Subject: Tools or No Tools ?
Posted by Oz on Sun, 05 Oct 2014 19:49:01 GMT
View Forum Message <> Reply to Message

When it is time to consider writing Apple IIgs video games or a demo, the first question is usually to decide which would be the best environment (Operating system) to use.

There are 3 available options :

- Proprietary OS
- Prodos 8
- GS/OS

Everyone who has watched the FTA Demos (Nucleus, Modulae, XMas, Delta...) / Games (Space Harrier, Bouncin Ferno, Oil Landers...) / Utilities (Photonix, NoiseTracker), the Miami Software (aka F.*.C.K) products (Teenage Queen, Sensei, Show 3200, Hot Cookies, Space Shark, ZZ Copy...) or some of the Mr Z Demo (California Demo, Pom's / Toolbox disk introduction, Crack Intro...) have been amazed by what a basic Apple IIgs could do. For a moment, the slow computer they were facing everyday with Prodos 16 was muted into a computer where everything was fast, in color and in stereo ! Most of these products use a proprietary OS, very light and very fast to be loaded from a 3,5 inches disk.

Most of the game, even the best ones like Rastan, were based on a Prodos 8 system. Few of them, especially the latest ones, were GS/OS compatible (Out Of this World, Wolf 3D, Ultima I) and NOT protected against the copy.

The real question is about GS/OS. Is the choice of GS/OS a constraint or an advantage for Game programmers ? Why so many people were ignoring GS/OS when they where writing their games ? Could we write also Demos under GS/OS ?

It is important to understand that programming a video game under GS/OS won't prevent you to use all the required tricks for fast sprites animation, scrolling & co. Using GS/OS DOES NOT MEAN using Quickdraw to write pixels on the screen. Using GS/OS, it is just about allocating memory and making sure we do not blow up other programs memory spaces. The 'No Tools' and 'No GS/OS' is not the same. You can program low level and use the GS/OS to allocate for you the right memory space before using it. And when it will be time to quit the game, free the allocated memory. There is no constraints to use GS/OS. Only advantages (SynthLab / Tool 219 / Tool 220 musics).

The only reason why mot of the game were using Prodos 8 at the time was linked to the small memory size of the Apple IIgs (in the USA, some of them had only 512 KB). So, in order to keep the maximum memory space for the game itself, the choice of Prodos 8 was done (it is also to have the maximum space on the 800 KB disk, where Prodos 8 was taken less space than Prodos 16 or GS/OS). For anyone having 2 MB or more, GS/OS is much more friendly environment. You can install your games on your hard drive, launch them and go back to the finder when exit. Because the hard drive is faster than the disk, the games loading is faster. Today, with the use of emulators, using a image disk for a game (and booting from the disk) is painful.

The specific OS, like the ones we had at the Apple II time, has been use mostly for demos or games. Because such programs do no require to write something back to disk, the usage of the disk was only done by fast read-block access. This is fast but once again, such program can't benefit of a better Apple IIgs configuration. They are limited to the usage of the 800 KB disk drive and can't be installed on a hard drive (which would be faster, at the end, than a 800 KB drive).

In order to show that programming under GS/OS in not complex, you will find next a skeleton of program. Build it with Merlin 16+ and you will get a S16 executable file. Launch it from GS/OS. Hit a key to quit. The program does nothing spectacular but this could be a base for any game or demo under GS/SOS.

The code itself is self-explanatory, there is nothing tricky. You can notice that the only thing we really need, as game programmer, is an access to the $01/2000 area, which is the graphic page once the Shadowing is active. We simply need to ask GS/OS for this memory area. Because the Apple IIgs is a single process at a time system, once your program is running you are the only one doing something and you have full CPU power. In the case GS/OS still need to interrupt your program (AppleTalk, Sound Interrupt, Graphic Interrupt...), you can protect yourself by using the SEI /CLI opcodes. All code between a SEI and the CLI can't be interrupted, so you can do what you want (we think here about the move of the Stack & Direct Page in Bank $01 for fast graphic routines).

