How do I get mouse messages faster than WM_MOUSEMOVE?

devblogs.microsoft.com/oldnewthing/20120314-00

March 14, 2012

Raymond Chen

We saw some time ago that the rate at which you receive `WM_MOUSEMOVE` messages is entirely up to how fast your program calls `GetMessage`. But what if your program is calling `GetMessage` as fast as it can, and it's still not fast enough?

You can use the `GetMouseMovePointsEx` function to ask the window manager, "Hey, can you tell me about the mouse messages I missed?" I can think of two cases where you might want to do this:

- You are a program like Paint, where the user is drawing with the mouse and you want to capture every nuance of the mouse motion.
- You are a program that supports something like mouse gestures, so you want the full mouse curve information so you can do your gesture recognition on it.

Here's a program that I wrote for a relative of mine who is a radiologist. One part of his job consists of sitting in a dark room studying medical images. He has to use his years of medical training to identify the tumor (if there is one), and then determine what percentage of the organ is afflicted. To use this program, run it and position the circle so that it matches the location and size of the organ under study. Once you have the circle positioned properly, use the mouse to draw an outline of the tumor. When you let go of the mouse, the title bar will tell you the size of the tumor relative to the entire organ.

(Oh great, now I'm telling people to practice medicine without a license.)

First, we'll do a version of the program that just calls `GetMessage` as fast as it can. Start with the new scratch program and make the following changes:

```
class RootWindow : public Window
{
public:
 virtual LPCTSTR ClassName() { return TEXT("Scratch"); }
 static RootWindow *Create();
protected:
 LRESULT HandleMessage(UINT uMsg, WPARAM wParam, LPARAM lParam);
 void PaintContent(PAINTSTRUCT *pps);
 BOOL WinRegisterClass(WNDCLASS *pwc);
private:
 RootWindow();
 ~RootWindow();
 void OnCreate();
 void UpdateTitle();
 void OnSizeChanged(int cx, int cy);
 void AlwaysAddPoint(POINT pt);
 void AddPoint(POINT pt);
 void OnMouseMove(LPARAM lParam);
 void OnButtonDown(LPARAM lParam);
 void OnButtonUp(LPARAM lParam);
 // arbitrary limit (this is just a demo!)
 static const int cptMax = 1000;
private:
 POINT  m_ptCenter;
 int    m_radius;
 BOOL   m_fDrawing;
 HPEN   m_hpenInside;
 HPEN   m_hpenDot;
 POINT  m_ptLast;
 int    m_cpt;
 POINT  m_rgpt[cptMax];
};
RootWindow::RootWindow()
 : m_fDrawing(FALSE)
 , m_hpenInside(CreatePen(PS_INSIDEFRAME, 3,
                                  GetSysColor(COLOR_WINDOWTEXT)))
 , m_hpenDot(CreatePen(PS_DOT, 1, GetSysColor(COLOR_WINDOWTEXT)))
{
}
RootWindow::~RootWindow()
{
 if (m_hpenInside) DeleteObject(m_hpenInside);
 if (m_hpenDot) DeleteObject(m_hpenDot);
}
BOOL RootWindow::WinRegisterClass(WNDCLASS *pwc)
{
 pwc->style |= CS_VREDRAW | CS_HREDRAW;
 return __super::WinRegisterClass(pwc);
}
void RootWindow::OnCreate()
{
 SetLayeredWindowAttributes(m_hwnd, 0, 0xA0, LWA_ALPHA);
```

```
}
void RootWindow::UpdateTitle()
{
 TCHAR szBuf[256];
 // Compute the area of the circle using a surprisingly good
 // rational approximation to pi.
 int circleArea = m_radius * m_radius * 355 / 113;
 // Compute the area of the region, if we have one
 if (m_cpt > 0 && !m_fDrawing) {
  int polyArea = 0;
  for (int i = 1; i < m_cpt; i++) {
   polyArea += m_rgpt[i-1].x * m_rgpt[i  ].y -
               m_rgpt[i  ].x * m_rgpt[i-1].y;
  }
