Apple II Developer´s Conference

Software Design with an EMPHASIS on games

I. Introduction

a. Definitions - Some terms to get us started.

Data Structure - a method of representing data.

Masking - masking is necessary when a background is to appear underneath a sprite.

Sprites - are objects, for instance the paddle in Arkanoid is a sprite.

VBL - Short for video blank. This is the period in which the computer isn´t scanning (redrawing) part of the screen.

Video synchronization - A small loop that waits for the beam to go off the screen (generally) and then exits.

b. Sample code

Masking - to mask, you must do the following:

1. Load the screen data
2. And the screen data with the sprite mask
3. Or the screen data to the sprite data
4. Store in to memory.

This implies that each object needs a mask.

Synchronization - can be achieved by monitoring $c02e or $c019. See the Apple Technotes about $c02e.

8 bit a

```
]lp     ldal    $e1c019
        bmi     ]lp
]lp2    ldal    $e1c019
        bpl     ]lp
```

II. Design Stage

a. What game do you wish to spend a good part of your life on?
Factors involved: skill level, artist available, time, feasibility, music.

The design stage is crucial. All of the factors involved must be there, or the game will never be finished.

III. Putting it all together

a. Grafix and sound. There isn't much to say about the artwork and music. It must be finished, but that's the artist's and musician's problem. It isn't a bad idea to use temporary shapes to begin with, and music can always be added later.

b. Code. The code is the challenging part. A lot of factors must be accounted for when writing a program of any type.

i. Screen update. If you don't have stackbased update libraries, then try using direct screen access. You might be pleasantly surprised at what the GS can do at 2.8 mhz. It wouldn't be a bad idea to write some basic screen routines, to speed your development time. If you know you are going to need fast updates, then you should skip the slow routines and work on the fast ones. The basic concept of smooth animation is to update the screen during the VBL. It is impossible to update the whole screen in a VBL, but we can try. Use the video sync to reduce tears in animation, and to make the program accelerator friendly.

ii. Sprites. If your sprites need to be masked onto the background, then your code must handle it. Unfortunately, the GS hardware doesn't support sprites. You should design a data structure for storing sprites and masks.

iii. Gameplay. The events that happen within the game are important. Your main loop is crucial, because it makes the events happen.

c. Optimizing. Now that your code works, how can you make it better? The best way to determine what part of the code needs to be optimized is to use the program and then decide what can be improved. I'll focus on optimizing the screen updates, because making your screen updates faster will also speed up your gameplay.

i. Screen updates. This can be a problem as the GS graphics are bottle necked at 1 mhz.

Unrolling loops. In Merlin your code might look like this: (to draw an 8 x 8 block)

```
]screen    =    $e12000
]sdata     =    0

                     ;x contains offset on screen
                     ;y contains offset in shape table

          lup  8          ;number of lines tall
          lda  ]sdata,y   ;loads from shape, stores to screen
          stal ]screen,x  ;does 4 pixels (assuming 16 bit)
          lda  ]sdata+2,y ;loads next 4 and stores
          stal ]screen+2,x
]sdata    equ  ]sdata+4   ;increment 4 bytes into shape data
]screen   equ  ]screen+160 ;jump to the next line on the
          --^              ;screen.
```

This lup would get expanded to a bunch of lda's and stal's. For example:

```
          lda  $0000,y
          stal $e12000,x
          lda  $0002,y
          stal $e12002,x
          lda  $0004,y
          stal $e12004,x
          lda  $0006,y
          stal $e12006,x
          etc.
```

Read from bank $01. When reading the SHR memory, remember that reading from bank $01 (assuming shadowing is on) occurs at fast speed, not 1 mhz. This can be important when masking sprites on the screen.

Compiled shapes. Load, store and mask only the necessary parts of sprites. So, if the sprite is large but predominantly see through, a compiled shape would help. The disadvantage is the sprite must also be erased.

Stack based or TSB/TRB screen updates. This depends on how you decide to do you animations. The Stack/TSB/TRB update relies on the idea that you turn off shadowing, and update bank $01 at 2.8 mhz, then use a Stack/TSB/TRB update routine to shadow the information onto the video screen. Task Force uses compiled shapes and a TRB/TSB update routine.

A sample loop for updating the screen might look like this:

```
     turn off shadowing
     draw background
     mask sprites onto background
     Turn on shadowing
     Stack/TSB/TRB update
```

A simpler variation of this is (assuming static background):

```
     shadowing is on
     mask sprites
```

The difference is in the mask routine and the sprites. The mask routine would look like the following:

```
        ldal $012000,x
        and  mask,y
        ora  data,y
        stal $e12000,x
```

The routine preserves bank $01. Using this method, the sprites cannot be compiled. Also, the sprites must be a little larger than normal, so the routine will restore the other parts of the screen.

ii. General tips. The cliche goes "90% of the time spent is in 10% of the code". Two basic ways to optimize this, make the routine faster, and/ or find out why the code is spending so much time in this part of the game. From my experience, 1 or 2 cycles don't do that much, unless they are in a loop, or nested loop (even more so).

IV. An example! Dueltris.

V. Where to get help. Don't be afraid to ask!

a. Other computer platforms. This is something I'd really like to encourage. IBMs, Amigas and Macs are great places to get game ideas or maybe even graphics, music or code, although you should use their graphics, music or code without permission.

b. The Internet. The Internet is a great source for information. If you can't get a school account, try a Proline or commercial site.

For more information about games, 3200's, DreamGrafix, or just life in general, you can contact us through these channels...

DreamWorld Software
P.O. Box 830
Iowa City, IA 52244-0830

America_Online:	DWS Steve
Genie:	S.Chiang4
Internet:	stc7@cunixb.cc.columbia.edu
	dwsjason@gator.netcom.com (I think)

Customary Plug

Stop by our booth, and check out the only 16, 256 and 3200 color paint program, DreamGrafix, and our newest in entertainment software. Apple II forever!