Feel free to ask any explanations about this small code set.

```
*-----  Merlin 16+ Directives

        mx      %00             ; assemble in 16 bit

        rel                     ; Build a relocated S16 file
        dsk     Hello.l         ; Name of your program on disk

        use     4/Locator.Macs  ; Macro Definition Files
        use     4/Mem.Macs
        use     4/Misc.Macs
        use     4/Util.Macs
        use     4/Sound.Macs

*-----  Begin Of Program   ----------

        PHK                     ; Data Bank Register = Program Bank Register
        PLB

        CLC                     ; 16 bit
        XCE
        REP     #$30

        JSR     ToolInit        ; Init Tools + Compact Memory + Ask Shadowing
        JSR     BackupEnv       ; Backup environment (colors...)
```

```
*-----   Your Code Starts Here   ----------

         JSR      WaitForKey        ; Wait until a Key is pressed

*-----   End Of Program   ---------

End        JSR      RestoreEnv        ; Restore environment (colors...)

         JSR      ToolTerm          ; End up Tools
         JMP      Exit              ; Quit to the Launcher

**********************************************************
*******        INIT TOOL SET/ FREE TOOL CODE       *******
**********************************************************

ToolInit    _TLStartUp                 ; Start Tools
         PHA
         _MMStartUp                  ; Start Memory Manager Tool Set
         PLA
         STA      myID               ; Get current ID
*--
         _MTStartUp                  ; Start Miscellaneous Tool Set
*--
         PushLong  #0                ; Allocate Page Direct in Bank 00
         PushLong  #$000100
         PushWord  myID
         PushWord  #$C005            ; Fixed, Page Aligned, Locked, Unpurgeable
         PushLong  #0
         _NewHandle
         PLX                         ; Handle Low  Address
         PLA                         ; Handle High Address (00XX bank)
         XBA
         STA      TI_1+2
TI_1       LDAL     $000000,X          ; Get Low Address  A=??/XXXX
         PHA
*--
         _SoundStartUp               ; Start Sound Tool Set
*--
         PushLong  #0                ; Compact Memory
         PushLong  #$8fffff
         PushWord  myID
         PushWord  #%11000000_00000000
         PushLong  #0
         _NewHandle
         _DisposeHandle
         _CompactMem
*--
         PushLong  #0                ; Ask Shadowing Screen ($8000 bytes from $01/2000)
```

```
          PushLong  #$8000
          PushWord  myID
          PushWord  #%11000000_00000011
          PushLong  #$012000
          _NewHandle
          PLA
          PLA
*--
          RTS

*-------

ToolTerm    _SoundShutDown              ; Stop Tools
          _MTShutDown
          PushWord  myID
          _DisposeAll
          PushWord  myID
          _MMShutDown
          _TLShutDown
          RTS

myID       ds       2                  ; ID of this Program in memory

*----------------------------------------

BackupEnv   SEP      #$30              ; Backup Environment values (color, border...)
          LDAL     $00C022
          STA      BE_C022
          LDAL     $00C029
          STA      BE_C029
          LDAL     $00C034
          STA      BE_C034
          LDAL     $00C035
          STA      BE_C035
          REP      #$30
          RTS

*-----

RestoreEnv  SEP      #$30              ; Restore Environment values (color, border...)
          LDA      BE_C035
          STAL     $00C035
          LDA      BE_C034
          STAL     $00C034
          LDA      BE_C029
          STAL     $00C029
          LDA      BE_C022
          STAL     $00C022
          REP      #$30
          RTS

BE_C022    HEX      00              ; Background Color
BE_C029    HEX      00              ; Linearization of the Graphic Page
BE_C034    HEX      00              ; Border Color
BE_C035    HEX      00              ; Shadowing

************************************************************
*******                GS/OS CODE               *******
************************************************************

GSOS       =        $E100A8

*-------

Exit       JSL      GSOS            ; Quit Program
          dw       $2029
          adrl     gsosQUIT

*-------

gsosQUIT   dw       2               ; pCount
          ds       4               ; pathname
          ds       2               ; flags


************************************************************
*******          EVENT HANDLER CODE             *******
************************************************************

WaitForKey  SEP      #$30           ; Wait for a Key Press
WFK_1      LDAL     $00c000
          BPL      WFK_1
          STAL     $00c010
          REP      #$30
          RTS

************************************************************
```