Arashi in Black and White

It started with a challenge:

> **"Calling all Programmers - Arashi in black and white**

I've been playing with trying to get Arashi running on my SE/30. I can get it to build okay, and even not to die with an error when I run in in Black & White mode - but I can't actually get it to display anything on screen either.

I wondered if anyone here might have more luck? B&W Arashi, on the SE/30 and Classic II would be an excellent addition to the Mac Garden - even if a version for the Mac Plus is probably impossible!

What do you think? Are you interested in this project? I'll upload what changes I've made to GitHub and you could have a crack at the puzzle too…"

Why we did it

There were two initial reasons for this project - and two more which became apparent later. The first two are…

1. Arashi is a fantastic game, and very enjoyable on any colour Macintosh - but it doesn't actually need to be a colour game. Tempest, the game that it is based on, is very enjoyable on a Vectrex - and that can't display colour at all. Wouldn't it be great if the compact Macs - and, at the very least, the more powerful compact Macs (Classic II, Performa 200, SE/30) and 1bit PowerBooks (PowerBook 140 and PowerBook 170) could play Arashi too? A stretch goal would be for it to be playable on 68000 Macs (Plus, SE, Portable, PowerBook 100) - but that doesn't appear to be feasible for performance reasons.
2. New games are being released seemingly every day for all kinds of Retro computers. Commodore 64, Spectrum, BBC, IBM PC, Amiga, ST and even for obscurities like the MTX, Oric and Enterprise - but few new games are being released for the 68k classic Macintoshes. It's time to redress the balance.

The reasons that became apparent later are…

1. It proved to be an interesting conversation and way to meet, and learn from, talented software developers who also have an interest in the classic Macintosh.

2. Hopefully it can become a seed for future 68k Macintosh software development - and especially for those original Macs.

End result of how

After a great deal of hard work and experimentation, several changes were identified as necessary to enable Arashi to run in Black and White. Additionally, several optimisations were identified to improve the speed on lower performance black and white models.

- Mapped colors to black and white.
- "New drawing engine".
- Optional dotted lines, for slower machines.
- NOTE: Text (which uses lines) is never drawn dotted if that option is selected, since it is unreadable if drawn that way.
- Skipping drawing every other "spot" and "pixel", for slower machines.
- Inlined C for all drawing when possible (huge boost).
- A few new routines to replace palette animation of original.
- Support 68000 only machines for 1.1.0 (but too slow to be playable un-accelerated!)
- Support for 1 bit and 8 bit displays only for non-QuickDraw versions.

NOTE: 8 bit version still draws only in black and white. I implemented it only for two reasons: It was very easy to draw as each pixel is just a byte, and it allowed me to use MacOS X 10.4's Classic mode on a fast machine. (MacOS X doesn't support 1 bit mode).

What is included here or in github

For both versions 1.1.0 and 1.1.1 (included as much as possible)

- DiskCopy 6 images
- Disk images for MvM
- Sit files
- Source code
- Prebuilt binaries
- Think C 5 (for 1.1.0) and Symantec C++ 7 (for 1.1.1)

The github repo is https://github.com/laulandn/Arashi

Quick overview of source

NOTE: I didn't write any of these, just patched them!
VAKit: Really cool Vector drawing library, could be used by your own code outside of Arashi.
SoundKit: Really cool sound playing library, could be used by your own code outside of Arashi
Class library: A string/dictionary library, in Think C w/ Objects for 1.1.0 and C++ for 1.1.1
Misc: A few hacky things, such as init code and other MacOS specifics.
Game Source: The game itself, in various folders.

Note:

