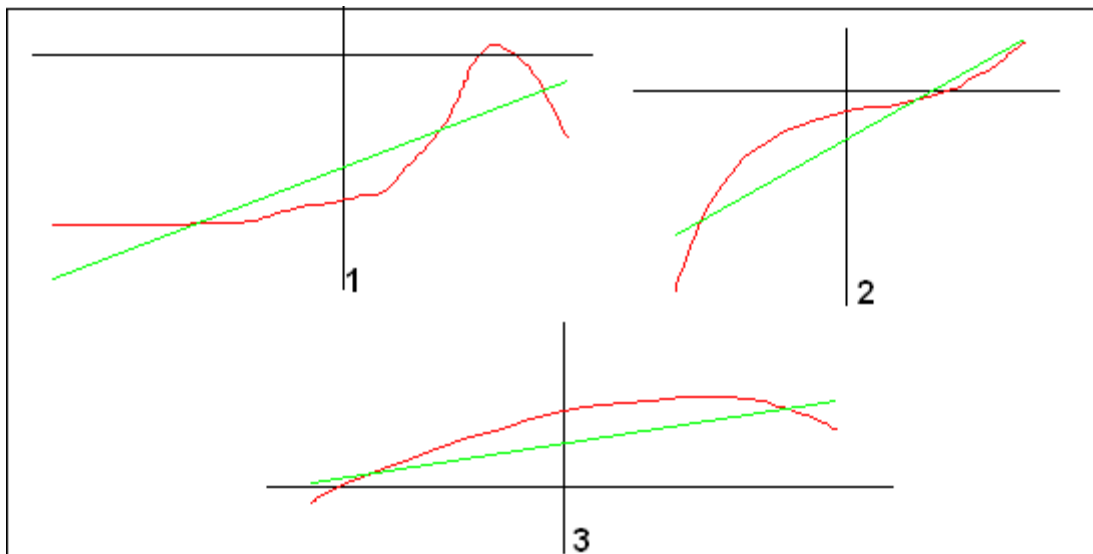


Figure 3 Points and the approximation

The following code shows how coefficients of the line formula are computed, based on the array of pointers captured in `HandlePointerEventL()` method.

```
// -----
```

```

//
// Transform coordinates from pixel-based
// coordinate to 0,0 system
//
// -----
TPoint TransformCoord( TPoint aPoint ) const
{
    return TPoint( aPoint.iX - Rect().Width() / 2, -(aPoint.iY -
Rect().Height() / 2) );
}

// -----
//
// Transform coordinates back
//
// -----
TPoint TransformCoordBack( TPoint aPoint ) const
{
    return TPoint( aPoint.iX + Rect().Width() / 2, Rect().Height() -
(aPoint.iY + Rect().Height() / 2) );
}

// -----
//
// Calculate linear regression (least square)
// coefficients
//
// -----
Void CalculateLinearRegressionCoef()
{
    // LR variables
    TReal n, sumX, sumY, sumXsqr, sumXY;
    n = sumX = sumY = sumXsqr = sumXY = 0;

    // Transform from pixel coordinates
    RArray<TPoint> points;
    for (TInt i = 0; i < iPoints.Count() - 1; i++)
        points.Append( TransformCoord(iPoints[i]) );

    // Compute LR variables
    for (TInt i = 0; i < points.Count() - 1; i++)
    {
        n++;
        sumX += points[i].iX;
        sumY += points[i].iY;
        sumXsqr += points[i].iX * points[i].iX;
        sumXY += points[i].iX * points[i].iY;
    }

    //  $y = iA + iB * x$ 
    iB = ( n * sumXY - sumY * sumX ) / ( n * sumXsqr - sumX * sumX );
    iA = ( sumY - iB * sumX ) / n;

    points.Close();
}

```

As a result of linear regression we have a and b coefficients for the straight line formula $y=b*x+a$. Having those values it is easy to get the angle of the trajectory and the direction of the movement (e.g. from bottom to left, etc.). I applied following rules for the detection of the movement direction:

