EECS205: Fundamentals of Computer Systems Software
Assignment 3 – Bit Blitting
Due: Tuesday, February 16, 2016 11:59PM

Overview

This quarter you will implement a single player action video game in IA32 assembly language. The programming assignments will require you to build small models of the game including graphics, game play, and sound which will all be put together by the end of the quarter. This assignment is the next step in your quarter long video game project in IA32 assembly language. For this assignment you will implement blitting routines. Biltters are usually performance critical loops that draw graphics for game-play including sprites for on-screen characters and background images. Because there is such an emphasis on speed, they are often written in assembly language. So, this assignment is not contrived at all. You will write the BasicBlit routine which draws foreground sprites. This assignment will test your ability to implement conditional statements and work with arrays and pointers. You will also write the RotateBlit routine which is capable of drawing a rotated bitmap.

In addition to drawing sprites, a good game engine has to be able to detect collisions of objects on the screen. You will therefore write CheckIntersectRect, a routine which checks to see if two rectangles overlap on the screen.

This assignment can be rather challenging. You may not be able to finish if you wait until the day before the assignment is due.

Bitmaps

We will define a new structure (EECE205BITMAP) which describes the bitmaps that we are trying to draw on the screen. Take a look at the blit.inc file to see how this structure is defined. The bitmap is essentially a group of pixel values that describe how an image should appear on the screen. The important fields of this structure are dwWidth, dwHeight, bTransparent and lpBytes. They hold the width of the bitmap, its height, color for transparency, and correspond to the start of the bitmap's color values, respectively. Since the color depth for this assignment is 8-bits, the pixel data for a bitmap is of size dwWidth*dwHeight bytes. This is essentially a one-dimensional array of color values. The first value of this array is the first pixel of the bitmap. In general, the pixel corresponding to (x,y) would be index ((y * dwWidth) + x). There is one special bitmap color (specified by bTransparent) which indicates where the bitmap should be transparent. This allows you to have bitmaps which appear in the foreground (since transparent portions will not obscure the background). Note that this is a byte quantity, not a DWORD. You will write procedures that draw bitmaps (w/ transparency) onto the screen at specified coordinates. You should essentially copy the bitmap onto the screen (via ScreenBitsPtr as you did for the last assignment), skipping over pixels that match the transparency color. In the process of drawing, your procedure might find that some or all of the bitmap won't fit into the screen (e.g. some portion of the bitmap doesn't fit into the 640x480 screen). In that case, you should clip the bitmap, drawing the portions that really fit into the screen and not attempt to draw regions which don't fit on the screen. We will provide startup code and library routines to support your module. Basically, all you need to do is write the following procedures (our library code will call them). You can use your star routine source code (star.asm) from the last assignment to create a nice background. You don't have to do anything extra with this code.

You will write two bitmap routines: BasicBlit which draws a simple bitmap to the screen and RotateBlit which draws a bitmap to the screen rotated by a specified angle.

`BasicBlit PROTO STDCALL ptrBitmap:PTR EECS205BITMAP, xcenter:DWORD, ycenter:DWORD`
ptrBitmap holds the address of a EECS205BITMAP. You will need to draw it so that the center of the bitmap appears at (xcenter, ycenter).

`RotateBlit PROTO STDCALL ptrBitmap:PTR EECS205BITMAP, xcenter:DWORD, ycenter:DWORD, angle:FXPT`
ptrBitmap holds the address of a EECS205BITMAP. You will need to draw it so that the center of the bitmap appears at (xcenter, ycenter) and in addition, the bitmap should be rotated by the given radian fixed point angle. You can use your trig functions from Assignment 2 and the following pseudocode. This is not the most optimized code...you can actually rewrite it so that there are no multiplication operations in the inner loop. If you feel like a challenge, you can rewrite it so that it just uses addition...contact us if you are interested in this option.

```
cosa = FixedCos(angle)
sina = FixedSin(angle)

esi = lpBitmap

shiftX = (EECS205BITMAP PTR [esi]).dwWidth * cosa / 2   -  (EECS205BITMAP
PTR [esi]).dwHeight * sina / 2

 shiftY = (EECS205BITMAP PTR [esi]).dwHeight * cosa / 2 +  (EECS205BITMAP
PTR [esi]).dwWidth * sina / 2

dstWidth= (EECS205BITMAP PTR [esi]).dwWidth +  (EECS205BITMAP PTR
[esi]).dwHeight;  dstHeight= dstWidth

for(-dstX = dstWidth; dstX < dstWidth; dstX++)
      for(-dstY = dstHeight; dstY < dstHeight; dstY++)
            srcX = dstX*cosa + dstY*sina
            srcY = dstY*cosa – dstX*sina
            if (srcX >= 0 && srcX < (EECS205BITMAP PTR [esi]).dwWidth &&
                srcY >= 0 && srcY < (EECS205BITMAP PTR [esi]).dwHeight &&
               (xcenter+dstX-shiftX) >= 0 && (xcenter+dstX-shiftX) < 639 &&
               (ycenter+dstY-shiftY) >= 0 && (ycenter+dstY-shiftY) < 479 &&
               bitmap pixel (srcX,srcY) is not transparent) then
                        Copy color value from bitmap (srcX, srcY)  to screen
                        (xcenter+dstX-shiftX, ycenter+dstY-shiftY)
```

Collisions
For the last part of the assignment, you will write a routine that performs some simple collision detection. This is obviously useful in a video game. Collisions occur all the time, for better or for worse. We will use a new structure to help us manage collisions: EECS205RECT which represents a rectangle on the screen as a collection of four DWORDs (dwLeft, dwTop, dwRight,

dwBottom) which correspond to the left, top, right, and bottom edges of the rectangle. By way of example, the top left corner of the rectangle would be given by (dwLeft, dwTop) and the bottom right corner would be (dwRight, dwBottom). Look at the structure definition in blit.inc to see how this is defined. Given this structure you will implement the following routine:

```
CheckIntersectRect PROTO STDCALL one:PTR EECS205RECT, two:PTR EECS205RECT
```

Which takes two pointers to rectangles (one and two). Given those two rectangles, this routine should return zero if there is intersection of these rectangles (e.g. they do not overlap) and a non-zero value in the case that they do in fact have a non-zero intersection (overlap). Assume that top <= bottom and left <= right. Think about what sorts of comparisons you will need to make to produce the right results.

Getting Started

We will assume that you already have MASM32 downloaded and installed. It may be useful to have the assistance of a debugger for this assignment. Please contact us if you need help setting one up. Next download and unpack the assignment files from the courseweb page.

Then copy your stars.asm and lines.asm files from the previous assignments into your Assignment 3 directory. You should only need to modify blit.asm. Fill in your own code to implement the two blit routines and the collision detection. You will essentially be filling in function bodies. The library code (provided) will call those functions. You really don't have to do anything else. You may find it useful to declare helper variables to store intermediate calculations or useful values. Also, it is a good strategy to try to complete this assignment in phases (e.g. first be able to draw a solid color rectangle, second directly copy bits to the screen, third add transparency, fourth add support for clipping). You can use the same build process to compile and execute this assignment.

Logistics

This assignment is due Tuesday February 16 at 11:59 PM. You should hand in your assignment via Canvas. Specifically, you will need to provide the assembly source file blit.asm. Do not hand in the executable file (blit.exe). To receive full credit, your source file must be fully commented and must compile correctly. Your executable should not crash (e.g. due to some pointer weirdness) or cause images to wrap around the screen.