BeginBufferedPaint: It's not just for buffered painting any more

devblogs.microsoft.com/oldnewthing/20110520-00

May 20, 2011

Raymond Chen

I covered the `BeginBufferedPaint` function in my 2008 PDC presentation, but one thing I didn't mention is that the buffered paint functions are very handy even if you have no intention of painting.

Since the buffered paint functions maintain a cache (provided that you remembed to call `BufferedPaintInit`), you can use `BeginBufferedPaint` to get a temporary bitmap even if you have no intention of actually painting to the screen. You might want a bitmap to do some off-screen composition, or for some other temporary purpose, in which case you can ask `BeginBufferedPaint` to give you a bitmap, use the bitmap for whatever you like, and then pass `fUpdateTarget = FALSE` when you call `EndBufferedPaint` to say "Ha ha, just kidding."

One thing to have to be aware of is that the bitmap provided by `BeginBufferedPaint` is not guaranteed to be exactly the size you requested; it only promises that the bitmap will be *at least* the size you requested. Most of the time, your code won't care (there are just pixels out there that you aren't using), but if you use the `GetBufferedPaintBits` function to obtain direct access to the bits, don't forget to take the stride into account.

Consider this artifical example of a program that uses `CreateDIBSection` to create a temporary 32bpp bitmap for the purpose of updating a layered window. Start with the scratch program and make these changes:

```
BOOL
OnCreate(HWND hwnd, LPCREATESTRUCT lpcs)
{
 BOOL fRc = FALSE;
 HDC hdcWin = GetDC(hwnd);
 if (hdcWin) {
  HDC hdcMem = CreateCompatibleDC(hdcWin);
  if (hdcMem) {
   const int cx = 200;
   const int cy = 200;
   RECT rc = { 0, 0, cx, cy };
   BITMAPINFO bmi = { 0 };
   bmi.bmiHeader.biSize = sizeof(bmi.bmiHeader);
   bmi.bmiHeader.biWidth = cx;
   bmi.bmiHeader.biHeight = cy;
   bmi.bmiHeader.biPlanes = 1;
   bmi.bmiHeader.biBitCount = 32;
   bmi.bmiHeader.biCompression = BI_RGB;
   RGBQUAD *prgbBits;
   HBITMAP hbm = CreateDIBSection(hdcWin, &bmi,
             DIB_RGB_COLORS, &reinterpret_cast<void*&>(prgbBits),
                                                    NULL, 0);
   if (hbm) {
    HBITMAP hbmPrev = SelectBitmap(hdcMem, hbm);
    // Draw a simple picture
    FillRect(hdcMem, &rc,
                     reinterpret_cast<HBRUSH>(COLOR_INFOBK + 1));
    rc.left = cx / 4;
    rc.right -= rc.left;
    rc.top = cy / 4;
    rc.bottom -= rc.top;
    FillRect(hdcMem, &rc,
                  reinterpret_cast<HBRUSH>(COLOR_INFOTEXT + 1));
    // Apply the alpha channel (and premultiply)
    for (int y = 0; y < cy; y++) {
     for (int x = 0; x < cx; x++) {
      RGBQUAD *prgb = &prgbBits[y * cx + x];
      BYTE bAlpha = static_cast<BYTE>(cx * x / cx);
      prgb->rgbRed = static_cast<BYTE>(prgb->rgbRed * bAlpha / 255);
      prgb->rgbBlue = static_cast<BYTE>(prgb->rgbBlue * bAlpha / 255);
      prgb->rgbGreen = static_cast<BYTE>(prgb->rgbGreen * bAlpha / 255);
      prgb->rgbReserved = bAlpha;
     }
    }
    // update the layered window
    POINT ptZero = { 0, 0 };
    SIZE siz = { cx, cy };
    BLENDFUNCTION bf =  { AC_SRC_OVER, 0, 255, AC_SRC_ALPHA };
    fRc = UpdateLayeredWindow(hwnd, NULL, &ptZero, &siz, hdcMem,
                              &ptZero, 0, &bf, ULW_ALPHA);
    SelectBitmap(hdcMem, hbmPrev);
    DeleteObject(hbm);
```

```
   }
   DeleteDC(hdcMem);
  }
  ReleaseDC(hwnd, hdcWin);
 }
 return fRc;
}
```