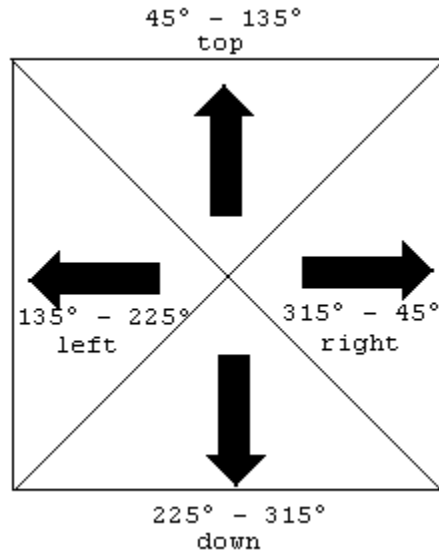


Figure 4 Angles and the direction

Following code shows how the angle is computed and how the direction is determined:

```
// -----
//
// Get angle in degrees (not radians!)
//
// -----
Tint Angle( TReal& aAngleDeg ) const
{
    if ( EMovement == iState )
    {
        TReal angle = 0;
        Math::ATan( angle, iB );
        angle *= 180/KPi;

        TInt y1 = iB*1 + iA;
        TInt y2 = iB*2 + iA;

        if ( y1 < y2 )
        {
            if ( iPoints[0].iX > iPoints[iPoints.Count() - 1].iX )
                aAngleDeg = angle + 180;
            else
                aAngleDeg = angle;
        }
        else
        {
            if ( iPoints[0].iX > iPoints[iPoints.Count() - 1].iX )
                aAngleDeg = angle + 180;
        }
    }
}
```

```

        else
            aAngleDeg = angle + 360;
        }

        return KErrNone;
    }
    else
        return KErrNotFound;
}
// -----
//
// Get the direction
//
// -----
Tint Direction( TRotationDirection& aDirection) const
{
    TReal angle;
    if ( Angle( angle ) == KErrNone )
    {
        if ( angle >= 45 & angle < 45 + 90 )
            aDirection = ERotationUp;
        else
            if ( angle >= 45 + 90 & angle < 45 + 180 )
                aDirection = ERotationLeft;
            else
                if ( angle >= 45 + 180 & angle < 45 + 270 )
                    aDirection = ERotationDown;
                else
                    aDirection = ERotationRight;

        return KErrNone;
    }
    else
        return KErrNotFound;
}

```

3. OpenGL, examples and links

I have used two OpenGL ES examples from the S60 5th SDK (SimpleCube and Texture). I have been using the OpenGL for the first time, so I have made only few “dirty” modifications in already existing examples code – just enough to present the idea. To show application icons I have used still images, instead of loading the applications icons dynamically. This is out of scope of this example – I focused on the idea of the application and the way how to analyze the movement trajectory.

To have fully working solution, one must deal with dynamically loaded textures and installed application enumeration (Enumeration of installed applications and obtaining its icons is quite easy - just check methods `GetAllApps()`, `GetAppCapability()`, `GetAppIconSizes()` and `GetAppIcon()` from class `RApaLsSession`). The last issue to be solved is application launching – for this I would suggest to detect the long tap with `CAknLongTapDetector` class.

To make application working four images from `gfx` folder must be copied to the `\S60_5th_Edition_SDK_v0.9\epoc32\winscw\c\` folder, otherwise the application will panic after startup.

In the `WikiLauncherAppView.cpp` file must be defined one of three macros `__DRAW_DEBUG__`, `__SIMPLE_CUBE__` and `__TEXTURE_CUBE__`. Depending on which macro is defined the application displays the different content related to the different stage of the example preparation.

Download the example (zip):

Download this text (pdf):

Related links:

http://en.wikipedia.org/wiki/Linear_regression

http://en.wikipedia.org/wiki/Least_squares

http://www.khronos.org/opengles/documentation/opengles1_0/html/

http://www.forum.nokia.com/info/sw.nokia.com/id/567330dd-130f-4f1d-9380-fac5aec5a6a9/S60_Platform_Image_Converter_Example.html