  if (polyArea < 0) polyArea = -polyArea; // ignore orientation
  polyArea /= 2;
  wnsprintf(szBuf, 256,
            TEXT("circle area is %d, poly area is %d = %d%%"),
            circleArea, polyArea,
            MulDiv(polyArea, 100, circleArea));
 } else {
  wnsprintf(szBuf, 256, TEXT("circle area is %d"), circleArea);
 }
 SetWindowText(m_hwnd, szBuf);
}
void RootWindow::OnSizeChanged(int cx, int cy)
{
 m_ptCenter.x = cx / 2;
 m_ptCenter.y = cy / 2;
 m_radius = min(m_ptCenter.x, m_ptCenter.y) - 6;
 if (m_radius < 0) m_radius = 0;
 UpdateTitle();
}
void RootWindow::PaintContent(PAINTSTRUCT *pps)
{
 HBRUSH hbrPrev = SelectBrush(pps->hdc,
                              GetStockBrush(HOLLOW_BRUSH));
 HPEN hpenPrev = SelectPen(pps->hdc, m_hpenInside);
 Ellipse(pps->hdc, m_ptCenter.x - m_radius,
                   m_ptCenter.y - m_radius,
                   m_ptCenter.x + m_radius,
                   m_ptCenter.y + m_radius);
 SelectPen(pps->hdc, m_hpenDot);
 Polyline(pps->hdc, m_rgpt, m_cpt);
 SelectPen(pps->hdc, hpenPrev);
 SelectBrush(pps->hdc, hbrPrev);
}
void RootWindow::AddPoint(POINT pt)
{
 // Ignore duplicates
 if (pt.x == m_ptLast.x && pt.y == m_ptLast.y) return;
 // Stop if no room for more
```

```
 if (m_cpt >= cptMax) return;
 AlwaysAddPoint(pt);
}
void RootWindow::AlwaysAddPoint(POINT pt)
{
 // Overwrite the last point if we can't add a new one
 if (m_cpt >= cptMax) m_cpt = cptMax - 1;
 // Invalidate the rectangle connecting this point
 // to the last point
 RECT rc = { pt.x, pt.y, pt.x+1, pt.y+1 };
 if (m_cpt > 0) {
  RECT rcLast = { m_ptLast.x,   m_ptLast.y,
                  m_ptLast.x+1, m_ptLast.y+1 };
  UnionRect(&rc, &rc, &rcLast);
 }
 InvalidateRect(m_hwnd, &rc, FALSE);
 // Add the point
 m_rgpt[m_cpt++] = pt;
 m_ptLast = pt;
}
void RootWindow::OnMouseMove(LPARAM lParam)
{
 if (m_fDrawing) {
  POINT pt = { GET_X_LPARAM(lParam), GET_Y_LPARAM(lParam) };
  AddPoint(pt);
 }
}
void RootWindow::OnButtonDown(LPARAM lParam)
{
 // Erase any previous polygon
 InvalidateRect(m_hwnd, NULL, TRUE);
 m_cpt = 0;
 POINT pt = { GET_X_LPARAM(lParam), GET_Y_LPARAM(lParam) };
 AlwaysAddPoint(pt);
 m_fDrawing = TRUE;
}
void RootWindow::OnButtonUp(LPARAM lParam)
{
 if (!m_fDrawing) return;
 OnMouseMove(lParam);
 // Close the loop, eating the last point if necessary
 AlwaysAddPoint(m_rgpt[0]);
 m_fDrawing = FALSE;
 UpdateTitle();
}
LRESULT RootWindow::HandleMessage(
                          UINT uMsg, WPARAM wParam, LPARAM lParam)
{
 switch (uMsg) {
  case WM_CREATE:
   OnCreate();
   break;
```

```
  case WM_NCDESTROY:
   // Death of the root window ends the thread
   PostQuitMessage(0);
   break;
  case WM_SIZE:
   if (wParam == SIZE_MAXIMIZED || wParam == SIZE_RESTORED) {
    OnSizeChanged(GET_X_LPARAM(lParam), GET_Y_LPARAM(lParam));
   }
   break;
  case WM_MOUSEMOVE:
   OnMouseMove(lParam);
   break;
  case WM_LBUTTONDOWN:
   OnButtonDown(lParam);
   break;
  case WM_LBUTTONUP:
   OnButtonUp(lParam);
   break;
 }
 return __super::HandleMessage(uMsg, wParam, lParam);
}
RootWindow *RootWindow::Create()
{
 RootWindow *self = new(std::nothrow) RootWindow();
 if (self && self->WinCreateWindow(WS_EX_LAYERED,
       TEXT("Scratch"), WS_OVERLAPPEDWINDOW,
       CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
       NULL, NULL)) {
      return self;
  }
 delete self;
 return NULL;
}
```

This program records every mouse movement while the button is down and replays them in the form of a dotted polygon. When the mouse button goes up, it calculates the area both in terms of pixels and in terms of a percentage of the circle.

