Subject: Re: Using Pen with U++
Posted by Tom1 on Fri, 19 Mar 2021 15:02:57 GMT
View Forum Message <> Reply to Message

Hi Mirek,

OK, now I'm nearly there with Windows. What's missing is that I cannot get RectTracker to work with Pen. I would really appreciate, if you could take a look at that. There's a new testcase with on-screen instructions below to test the central functions of pen and mouse...

In order to keep everything else working with mouse and a pen emulating a mouse, I have introduced a new flag and function for enabling accurate pen support for Ctrls requiring fine drawing. The reason I had to do this is that I could not retain usable mouse emulation when processing WM_POINTER* messages for fine drawing. (As an example, the buttons would not work with WM_POINTER* generated LEFTDOWN/UPs.)

Anyway, here it is.

Add to "Ctrl{" in CtrlCore.h:

```
 bool   pen_supported;

 void         EnablePenSupport(bool flag=true)  { pen_supported=flag; }
```

Add to "Ctrl::Ctrl() {" in Ctrl.cpp:

```
 pen_supported = false;
```

Changes in "WindowProc()" in Win32Proc.cpp:

```
...
LRESULT Ctrl::WindowProc(UINT message, WPARAM wParam, LPARAM lParam) {
 GuiLock __;
 eventid++;
// LLOG("Ctrl::WindowProc(" << message << ") in " << ::Name(this) << ", focus " << (void
*)::GetFocus());
 Ptr<Ctrl> _this = this;
 HWND hwnd = GetHWND();

 auto DoMouseMove = [&](POINT p) {
  if(ignoreclick)
   EndIgnore();
  else {
   if(_this) DoMouse(MOUSEMOVE, Point(p));
   DoCursorShape();
  }
 };

 static bool left_down=false;
```

```
 static bool right_down=false;
 static Point pendownpos=Null;

 switch(message) {
 case WM_POINTERDOWN:
 case WM_POINTERUP:
 case WM_POINTERUPDATE:{
  if(!pen_supported) break; // Go with mouse emulation (default processing) instead

  POINT p = Point((LONG)lParam);
  ScreenToClient(hwnd, &p);

  pen = false;
  pen_pressure = pen_rotation = Null;
  pen_tilt = Null;
  pen_eraser = false;
  pen_barrel = false;
  pen_inverted = false;
  pen_history = false;

  static BOOL (WINAPI *GetPointerType)(UINT32 pointerId, POINTER_INPUT_TYPE
*pointerType);
  static BOOL (WINAPI *GetPointerInfo)(UINT32 pointerId, POINTER_INFO *pointerInfo);
  static BOOL (WINAPI *GetPointerPenInfo)(UINT32 pointerId, POINTER_PEN_INFO *penInfo);
  static BOOL (WINAPI *GetPointerPenInfoHistory)(UINT32 pointerId, UINT32 *entriesCount,
POINTER_PEN_INFO *penInfo);

  ONCELOCK {
   DllFn(GetPointerType, "User32.dll", "GetPointerType");
   DllFn(GetPointerInfo, "User32.dll", "GetPointerInfo");
   DllFn(GetPointerPenInfo, "User32.dll", "GetPointerPenInfo");
   DllFn(GetPointerPenInfoHistory, "User32.dll", "GetPointerPenInfoHistory");
  };

  if(!(GetPointerType && GetPointerInfo && GetPointerPenInfo && GetPointerPenInfoHistory))
   break;

  POINTER_INPUT_TYPE pointerType;

  auto ProcessPenInfo = [&] (POINTER_PEN_INFO& ppi) {
   pen = true;
   if(ppi.penFlags & PEN_FLAG_BARREL)
    pen_barrel = true;
