Taxes: Remote Desktop Connection and painting

devblogs.microsoft.com/oldnewthing/20060103-12

January 3, 2006

Raymond Chen

An increasingly important developer tax is supporting Remote Desktop Connection properly. When the user is connected via a Remote Desktop Connection, video operations are transferred over the network connection to the client for display. Since networks have high latency and nowhere near the bandwidth of a local PCI or AGP bus, you need to adapt to the changing cost of drawing to the screen.

If you draw a line on the screen, the “draw line” command is sent over the network to the client. If you draw text, a “draw text” command is sent (along with the text to draw). So far so good. But if you copy a bitmap to the screen, the entire bitmap needs to be transferred over the network.

Let’s write a sample program that illustrates this point. Start with our new scratch program and make the following changes:

```
void Window::Register()
{
    WNDCLASS wc;
    wc.style         = CS_VREDRAW | CS_HREDRAW;
    wc.lpfnWndProc   = Window::s_WndProc;
    ...
}
class RootWindow : public Window
{
public:
 virtual LPCTSTR ClassName() { return TEXT("Scratch"); }
 static RootWindow *Create();
protected:
 LRESULT HandleMessage(UINT uMsg, WPARAM wParam, LPARAM lParam);
 LRESULT OnCreate();
 void PaintContent(PAINTSTRUCT *pps);
 void Draw(HDC hdc, PAINTSTRUCT *pps);
private:
 HWND m_hwndChild;
};
void RootWindow::Draw(HDC hdc, PAINTSTRUCT *pps)
{
 FillRect(hdc, &pps->rcPaint, (HBRUSH)(COLOR_WINDOW + 1));
 RECT rc;
 GetClientRect(m_hwnd, &rc);
 for (int i = -10; i < 10; i++) {
  TextOut(hdc, 0, i * 15 + rc.bottom / 2, TEXT("Blah blah"), 9);
 }
}
void RootWindow::PaintContent(PAINTSTRUCT *pps)
{
 Draw(pps->hdc, pps);
}
LRESULT RootWindow::HandleMessage(
                          UINT uMsg, WPARAM wParam, LPARAM lParam)
{
 switch (uMsg) {
 ...
 case WM_ERASEBKGND: return 1;
 ...
}
```

There is an odd division of labor here; the `PaintContent` method doesn't actually do anything aside from handing the work off to the `Draw` method to do the actual drawing. (You'll see why soon.) Make sure "Show window contents while dragging" is enabled and run this program and resize it vertically. Ugh, what ugly flicker. We fix this by the traditional technique of double-buffering.

```
void RootWindow::PaintContent(PAINTSTRUCT *pps)
{
 if (!IsRectEmpty(&pps->rcPaint)) {
  HDC hdc = CreateCompatibleDC(pps->hdc);
  if (hdc) {
   int x = pps->rcPaint.left;
   int y = pps->rcPaint.top;
   int cx = pps->rcPaint.right - pps->rcPaint.left;
   int cy = pps->rcPaint.bottom - pps->rcPaint.top;
   HBITMAP hbm = CreateCompatibleBitmap(pps->hdc, cx, cy);
   if (hbm) {
    HBITMAP hbmPrev = SelectBitmap(hdc, hbm);
    SetWindowOrgEx(hdc, x, y, NULL);
    Draw(hdc, pps);
    BitBlt(pps->hdc, x, y, cx, cy, hdc, x, y, SRCCOPY);
    SelectObject(hdc, hbmPrev);
    DeleteObject(hbm);
   }
   DeleteDC(hdc);
  }
 }
}
```

Our new `PaintContent` method creates an offscreen bitmap and asks the `Draw` method to draw into it. Once that's done, the results are copied to the screen at one go, thereby avoiding flicker. If you run this program, you'll see that it resizes nice and smooth.

Now connect to the computer via a Remote Desktop Connection and run it again. Since Remote Desktop Connection disables "Show window contents while dragging", you can't use resizing to trigger redraws, so instead maximize the program and restore it a few times. Notice the long delay before the window is resized when you maximize it. That's because we are pumping a huge bitmap across the Remote Desktop Connection as part of that `BitBlt` call.

Go back to the old version of the `PaintContent` method, the one that just calls `Draw`, and run it over Remote Desktop Connection. Ah, this one is fast. That's because the simpler version doesn't transfer a huge bitmap over the Remote Desktop Connection; it just sends twenty `TextOut` calls on a pretty short string of text. These take up much less bandwidth than a 1024×768 bitmap.

We have one method that is faster over a Remote Desktop Connection, and another method that is faster when run locally. Which should we use?

We use both, choosing our drawing method based on whether the program is running over a Remote Desktop Connection.

```
void RootWindow::PaintContent(PAINTSTRUCT *pps)
{
 if (GetSystemMetrics(SM_REMOTESESSION)) {
  Draw(pps->hdc, pps);
 } else if (!IsRectEmpty(&pps->rcPaint)) {
  ... as before ...
 }
}
```

Now we get the best of both worlds. When run locally, we use the double-buffered drawing which draws without flickering, but when run over a Remote Desktop Connection, we use the simple `Draw` method that draws directly to the screen rather than to an offscreen bitmap.

This is a rather simple example of adapting to Remote Desktop Connection. In a more complex world, you may have more complicated data structures associated with the two styles of drawing, or you may have background activities related to drawing that you may want to turn on and off based on whether the program is running over a Remote Desktop Connection. Since the user can dynamically connect and disconnect, you can't just assume that the state of the Remote Desktop Connection when your program starts will be the state for the lifetime of the program. We'll see next time how we can adapt to a changing world.

Raymond Chen

Follow