Blitting and Buffering

Overview

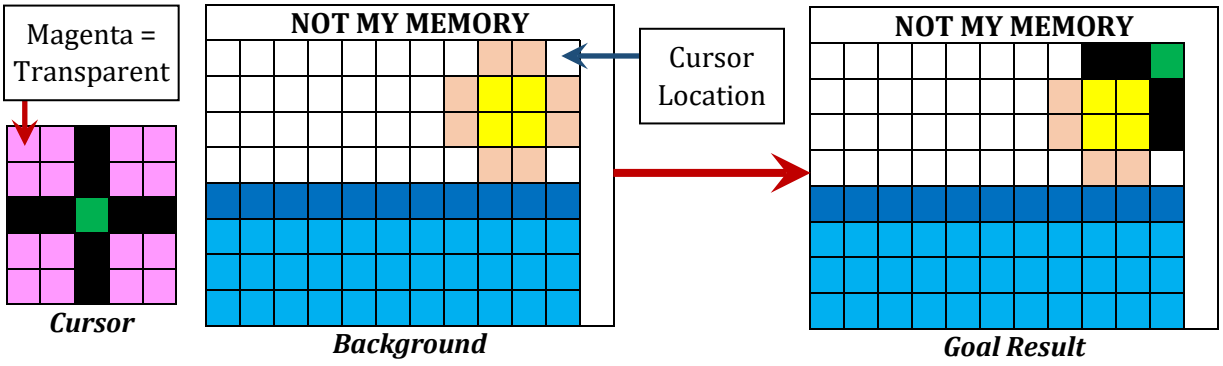
Efficiently handling memory and performant buffering presents a unique challenge to developers. Here, we outline the concept of **blitting** between memory regions along with two techniques commonly used with multidimensional I/O buffering – namely, the **double-buffer** and **dirty rectangles** approaches.

Blitting

A ***blit*** operation is a rapid copy from one section of memory to another. One-dimensional blits are usually straightforward – the program simply copies bits using an incrementing index – but multi-dimensional blits are more complex. Copying regions of one matrix to another or one image to another are common examples of multi-dimensional blitting. For example, consider a platformer game. The program must first draw the background. Then it must draw foreground terrain on top of the background, then AI characters, and finally, the player character. All of these steps require a separate multi-dimensional blitting operation. Blitting generally requires four major steps:

1) Check and adjust bounds of operation, if necessary (clipping)
2) Identify the correct ranges, and their offsets, within the source memory area
3) Identify the correct ranges, and their offsets, within the destination memory area
4) Copy the source ranges into the destination ranges

Suppose a program needs to draw a sprite on the screen representing a mouse cursor. It will need to identify which sections of the cursor to copy, as well as the offset of those sections on the destination:



When drawing, not only must we calculate the offsets, but we must also make sure we do not draw transparent pixels or draw outside of the destination memory – that is, we must engage in **clipping**. We clip the cursor image and only draw the low-left portion on the screen.

Double-Buffering

When working with devices, we often keep a local copy of the device memory in working memory (RAM) what we edit. We only copying the memory image over as needed to update the device. This approach is referred to as ***double-buffering***. In double-buffering, there are two buffers – the ***front buffer*** (device) and the ***back buffer*** (main memory). Note that the stack size is limited in real mode, so large buffers (such as those representing video memory) require that the back buffer is allocated as dynamic memory.

Dirty Rectangles

It may be tempting when dealing with complex multi-dimensional buffers to rewrite the entire buffer for each update. Once you've written the basic display routine, it is much simpler to write code to re-copy the entire background image, then all foreground images, then the spites onto the video back buffer in a game. However, this comes at a significant performance cost – that's a lot of memory to copy and replace! Instead, large multi-dimensional buffers are usually updated by identifying "dirty areas" that need to be updated and only updating those spaces – i.e., identifying **dirty rectangles**.