This program works well. My relative's hand moves slowly enough (after all, it has to trace a tumor) that the `GetMessage` loop is plenty fast enough to keep up. But just for the sake of illustration, suppose it isn't. To make the effect easier to see, let's add some artificial delays:

```
void RootWindow::OnMouseMove(LPARAM lParam)
{
 if (m_fDrawing) {
  POINT pt = { GET_X_LPARAM(lParam), GET_Y_LPARAM(lParam) };
  AddPoint(pt);
  UpdateWindow(m_hwnd);
  Sleep(100);
 }
}
```

Now, if you try to draw with the mouse, you see all sorts of jagged edges because our program can't keep up. (The `UpdateWindow` is just to make the most recent line visible while we are sleeping.)

Enter `GetMouseMovePointsEx`. This gives you all the mouse activity that led up to a specific point in time, allowing you to fill in the data that you missed because you weren't pumping messages fast enough. Let's teach our program how to take advantage of this:

```
class RootWindow : public Window
{
...
 void AlwaysAddPoint(POINT pt);
 void AddMissingPoints(POINT pt, DWORD tm);
 void AddPoint(POINT pt);
...
 POINT m_ptLast;
 DWORD m_tmLast;
 int   m_cpt;
};
void RootWindow::AddMissingPoints(POINT pt, DWORD tm)
{
 // See discussion for why this code is wrong
 ClientToScreen(m_hwnd, &pt);
 MOUSEMOVEPOINT mmpt = { pt.x, pt.y, tm };
 MOUSEMOVEPOINT rgmmpt[64];
 int cmmpt = GetMouseMovePointsEx(sizeof(mmpt), &mmpt,
                            rgmmpt, 64, GMMP_USE_DISPLAY_POINTS);
 POINT ptLastScreen = m_ptLast;
 ClientToScreen(m_hwnd, &ptLastScreen);
 int i;
 for (i = 0; i < cmmpt; i++) {
  if (rgmmpt[i].time < m_tmLast) break;
  if (rgmmpt[i].time == m_tmLast &&
      rgmmpt[i].x == ptLastScreen.x &&
      rgmmpt[i].y == ptLastScreen.y) break;
 }
 while (--i >= 0) {
   POINT ptClient = { rgmmpt[i].x, rgmmpt[i].y };
   ScreenToClient(m_hwnd, &ptClient);
   AddPoint(ptClient);
 }
}
void RootWindow::AlwaysAddPoint(POINT pt)
{
...
 // Add the point
 m_rgpt[m_cpt++] = pt;
 m_ptLast = pt;
 m_tmLast = GetMessageTime();
}
void RootWindow::OnMouseMove(LPARAM lParam)
{
 if (m_fDrawing) {
  POINT pt = { GET_X_LPARAM(lParam), GET_Y_LPARAM(lParam) };
  AddMissingPoints(pt, GetMessageTime());
  AddPoint(pt);
  UpdateWindow(m_hwnd);
  Sleep(100); // artificial delay to simulate unresponsive app
 }
}
```

Before updating the the current mouse position, we check to see if there were other mouse motions that occurred while we weren't paying attention. We tell `GetMouseMovePointsEx`, "Hey, here is a mouse message that I have right now. Please tell me about the stuff that I missed." It fills in an array with recent mouse history, most recent events first. We go through that array looking for the previous point, and give up either when we find it, or when the timestamps on the events we received take us too far backward in time. Once we find all the points that we missed, we play them into the `AddPoint` function.

Notes to people who like to copy code without understanding it: The code fragment above works only for single-monitor systems. To work correctly on multiple-monitor systems, you need to include the crazy coordinate-shifting code provided in the documentation for `GetMouseMovePointsEx`. (I omitted that code because it would just be distracting.) Also, the management of `m_tmLast` is now rather confusing, but I did it this way to minimize the amount of change to the original program. It would probably be better to have added a `DWORD tm` parameter to `AddPoint` instead of trying to infer it from the current message time.

The `GetMouseMovePointsEx` technique is also handy if you need to refer back to the historical record. For example, if the user dragged the mouse out of your window and you want to calculate the velocity with which the mouse exited, you can use `GetMouseMove-PointsEx` to get the most recent mouse activity and calculate the velocity. This saves you from having to record all the mouse activity yourself on the off chance that the mouse might leave the window.

Raymond Chen

Follow