Pretty standard stuff. But let's convert this to use the buffered paint functions to take advantage of the buffered paint bitmap cache.

```
BOOL
OnCreate(HWND hwnd, LPCREATESTRUCT lpcs)
{
 BOOL fRc = FALSE;
 HDC hdcWin = GetDC(hwnd);
 if (hdcWin) {
  HDC hdcMem;
  // HDC hdcMem = CreateCompatibleDC(hdcWin);
  // if (hdcMem) {
   const int cx = 200;
   const int cy = 200;
   RECT rc = { 0, 0, cx, cy };
   // BITMAPINFO bmi = { 0 };
   // bmi.bmiHeader.biSize = sizeof(bmi.bmiHeader);
   // bmi.bmiHeader.biWidth = cx;
   // bmi.bmiHeader.biHeight = cy;
   // bmi.bmiHeader.biPlanes = 1;
   // bmi.bmiHeader.biBitCount = 32;
   // bmi.bmiHeader.biCompression = BI_RGB;
   RGBQUAD *prgbBits;
   BP_PAINTPARAMS params = { sizeof(params), BPPF_NOCLIP };
   HPAINTBUFFER hbp = BeginBufferedPaint(hdcWin, &rc,
                              BPBF_TOPDOWNDIB, &params, &hdcMem);
   if (hbp) {
    int cxRow;
    if (SUCCEEDED(GetBufferedPaintBits(hpb, &prgbBits, &cxRow))) {
   // HBITMAP hbm = CreateDIBSection(hdcWin, &bmi,
   //         DIB_RGB_COLORS, &reinterpret_cast<void*&>(prgbBits),
   //                                                  NULL, 0);
   // if (hbm) {
    // HBITMAP hbmPrev = SelectBitmap(hdcMem, hbm);
    // Draw a simple picture
    FillRect(hdcMem, &rc,
                     reinterpret_cast<HBRUSH>(COLOR_INFOBK + 1));
    rc.left = cx / 4;
    rc.right -= rc.left;
    rc.top = cy / 4;
    rc.bottom -= rc.top;
    FillRect(hdcMem, &rc,
                   reinterpret_cast<HBRUSH>(COLOR_INFOTEXT + 1));
    // Apply the alpha channel (and premultiply)
    for (int y = 0; y < cy; y++) {
     for (int x = 0; x < cx; x++) {
      RGBQUAD *prgb = &prgbBits[y * cxRow + x];
      BYTE bAlpha = static_cast<BYTE>(cx * x / cx);
      prgb->rgbRed = static_cast<BYTE>(prgb->rgbRed * bAlpha / 255);
      prgb->rgbBlue = static_cast<BYTE>(prgb->rgbBlue * bAlpha / 255);
      prgb->rgbGreen = static_cast<BYTE>(prgb->rgbGreen * bAlpha / 255);
      prgb->rgbReserved = bAlpha;
     }
    }
    // update the layered window
```

```
    POINT ptZero = { 0, 0 };
    SIZE siz = { cx, cy };
    BLENDFUNCTION bf =  { AC_SRC_OVER, 0, 255, AC_SRC_ALPHA };
    fRc = UpdateLayeredWindow(hwnd, NULL, &ptZero, &siz, hdcMem,
                              &ptZero, 0, &bf, ULW_ALPHA);
    // SelectBitmap(hdcMem, hbmPrev);
    // DeleteObject(hbm);
   }
   EndBufferedPaint(hpb, FALSE);
   // DeleteDC(hdcMem);
  }
  ReleaseDC(hwnd, hdcWin);
 }
 return fRc;
}
// changes to WinMain
 if (SUCCEEDED(BufferedPaintInit())) {
 // if (SUCCEEDED(CoInitialize(NULL))) {/* In case we use COM */
  hwnd = CreateWindowEx(WS_EX_LAYERED,
  // hwnd = CreateWindow(
  ...
  BufferedPaintUnInit();
  // CoUninitialize();
  ...
```

We're using the buffered paint API not for buffered painting but just as a convenient way to get a bitmap and a DC at one shot. It saves some typing (you don't have to create the bitmap and the DC and select the bitmap in and out), and when you return the paint buffer to the cache, some other window that calls `BeginBufferedPaint` may be able to re-use that bitmap.

There are a few tricky parts here. First, if you're going to be accessing the bits directly, you need to call `GetBufferedPaintBits` and use the `cxRow` to determine the bitmap stride. Next, when we're done, we pass `FALSE` to `EndBufferedPaint` to say, "Yeah, um, thanks for the bitmap, but don't `BitBlt` the results back into the DC we passed to `BeginBufferedPaint`. Sorry for the confusion."

A less obvious trick is that we used `BPPF_NOCLIP` to get a full bitmap. By default, `BeginBufferedPaint` returns you a bitmap which is clipped to the DC you pass as the first parameter. This is an optimization to avoid allocating memory for pixels that can't be seen anyway when `EndBufferedPaint` goes to copy the bits back to the original DC. We don't want this optimization, however, since we have no intention of blitting the results back to the original DC. The clip region of the original DC is irrelevant to us because we just want a temporary bitmap for some internal calculations.

Anyway, there you have it, an example of using `BeginBufferedPaint` to obtain a temporary bitmap. It doesn't win much in this example (since we call it only once, at window creation time), but if you have code which creates a lot of DIB sections for temporary use, you

can use this trick to take advantage of the buffered paint cache and reduce the overhead of bitmap creation and deletion.

Pre-emptive snarky comment: "How dare you show us an alternative method that isn't available on Windows 2000!"

Raymond Chen

Follow