   if(ppi.penFlags & PEN_FLAG_INVERTED)
    pen_inverted = true;
   if(ppi.penFlags & PEN_FLAG_ERASER)
    pen_eraser = true;
   if(ppi.penMask & PEN_MASK_PRESSURE)
```

```
      pen_pressure = ppi.pressure / 1024.0;
     if(ppi.penMask & PEN_MASK_ROTATION)
      pen_rotation = ppi.rotation * M_2PI / 360;
     if(ppi.penMask & PEN_MASK_TILT_X)
      pen_tilt.x = ppi.tiltX * M_2PI / 360;
     if(ppi.penMask & PEN_MASK_TILT_Y)
      pen_tilt.y = ppi.tiltY * M_2PI / 360;
    };

    UINT32 pointerId = GET_POINTERID_WPARAM(wParam);
    if(GetPointerType(pointerId, &pointerType) && pointerType == PT_PEN) {
     UINT32 hc = 256;
     Buffer<POINTER_PEN_INFO> ppit(hc);
     if(message == WM_POINTERUPDATE && (right_down || left_down) &&
GetPointerPenInfoHistory(pointerId, &hc, ppit)) {
      for(int i = hc - 1; i >= 0; i--) {
       ProcessPenInfo(ppit[i]);
       POINT hp = ppit[i].pointerInfo.ptPixelLocation;
       ScreenToClient(hwnd, &hp);
       pen_history = (bool)i;
       DoMouseMove(hp);
      }
      pen_history = false;
     }
     else{
      POINTER_PEN_INFO ppi;
      if(GetPointerPenInfo(pointerId, &ppi)){
       ProcessPenInfo(ppi);
       switch(message) {
        case WM_POINTERDOWN:
         ClickActivateWnd();
         if(ignoreclick) return 0L;
         right_down=pen_barrel;
         left_down=!right_down;
         if(right_down) DoMouse(RIGHTDOWN, pendownpos=Point(p));
         else DoMouse(LEFTDOWN, pendownpos=Point(p), 0);
         if(_this) PostInput();
         break;
       }
      }
     }
    }
   }
   break;
  case WM_POINTERLEAVE:
   pen = false;
   right_down=false;
   left_down=false;
```

```
  break;

...

 case WM_LBUTTONDOWN:
  ClickActivateWnd();
  if(ignoreclick) return 0L;
  if(pendownpos==Point((dword)lParam)) { pendownpos=Null; return 0L; }
  else{
   DoMouse(LEFTDOWN, Point((dword)lParam));
   if(_this) PostInput();
  }
  return 0L;
 case WM_LBUTTONUP:
  if(ignoreclick)
   EndIgnore();
  else{
   if(right_down) DoMouse(RIGHTUP, Point((dword)lParam));
   else DoMouse(LEFTUP, Point((dword)lParam));
  }
  if(_this) PostInput();
  right_down=false;
  left_down=false;
  pendownpos=Null;
  return 0L;
 case WM_LBUTTONDBLCLK:
  ClickActivateWnd();
  if(ignoreclick) return 0L;
  DoMouse(LEFTDOUBLE, Point((dword)lParam));
  if(_this) PostInput();
  return 0L;
 case WM_RBUTTONDOWN:
  if(pendownpos==Point((dword)lParam)) { pendownpos=Null; return 0L; }
  ClickActivateWnd();
  if(ignoreclick) return 0L;
  DoMouse(RIGHTDOWN, Point((dword)lParam));
  if(_this) PostInput();
  return 0L;
 case WM_RBUTTONUP:
  if(ignoreclick)
   EndIgnore();
  else
   DoMouse(RIGHTUP, Point((dword)lParam));
  if(_this) PostInput();
  right_down=false;
  left_down=false;
  pendownpos=Null;
  return 0L;
```

```
...

 case WM_MOUSEMOVE:
  if(left_down || right_down) return 0L;
  LLOG("WM_MOUSEMOVE: ignoreclick = " << ignoreclick);
  DoMouseMove(Point((dword)lParam));
  return 0L;

...
```

