AppLauncher - S60 5th Touch UI

Overview

21 Dec 2008

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 touchscreen trajectory detection algorithm. The picture below shows how the full example will look like.

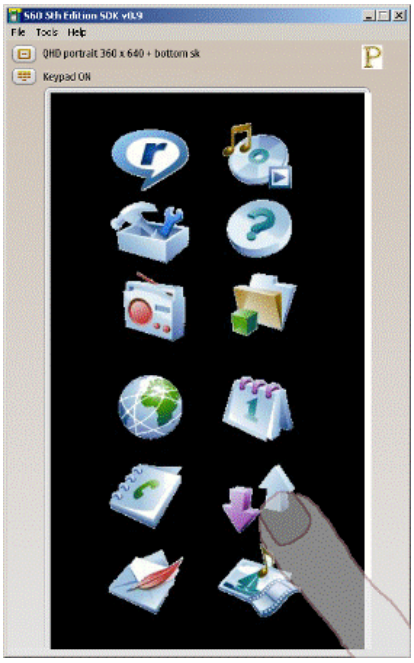

Figure 1. Animation

Movement detection

The following section explains how to use the touchscreen to rotate the cube. The issue to be solved is the detection of the stylus (or finger) movement trajectory. The only support from the API is the classic `HandlePointerEvent(const TPointerEvent &aPointerEvent)` method (well known from the UIQ platform), which gives only 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 events such as "the movement of the pixel length **x** from the point **a** to point **b** was detected".

The example focuses on three types of pointer events (`TPointerEvent::Type`):

- `EButton1Down`: Raised when the user touches the screen.
- `EButton1Up`: Raised when the user removes the stylus.
- `EDrag`: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 the events that will be used:

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

    ////////////////////////////////////
    // Stylus removwed away
    case TPointerEvent::EButton1Up:
    {
      …
      break;
    };

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


(C) Copyright Nokia 2013. All rights reserved.

```
}
```

The picture below shows the order in which pointer events are generated. Immediately after the 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.

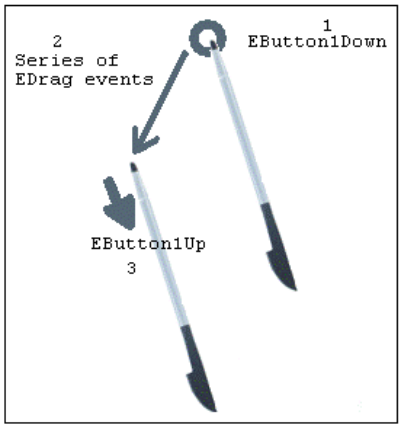

Figure 2. Pointer events

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

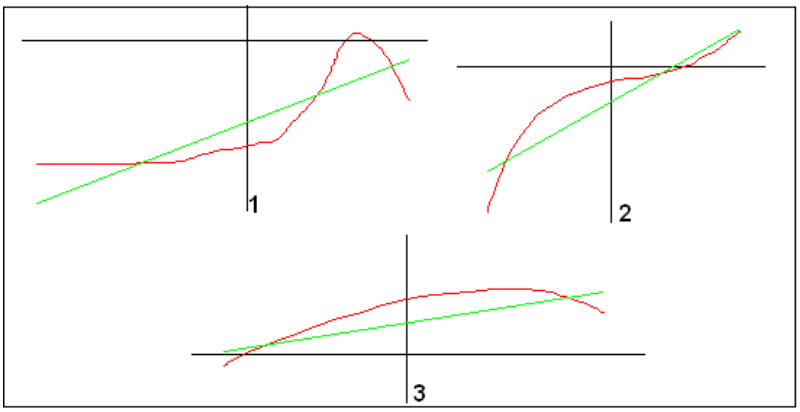

Figure 3. Points and the approximation

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

```
// ------------------------------------------------
//
// Transfrom 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) );
}

// ------------------------------------------------
//
// Transfrom 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)
// coeficients
//
// ------------------------------------------------
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++)
  {
```


(C) Copyright Nokia 2013. All rights reserved.

```
    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**. With these values it is easy to get the angle of the trajectory and the direction of the movement (for example, from bottom to left, and so on).

The following rules are applied for the detection of the movement direction:

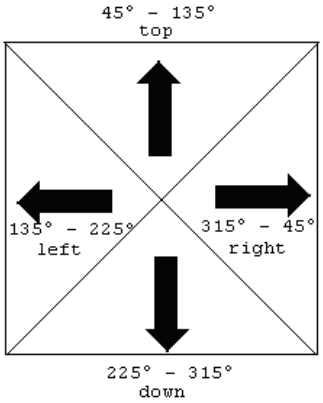

Figure 4. Angles and the direction

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;
}
```


(C) Copyright Nokia 2013. All rights reserved.

OpenGL and Others

Two OpenGL ES examples from the S60 5th SDK (**SimpleCube** and **Texture**) are used, with only few "dirty" modifications in the existing examples code – just enough to present the idea. Still images are used instead of loading the applications icons dynamically as that is out of scope of this example which focuses on the idea of the application and the way how to analyze the movement trajectory.

For a fully working solution, 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`) must also be dealt with. The last issue to be solved is application launching – for this detecting the long tap with `CAknLongTapDetector` class is suggested.

To make the application work, four images from the 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 one of the three macros `__DRAW_DEBUG__`, `__SIMPLE_CUBE__` and `__TEXTURE_CUBE__` must be defined. Depending on which macro is defined, the application displays different content related to the different stages of the example preparation.

Links and Downloads

Downloads:

Download the example (zip) TouchUI S60 5th Edition AppLauncher.zip

Download this text (pdf) TouchUI S60 5th Edition AppLauncher.pdf

Links:

Linear regression [1]

Least squares [2]

OpenGL ES documentation [3]

Platform Image Converter Example [4]

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.