Northwestern University
EECS 205 Winter 2016
Assignment 2: Lines

Due: Tuesday February 2, 2016 at 11:59pm (submit via Canvas)

For the next two assignments you will build some of the nuts and bolts of your game. Specifically, you will implement some core graphics and math functions. In this assignment, you will create a line drawing routine and implement a few trigonometric functions. The two major components of this assignment are not directly connected. So, if you are having trouble with the first half, feel free to skip ahead to the second.

One additional note. From now you should do the following in any procedures that you write. For any register that your code uses (except EAX), you should include that register in a USES list at the start of the procedure. This preserves the values used by caller so that when your function returns, the caller sees all the values in the registers that it did before the call (e.g. callee save register convention). You may want to go back add a USES list to your procedures from Assignment 1.

Part One: Trig Functions

The first two functions that you will write: FixedCos and FixedSin should take single parameters as inputs. These input parameters will be radian angles in fixed point. They should return (in EAX) the fixed point value corresponding to the cosine and sine of the corresponding radian values. For example, if you used the following code sequence:

```
xor eax, eax
INVOKE FixedCos, eax
mov cosx, eax

xor eax, eax
INVOKE FixedSin, eax
mov sinx, eax
```

The variables cosx and sinx should contain the cosine and sine of the radian angle 0.

To make your life a little easier, we have supplied a table of sine values. This table holds the values of the sin (x) for x in the range of [0,Pi/2) in increments of Pi/256. This table is held in memory as an array of WORDs (16 bit elements). The word values correspond to the fractional portion of the sine's fixed point value. For example, the first entry in the table holds the value 0 (since sin (0) = 0). The second value in the table (index 1) holds, the fractional component of sin (Pi / 256). In general, index i of the table holds sin (i * Pi / 256). The table is named SINTAB and

holds 128 values total. For this assignment whenever you encounter an angle that does not fit evenly into one of the SINTAB entries, you can choose either of its neighbors. For example, If you had to find the find the sine of the radian angle 1 = sin(1.0), you would probably quickly figure out that 1 did not have its own table entry. If it did have an entry, it would be between index 81 (whose angle is 0.99401955055) and index 82 (whose entry is 1.00629139685).

```
FixedSin PROTO angle:FXPT
FixedCos PROTO angle:FXPT
```

So you could choose either of these angles as an acceptable result. Note: There are better ways to compute this. See challenge at end of this assignment. So how do we compute the sine for x > Pi/2? Glad you asked. We can use trig identities to compute the sine (and cosine of any angle).

```
sin (x + 2 Pi) = sin (x)
sin (x + Pi) = - sin(x)
sin (x) = sin (Pi – x) (for Pi/2 < x < Pi)
cos (x) = sin (x + Pi/2)
```

So, you can implement the FixedSin function using the first three identities and then implement the FixedCos function by calling FixedSin. This should be relatively easy once you figure out what conditions you need and what calculations you need to perform on the angles. Take a look at the template files. We have provided some useful constants (like Pi/2, Pi, 2 Pi, etc.) in the template file. Feel free to define local variables or helper functions if needed...actually, we think this can make your life a whole lot easier.

You may find it useful to declare local variables to store intermediate calculations or useful values. You may also implement helper functions as needed. Also, it is a good strategy to try to complete this part in phases:

1. Get the sine function working with angles in the range of [0, Pi/2)
2. Improve on your sine function so that it works in the range of [0, Pi)
3. Improve on your sine function so that it works in the range of [0, 2 Pi)
4. Complete your sine function (any angle) and implement cosine

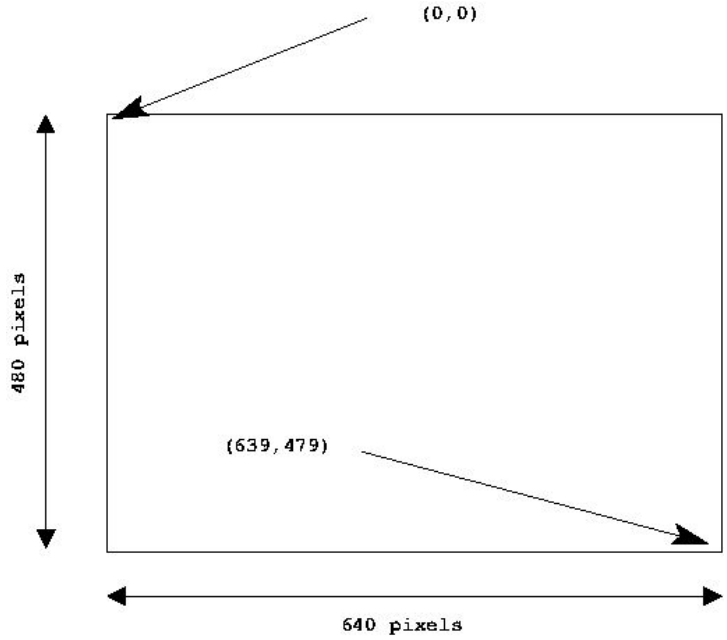
Optional: Note that because we are using lookup tables, we won't the most accurate answer for cases where the angle falls somewhere in between our multiples of Pi / 256 because the lookup process will truncate the answer down to the a multiple of Pi/256. There are a few ways to get better answers. If you feel up for a challenge implement a way to do this. You can for instance, use interpolation to get a better result.

