Drawing a monochrome bitmap with transparency

devblogs.microsoft.com/oldnewthing/20050803-16

August 3, 2005

Raymond Chen

Last time, I left you with a brief puzzle. Here are two approaches. I am not a GDI expert, so there may be even better solutions out there. To emphasize the transparency, I'll change the window background color to the application workspace color.

```
BOOL WinRegisterClass(WNDCLASS *pwc)
{
 pwc->hbrBackground = (HBRUSH)(COLOR_APPWORKSPACE + 1);
 return __super::WinRegisterClass(pwc);
}
```

Method 1: A big `MaskBlt`.

```
void RootWindow::PaintContent(PAINTSTRUCT *pps)
{
 HDC hdcMem = CreateCompatibleDC(pps->hdc);
 if (hdcMem) {
  int cxCheck = GetSystemMetrics(SM_CXMENUCHECK);
  int cyCheck = GetSystemMetrics(SM_CYMENUCHECK);
  HBITMAP hbmMono = CreateBitmap(cxCheck, cyCheck, 1, 1, NULL);
  if (hbmMono) {
   HBITMAP hbmPrev = SelectBitmap(hdcMem, hbmMono);
   if (hbmPrev) {
    RECT rc = { 0, 0, cxCheck, cyCheck };
    DrawFrameControl(hdcMem, &rc, DFC_MENU, DFCS_MENUCHECK);
    COLORREF clrTextPrev = SetTextColor(pps->hdc,
                                        GetSysColor(COLOR_MENUTEXT));
    // COLORREF clrBkPrev = SetBkColor(pps->hdc,
    //                                 GetSysColor(COLOR_MENU));
    MaskBlt(pps->hdc, 0, 0, cxCheck, cyCheck,
           hdcMem, 0, 0, hbmMono, 0, 0
           MAKEROP4(0x00AA0029, SRCCOPY));
    // SetBkColor(pps->hdc, clrBkPrev);
    SetTextColor(pps->hdc, clrTextPrev);
    SelectBitmap(hdcMem, hbmPrev);
   }
   DeleteObject(hbmMono);
  }
  DeleteDC(hdcMem);
 }
}
```

This has the least amount of typing but feels like overkill to me, using a quaternary raster operation as if were a ternary, just because I didn't want to create a pattern brush. (The raster operation 0x00AA0029 is the NOP operator; it leaves the destination alone. I didn't have this memorized; I looked it up in the documentation.) The `MAKEROP4` says that for each white pixel in the mask, do nothing (NOP), and for each black pixel, do a `SRCCOPY`.

Notice that the background color is never used (since it's supposed to be transparent); consequently, we can delete the code that sets and restores the DC's background color.

Method 2: The traditional two-step.

```
void RootWindow::PaintContent(PAINTSTRUCT *pps)
{
 HDC hdcMem = CreateCompatibleDC(pps->hdc);
 if (hdcMem) {
  int cxCheck = GetSystemMetrics(SM_CXMENUCHECK);
  int cyCheck = GetSystemMetrics(SM_CYMENUCHECK);
  HBITMAP hbmMono = CreateBitmap(cxCheck, cyCheck, 1, 1, NULL);
  if (hbmMono) {
   HBITMAP hbmPrev = SelectBitmap(hdcMem, hbmMono);
   if (hbmPrev) {
    RECT rc = { 0, 0, cxCheck, cyCheck };
    DrawFrameControl(hdcMem, &rc, DFC_MENU, DFCS_MENUCHECK);
    COLORREF clrTextPrev = SetTextColor(pps->hdc, RGB(0,0,0));
    COLORREF clrBkPrev = SetBkColor(pps->hdc, RGB(255,255,255));
    BitBlt(pps->hdc, cxCheck, 0, cxCheck, cyCheck,
           hdcMem, 0, 0, SRCAND);
    SetTextColor(pps->hdc, GetSysColor(COLOR_MENUTEXT));
    SetBkColor(pps->hdc, RGB(0,0,0));
    BitBlt(pps->hdc, cxCheck, 0, cxCheck, cyCheck,
           hdcMem, 0, 0, SRCPAINT);
    SetBkColor(pps->hdc, clrBkPrev);
    SetTextColor(pps->hdc, clrTextPrev);
    SelectBitmap(hdcMem, hbmPrev);
   }
   DeleteObject(hbmMono);
  }
  DeleteDC(hdcMem);
 }
}
```

This is the traditional two-step blit. The first erases the pixels that are about to be overwritten by setting the foreground to black and background to white, then using `SRCAND`. This has the effect of erasing all the foreground pixels to zero while leaving the background intact. The second blit does the same, but with `SRCPAINT`. This means that the background pixels need to be treated as black, so that when they are “or”d with the destination, the destination pixels are unchanged. The foreground pixels get the desired foreground color.

This method can be shortened by negating the first blit, reversing the sense of foreground and background, so that the color black doesn’t have to move between the background color and the text color.

```
void RootWindow::PaintContent(PAINTSTRUCT *pps)
{
 HDC hdcMem = CreateCompatibleDC(pps->hdc);
 if (hdcMem) {
  int cxCheck = GetSystemMetrics(SM_CXMENUCHECK);
  int cyCheck = GetSystemMetrics(SM_CYMENUCHECK);
  HBITMAP hbmMono = CreateBitmap(cxCheck, cyCheck, 1, 1, NULL);
  if (hbmMono) {
   HBITMAP hbmPrev = SelectBitmap(hdcMem, hbmMono);
   if (hbmPrev) {
    RECT rc = { 0, 0, cxCheck, cyCheck };
    DrawFrameControl(hdcMem, &rc, DFC_MENU, DFCS_MENUCHECK);
    COLORREF clrTextPrev = SetTextColor(pps->hdc, RGB(255,255,255));
    COLORREF clrBkPrev = SetBkColor(pps->hdc, RGB(0,0,0));
    BitBlt(pps->hdc, cxCheck, 0, cxCheck, cyCheck,
           hdcMem, 0, 0, 0x00220326); // DSna
    SetTextColor(pps->hdc, GetSysColor(COLOR_MENUTEXT));
    BitBlt(pps->hdc, cxCheck, 0, cxCheck, cyCheck,
           hdcMem, 0, 0, SRCPAINT);
    SetBkColor(pps->hdc, clrBkPrev);
    SetTextColor(pps->hdc, clrTextPrev);
    SelectBitmap(hdcMem, hbmPrev);
   }
   DeleteObject(hbmMono);
  }
  DeleteDC(hdcMem);
 }
}
```

Whether this shortening is actually an overall improvement is difficult to tell. It's possible that some display drivers have a highly optimized `SRCAND` handler whereas they are less likely to have an optimized `0x00220326` handler.

(Exercise: Why can't you instead reverse the second blit, converting it to a `MERGEPAINT`?)

Raymond Chen

Follow