What was that story about the WinHelp pen-writing-in-book animation?

devblogs.microsoft.com/oldnewthing/20100817-00

August 17, 2010

Raymond Chen

The first time you open a WinHelp file, you get this pen-writing-in-book animation while WinHelp does um *something* which it passes off as *preparing Help file for first use* or something similarly vague.

I remember a conversation about that animation. The Windows shell team suggested to the author of WinHelp that the program use the shell common animation control to display that animation. After all, it's a short animation and it met the requirements for the animation common control. But the author of WinHelp rejected the notion.

(Pre-emptive "I can't believe I had to write this": This conversation has been exaggerated for effect.)

"Your animation control is so huge and bloated. I can do it much smaller and faster myself. The RLE animation format generates frames by re-rendering the pixels that have changed, which means that at each frame of the animation, a new pen image would be recorded in the AVI file. The pen cycles through three different orientations at each location, there are ten locations on each row, and there are four rows. If I used an RLE animation, that'd be $3 \times 10 \times 4 = 120$ copies of the pen bitmap. Instead, I have just three pen bitmaps, and I manually draw them at the appropriate location for each frame. Something like this:

```
// NOTE: For simplicity, I'm ignoring the "turn the page" animation
void DrawFrame(int frame)
{
  // Calculate our position in the animation
  int penframe = frame % 3; // 3 pen images per location
  int column = (frame / 3) % 10; // 10 columns per row
  int row = (frame / 30) % 4; // 4 rows
  int i;
  POINT pt;
  DrawBlankPage(0, 0); // start with a blank sheet of paper
  // Draw the "text" that the pen "wrote" in earlier rows
  for (i = 0; i < row; i++) {
    DrawTextScribble(i, 0, 9);
  }
  // Draw the partially-completed row that the pen is on now
  DrawTextScribble(row, 0, column);
  // Position the pen image so the pen tip hits the "draw" point
  GetTextScribblePoint(column, row, &pt);
  DrawPenBitmap(penBitmaps[penframe], pt.x - 1, pt.y - 5);
}
```

“See? In just a few lines of code, I have a complete animation. All I needed was the three pen images and a background bitmap showing a book opened to a blank page. This is way more efficient both in terms of memory and execution time than your stupid animation common control. You shell guys could learn a thing or two about programming.”

“Okay, fine, don’t need to get all defensive about it. We were just making a suggestion, that’s all.”

Time passes, and Windows 95 is sent off for translation into the however many languages it is localized for. A message comes in from some of the localization teams. It seems that some locales need to change the animation. For example, the Arabic version of Windows needs the pen to write on the left-hand pages, the pen motion should be right to left, and the pages need to flip from left to right. Perhaps the Japanese translators are okay with the pen motion, but they want the pages to flip from left to right.

The localization team contacted the WinHelp author. “We’re trying to change the animation, but we can’t find the AVI file in the resources. Can you advise us on how we should localize the animation?”

Unfortunately, the WinHelp author had to tell the localization team that the direction of pen motion, and the locations of the ink marks are hard-coded into the program. Since the product had already passed code lockdown, there was nothing that could be done. WinHelp shipped with a pen that moved in the wrong direction in some locales.

Moral of the story: There’s more to software development than programming for performance. Today we learned about localizability.