Part Two: Line Drawing

For the second part of the assignment, you will implement a line drawing routine. This routine will be able to draw any line (vertical, horizontal, diagonal) within the game window (640x480).

To implement this procedure you will need to make good use of fixed point math, if-then-else conditionals, and loops. In addition, you will draw pixels that comprise the line by performing pointer manipulations. We will start by first describing how to draw on the screen and then we'll talk about the line drawing algorithm.

For this assignment, you will be responsible for drawing pixels in a backbuffer, a DirectX surface which serves a "workingspace" for your program. The library code that we provide will then copy this working space onto the screen using fast copy operations. This technique is fairly standard in video game design. By doing all the hard work off-screen, we can produce smooth, flicker free animation. You can think of a backbuffer as a large array which contains color values. These color values correspond to pixels on the screen. By changing the value of the colors, you can determine what will be displayed on the screen. For this assignment, we will be working in a 640x480 8-bit (width,height,color depth) screen mode. That means that each color value is a byte. You could say that ScreenWidth = 640 and ScreenHeight = 480. The coordinate system that we will be using is pretty standard for computer graphics, but is a bit different from what you may use to in math classes. The top left corner of the screen has coordinates (0,0). The bottom right corner of the screen has coordinates (639,479). Under no circumstances do you want to try to draw a pixel off-screen (e.g. outside those boundaries) – bad things might happen if you do. Your code must include checks to make sure that you do not draw outside of the bounds...even if you are given bad/illegal values to your line drawing routine!



The key to accessing the backbuffer the the global variable ScreenBitsPtr. This a pointer which holds the address of the first pixel on the game screen (at coordinate (0,0)). To draw a pixel, you set the corresponding memory location in the backbuffer to a value which represents the color you want. You should think of the entire backbuffer being an array of bytes with a total size of ScreenHeight * ScreenWidth. Adjacent pixels in a row are adjacent in memory and the items

in a row wrap around to the next row. By way of example, the first pixel of the first row is at (0,0) at the address held in ScreenBitsPtr, the next pixel of the first row (1,0) is at the next byte address and subsequent pixels in that row are at increasing addresses. When we reach the end of the row (639,0), the next byte contains the pixel color for (0,1) -- the first pixel of the second row. The next pixel (1,1) will be at the next memory address, and so on…

Your line draw routine will look like this:

```
DrawLine PROTO x0:DWORD, y0:DWORD, x1:DWORD, y1:DWORD, color:DWORD
```

Where the screen coordinates (x0,y0) is the start of the line and (x1,y1) is the end of the line. All coordinates are given as regular unsigned integers. Remember that the screen is 640x480, so note the valid value ranges for the the x and y coordinates respectively. You should be able to draw any type of line within this valid range. The `color` parameter specifies the color...duh. This is also a DWORD value, but as you may recall our pixel depth is 8 bits. So this procedure will only consider the least significant byte of `color` and use that as the color for the line.

You can use the following pseudocode to implement your LineDraw procedure. Note that use of ABS(), INT_TO_FIXED(), FIXED_TO_INT(), PLOT() are not meant to be proper function calls, merely they exist to abstract some of the nitty gritty details. PLOT(x,y,c) should draw a single dot at (x,y) with color c.

```
DrawLine(x0, y0, x1, y1, c)
  integer i;
  fixed fixed_inc, fixed_j;

  if (ABS(y1 - y0) < ABS(x1 - x0))
    fixed_inc = INT_TO_FIXED(y1 - y0) / INT_TO_FIXED(x1 - x0);

    if (x0 > x1)
      SWAP(x0, x1);
      fixed_j = INT_TO_FIXED(y1);
    else
      fixed_j = INT_TO_FIXED(y0);

    for(i = x0 to x1)
      PLOT(i, FIXED_TO_INT(fixed_j), c);
      fixed_j += fixed_inc;
```

```
else if (y1 != y0)
  fixed_inc = INT_TO_FIXED(x1 - x0) /  INT_TO_FIXED(y1 - y0);

  if (y0 > y1)
    SWAP(y0, y1);
    fixed_j = INT_TO_FIXED(x1);
  else
    fixed_j = INT_TO_FIXED(x0);

  for(i = y0 to y1)
    PLOT(FIXED_TO_INT(fixed_j), i, c);
    fixed_j += fixed_inc;
```

Library Routines

Our library routines will test your procedures (and allow you to test them with various input values). You don't need to call the routines yourself. If you want to use them to upgrade your starfield from the last assignment, feel free to do so.

Getting Started

Download and unpack the assignment files from the courseweb page. Copy your stars.asm from the last assignment -- you will need it for this assignment and all the subsequent ones. You can compile and execute your program from the command line or with the included make.bat script. When everything is successfully compiled, your should be able to run `lines.exe`. This should allow you to test/demo your assignment.

Submit your assignment on canvas as a single file called `lines.asm`. You should implement the **`FixedCos`**, **`FixedSin`**, and **`DrawLine`** routines as described above. Comment your code. Make sure that any code that you write for this assignment is in this file and that it correctly compiles using the standard `make.bat` file included with this assignment. Please clearly identify your name in the comments. All programming assignments in this class are to be completed independently. No collaboration!