Portable Demo Coding

Markku 'Marq' Reunanen
marq@iki.fi

Assembly'05

Agenda

- Personal background
- Overview of demo porting
- Why? Why not?
- Strategies for porting
- Portable development tools
- Common pitfalls and code examples

Personal background

- Working as a coder in Fit and Lieves!Tuore (MSX)
- Demo coding/porting on several platforms:
  - MS-DOS (1991-)
  - Amiga (1997-)
  - MSX (1997-1999)
  - Linux (1998-)
  - Atari Falcon (1999-2000)
  - Mac OS X (2002-)
  - Plus some others: GP32, IRIX, Dreamcast ...
- Also did some real work on some of these ;v)

Short history of demo porting (1)

- In the 80's and early 90's games were largely multiplatform (C64-Spectrum-Amstrad, Amiga-Atari-PC)
- Demos traditionally less so
- In mid-90's things started to change. Three early multiplatform demos:
  - HElliZER by QMG (1996)
  - Hard Rox by Skal (Wired 1996)
  - Megademo IV 2 by Artwork (MS 1998)

Short history of demo porting (2)

- Nowadays not a freak thing any more
- Some well-known multiplaform prods:
  - VIP2 invitation by Popsy Team (Win, Linux, OSX)
  - Dose2 by mfx (Win, Linux, OSX, others)
  - State of Mind by Bomb (DOS, Win, Linux, others)
  - Obsoleet by Unreal voodoo (Win, Linux)
  - Variform by Kewlers (Win, OSX)
- Some groups that have done multiplatform stuff:
  - Ananasmurska, Astral, Bandwagon, Bomb, Excess, Fit, Fresh!Mindworkz, Kewlers, mfx, MovSD, nan5/Bypass, PHn, Pwp, RGBA, Unreal voodoo, Woorlic

What's the difference? (1)

- In 80's and early 90's demos were tied to the hardware: showing what it's capable of
- Less so in mid-90's
  - Amiga: c2p, HAM8, module replayers such as proreplay & The player
  - PC: mode 13h and linear VESA2 modes, module replayers such as Midas
- Was there any difference?
  - Both played modules
  - Both had some sort of chunky framebuffer access
  - Effects not hardware based any more

What's the difference? (2)

- Nowadays...
  - Frame buffer access / OpenGL
  - Module/mp3/ogg replayers
  - Timer
  - Focus on design and algorithms instead of hardware banging
  - C and C++ instead of assembler
- Meaningful hardware differences:
  - Processor speed
  - Processor byteorder
  - FPU or not (embedded systems)
  - Amount of memory (embedded systems)

Why port?

- You might learn a lot
- Portable code tends to be cleaner
- Larger audience
- Nice alternative scenes
- See your stuff running on exotic devices
- Not a big deal once you know how

Why not port?

- Extra effort
- Hardware specific tricks not available
  - More relevant on old and low-end devices
  - Access to vertex/pixel shaders not yet fully standardized
- OS specific libraries etc. not available
  - Size optimization tricky
- The worst target platform dictates the overall result
- All of these can be overcome to a degree, but not completely :v(

Porting strategies

- Write your own wrapper libraries
  - Small, optimized
  - You learn a lot
  - Error-prone
- Use common libraries
  - Easy
  - No need to deal with platform specific bugs
  - Might be bloated
- Open source?
  - Someone else might do the work for you
  - Source code safe in many places, longer lifespan
  - But my code is so messy!

Programming languages

- Java
  - Portable by nature
  - Bad implementations, possibly slow
  - Not universally available
- C/C++
  - Portable in theory
  - C++: bad implementations, evolving standard, not available for low-end devices
  - C: great availability, but rather crude
- Others?
  - Pascal isn't that bad ;v)
  - Interpreted languages tend to be too slow

Multiplatform APIs

- 2D graphics
  - SDL (provides timing as well)
  - PTC
- 3D graphics
  - OpenGL:
    - The only portable choice
    - Lags behind in features
  - OpenGL frontends:
    - SDL, GLUT, others
- Sound
  - SDL, Portaudio
  - fmod, mikmod, mpg123, Midas (outdated)

Programming tools

- GCC
  - Free
  - Optimizes quite well
  - Available and supports cross-development
  - Cygwin & MinGW for Windows
- Make
  - Makefiles make life easier when porting
    - Compiler-specific workspaces generally not portable
  - Easy to switch compilers and compiler flags. Even VC++ can be used.
  - Takes some time to learn & maintain
  - Numerous alternatives such as jam exist