And finally the extended testcase:

```
#include <CtrlLib/CtrlLib.h>

using namespace Upp;

#define LLOG(x)  DLOG(x)

struct MyApp : TopWindow {
 Point pos;

 Point lDouble;
 Point rDouble;
 Point lHold;
 Point rHold;

 bool lDoublePen;
 bool rDoublePen;
 bool lHoldPen;
 bool rHoldPen;

 Vector<Vector<Tuple<double, Pointf>>> drawing;
 Vector<Vector<Tuple<double, Pointf>>> rdrawing;

 bool rdown;
 bool ldown;

 Point p1,p2; // Tracker test variables
 Rect trect;

 MyApp(){
  rdown=false;
  ldown=false;
  lDouble=Null;
```

```
 rDouble=Null;
 lHold=Null;
 rHold=Null;

 lDoublePen=false;
 rDoublePen=false;
 lHoldPen=false;
 rHoldPen=false;

 trect=Null;
 p1=p2=Null;

 EnablePenSupport();
}

virtual void RightHold(Point p, dword){
 LLOG((IsPointerPen()?"PenRightHold":"MouseRightHold"));
 rHold=p;
 rHoldPen=IsPointerPen();
 Refresh();
}

virtual void RightDouble(Point p, dword){
 LLOG((IsPointerPen()?"PenRightDouble":"MouseRightDouble"));
 rDouble=p;
 rDoublePen=IsPointerPen();
 Refresh();
}

virtual void RightDown(Point p, dword){
 LLOG((IsPointerPen()?"PenRightDown":"MouseRightDown"));
 rdown=true;
 rdrawing.Add().Add(MakeTuple(IsPointerPen()?GetPenPressure():0.1, p));
 Refresh();
}

virtual void RightUp(Point p, dword){
 LLOG((IsPointerPen()?"PenRightUp":"MouseRightUp"));
 rdown=false;
 Refresh();
}

virtual void LeftHold(Point p, dword){
 LLOG((IsPointerPen()?"PenLeftHold":"MouseLeftHold"));
 lHold=p;
 lHoldPen=IsPointerPen();
 Refresh();
}
```

```
 virtual void LeftDouble(Point p, dword){
  LLOG((IsPointerPen()?"PenLeftDouble":"MouseLeftDouble"));
  lDouble=p;
  lDoublePen=IsPointerPen();
  Refresh();
 }

 virtual void LeftDown(Point p, dword keyflags){
  if(keyflags&K_SHIFT){
   RectTracker tracker(*this);
   tracker.sync= [=](Rect r) {  };
   tracker.MinSize(Size(-100000,-100000));
   trect=tracker.Track(Rect(p,p));
  }
  else if(keyflags&K_CTRL){
   RectTracker tracker(*this);
   p1=p;
   p2=tracker.TrackLine(p.x,p.y);
  }
  else{
   LLOG((IsPointerPen()?"PenLeftDown":"MouseLeftDown"));
   ldown=true;
   drawing.Add().Add(MakeTuple(IsPointerPen()?GetPenPressure():0.1, p));
  }
  Refresh();
 }

 virtual void LeftUp(Point p, dword){
  LLOG((IsPointerPen()?"PenLeftUp":"MouseLeftUp"));
  ldown=false;
  Refresh();
 }

 Vector<String> report;

 virtual void MouseMove(Point p, dword keyflags) {
  pos = p;

  LLOG((IsPointerPen()?"PenMove":"MouseMove"));

  if((ldown && drawing.GetCount()) || (rdown && rdrawing.GetCount())) {
   if(rdown){
    if(!IsPenHistoryEvent()) rdrawing.Top().Add(MakeTuple(IsPointerPen()?GetPenPressure():0.1,
p));
   }
   else if(ldown) drawing.Top().Add(MakeTuple(IsPointerPen()?GetPenPressure():0.1, p));
  }
```

```
  report.Clear();
  report.Add() << AsString(pos);
  report.Add() << "Pen: " << IsPointerPen();
  report.Add() << "Pressure: " << GetPenPressure();
  report.Add() << "Rotation: " << GetPenRotation();
  report.Add() << "Tilt: " << GetPenTilt();
  report.Add() << "Barrel: " << IsPenBarrelPressed();
  report.Add() << "Inverted: " << IsPenInverted();
  report.Add() << "Eraser: " << IsPenEraserPressed();

  Refresh();
 }

 virtual void Paint(Draw& w0){
  DrawPainter w(w0, GetSize());
  w.Clear(SColorPaper());

  w.LineCap(LINECAP_ROUND);

  for(const auto& stroke : drawing)
   if(stroke.GetCount())
    for(int i = 0; i < stroke.GetCount() - 1; i++) {
     w.Move(stroke[i].b);
     w.Line(stroke[i + 1].b);
     w.Stroke(DPI(20) * stroke[i].a, LtBlue());
    }

  for(const auto& stroke : rdrawing)
   if(stroke.GetCount())
    for(int i = 0; i < stroke.GetCount() - 1; i++) {
     w.Move(stroke[i].b);
     w.Line(stroke[i + 1].b);
     w.Stroke(DPI(20) * stroke[i].a, Black());
    }

  if(!IsNull(trect)) w.RectPath(trect).Stroke(2.0,Red());
  if(!IsNull(p2)) w.Move(p1).Line(p2).Stroke(2.0,Green());

  int fcy = GetStdFontCy();
  int y = 10;
  auto Text = [&] (const String& text) {
   w.DrawText(10, y, text);
   y += fcy;
  };

   Text("1. Test Clicks, Holds, Drags and DoubleClicks using Mouse with both Left and Right
buttons.");
```

```
  Text("2. Test Clicks, Drags and DoubleClicks using Pen without and with Barrel button.");
  Text("3. Test RectTracker with Ctrl+LeftDrag and Shift+LeftDrag using Mouse and then Pen.");

  for(int i=0;i<report.GetCount();i++) Text(report[i]);

  if(!IsNull(lHold)) w.DrawText(lHold.x, lHold.y, "LeftHold", StdFont(), lHoldPen?LtBlue():Black());
  if(!IsNull(rHold)) w.DrawText(rHold.x, rHold.y, "RightHold", StdFont(),
rHoldPen?LtBlue():Black());
  if(!IsNull(lDouble)) w.DrawText(lDouble.x, lDouble.y, "LeftDouble", StdFont(),
lDoublePen?LtBlue():Black());
  if(!IsNull(rDouble)) w.DrawText(rDouble.x, rDouble.y, "RightDouble", StdFont(),
rDoublePen?LtBlue():Black());
 }
};

GUI_APP_MAIN
{
 PromptOK("Please tap OK with Pen to verify button operation!");
 MyApp().Run();
}
```

Best regards,

Tom