(GNU) Make example

```
CC = gcc
CFLAGS = -O2 -ffast-math
LDFLAGS = -lGL -lm
OBJ = main.o player.o effect.o

demo: $(OBJ)
	$(CC) -o $@ $(OBJ) $(LDFLAGS)

%.o: %.c
	$(CC) $(CFLAGS) -c $<

clean:
	rm *.o *.bak demo
```

Practical examples

- Finally, the good stuff
- Common pitfalls and practical code examples
- A more complete list available at:
  - http://ftp.kameli.net/pub/fit/misc/portability.txt

Encapsulation

- System-specific details hidden in modules and subroutines:
  - No need to search and replace them anywhere
  - Easier to rewrite just the specific routine
- Makes code better structured anyway
- This was probably a little trivial ;v)

A word on timing

- CPU-loops for timing are a thing of the past
  - Will break in both slower and faster computers
- High-resolution alarm timers
  - Different resolution in different systems
  - May slow down with system load
- Reading a simple high-resolution timer counter
  - Usually enough for timing
  - POSIX: gettimeofday()
  - SDL: SDL_GetTicks()
  - Sample counters usable for music sync

Endian worries (1)

- Little endian
  - x86, ARM (default), Alpha, SH-4, Z80, ...
- Big endian
  - PPC, MIPS, SPARC, 68k, 6809, ...
- What does it mean?
  - long a = 0x12345678;
  - In memory:
    - LE: 78 56 34 12
    - BE: 12 34 56 78
- Affects 16-bit and larger types

Endian worries (2)

- Construct 16-bit and larger entities with shifts
- Bad practice:

```
fread(&a,1,4,fp);
```

- Read one byte at a time and shift instead:

```
for(n=a=0;n<4;n++)
    a=(a<<8)+fgetc(fp);
```

- *Usually* this goes for 32-bit framebuffers as well: they are in native byte order
- Rare exceptions do exist. For example SDL has `SDL_MapRGB()`
- For OpenGL RGBA is RGBA in memory as well

Data type tricks (1)

- 'char' can be signed or unsigned
  - use the full type like 'signed char' for portability
- 'long' is 32-bit minimum, 'int' only 16-bit on some compilers
- Pointers don't necessarily fit in 'long'
- Globally defining your data types might not be a bad idea
- Struct members are automatically aligned in some platforms
  - access members only by their name and use sizeof() to get the real size of the struct

Data type tricks (2)

- Alignment can cause other unwanted behavior too:
  - Accessing unaligned 16-bit or 32-bit values may produce wrong results or cause an exception
  - Unaligned access is usually slow anyway
- Unix systems can traditionally handle large local arrays on stack:

```
void funkkari(void)
{
    int biig[1000000];
}
```

  - Not all systems do. `malloc()` or 'static int' solves the problem.

Misc. tips (1)

- Don't assume that uninitialized local variables are zero
- In modern systems string constants are read-only:
  - no-no:

```
char *text = “Zapp”;
text[1] = 'o';
```

  - better:

```
char text[] = “Zapp”;
```

- When using a function, #include it as well:
  - Compiler errors and odd bugs likely if you don't
  - For example math.h should be there if floats are used

Misc. tips (2)

- main() should exit with status `EXIT_SUCCESS`
  - For example OSX displays a warning if it doesn't
  - Ok, you're lazy. Return with zero is fine as well.
- For debug prints it's a good idea to flush the output
  - Most systems flush when \n is encountered
  - If you want to be sure, use `fflush()` or `<< flush;`
  - Without flushing the text might not get displayed
- Dealing with paths
  - Most systems can handle slash '/' as a separator
  - Better yet, prefix file paths with a constant:
    - `fp = fopen(DATAS "duck.3ds","rb");`

Source code formatting

- Tabulator width cannot be trusted
  - Mostly it is eight characters wide, but not always
  - Better save tabs as spaces to keep the source readable
- Case does matter
  - In many systems 'foobar.h', 'foobar.H' and 'FOOBAR.H' refer to a different file
  - Keep the case consistent
- Anything else than 7-bit ASCII is likely to break in some environment/editor
- The same goes for linefeeds as well. In practice all compilers eat at least the single character linefeed.

That's it!

Thank you for your attention
Any questions?