Tries and versions along the way

- Was unable to use existing drawing code, it was specific to 8 bit indexed.
- Original has a lot of assembly - anyone with 68000 assembly expertise on might be able to get Arashi running at acceptable speed on original Macs, but the replacements would have to be written from scratch. An exercise for the reader, perhaps.
- Original uses "Dual buffering" via palette switching, ie not REALLY dual buffers.
- The first try was using QuickDraw, but was too slow on '030's, although runs well on '040's and all but slowest PPC.
- Added "New drawing engine", in C, optimized and inlined as much as possible.
- Renamed unused and some dup functions and files to make clearer that they were so.
- Did some work to allow all code to be fully C++ compilable (useful for future non-Think/Symantec building).
- Added #ifdef NICK_NCQD that eliminates ALL Color QuickDraw use…so can run on Mac Plus/SE/etc (which only have old QuickDraw in rom).
- Adjusted some line spacing and made a bit more logical.
- Modernized most function def's that were old K&R (for non-Think/Symantec).
- Added function prototypes and includes where needed (for non-Think/Symantec compilers).
- Started work on CodeWarrior m68k version, builds, but crashes, not sure why.
- Started Carbon version, builds, but problems with Mac resources, I think(?)…also needs replacement for assembly, and a lot of other stuff, like low memory global access (mostly mouse reading).
- Kept 1.1.0 and 1.1.1 versions in sync as much as possible, since 1.1.0 is smaller and will compile with Think C 5, which uses less memory and can run on Mac Plus with enough room.

Different versions you can build

Both Arashi 1.1.0 (Think C 5) and 1.1.1 (Symantec C++ 7) are buildable. You'll probably only want to use 1.1.0 for low end machines.

Pre-built binaries supplied are…

- Arashi*bw_ncqd: No-ColorQuickDraw version for 68000's. (Too slow if not accel'd!)
- Arashi*bw_1bit: "New" graphics routines only runs on B+W screens.
- Arashi*bw_8bit: "New" graphics routines only runs on 256 color screens.
- Arashi*bw_qd: Pure QuickDraw, should run on any hardware, but slower (`040+ ?).

The 1bit and 8bit versions have code to detect the current screen mode and will refuse to run if it does not match.

The pure QuickDraw version will run correctly in any depth.

The ncqd version does not include any code to check depth (it'd require Color QuickDraw) and so WILL happily (and visually terribly) run if the wrong screen depth is used.

ncqd should be built to draw with dotted lines (for speed), the others with solid ones.

What all the #define's do

In no particular order. Be sure to do a full recompile if changing any of these.

NOTE: Some of these are in VA.h, but NICK_NCQD must be set globally in your compiler prefix/headers options. Sorry!

Uncomment for ALL original code. This disables all funcational changes I have made.

```
#define ORIG_ARASHI 1
```

Uncomment to use "new" drawing engine as opposed to QuickDraw

NOTE: QuickDraw should be used for running in MacOS X classic, certain graphics cards, or to run in any other mode than black and white or 256 colors.

```
#define NICK_NOT_QUICKDRAW 1
```

Uncomment to use 8 bit instead of 1 bit pixels in"new" drawing engine.

NOTE: This will allow you to run on a 256 color screen (but still in black and white).

```
#define NICK_256_NOT_1
```

Uncomment 2 below for "new" drawing engine to draw non-solid lines

NOTE: “statics” are lines that don’t move.

```
#define NICK_LINES_DOTTED 1

#define NICK_STATICS_DOTTED 1
```

This is the pattern for dotted lines, can be different for different uses (and may be defined differently in multiple files). This is logically AND’d with the coord along the line so 1 will skip every other pixel, 2 will draw two pixels then skip two. Try the values and you’ll see anything other than 1 can make things hard to see.

```
DOT_PATTERN
```

Uncomment 2 below for "new" drawing engine to skip drawing every other pixel/spot

NOTE: “spots” are larger dots than “pixels”, drawn with more than one pixel.

```
#define NICK_SKIP_PIXELS 1

#define NICK_SKIP_SPOTS 1
```

This will force checking all writes/reads are actually within screen buffer ram for new engine.

NOTE: This will slow drawing down if used, and is usually not necessary, as the game code is pretty solid.

```
NICK_CHECK_COORDS
```

This will leave ALL assembly out…might work, mostly replaced with C…but a lot still needs some replacements to be done in some places :(

```
DONT_USE_ASM
```

This will leave out ALL Color QuickDraw calls, needed for pure 68000 (the roms of Macs that shipped with 68000 processors did not include it, and will crash if poorly written code that uses it is run). This probably only works for Arashi 1.1.0 (not 1.1.1)!

NOTE: Color QuickDraw is more advanced, even for B+W use, used for such things as detecting screen depth. This means if you use this, the game will happily (and visibly terribly) run on the WRONG screen depths, as the depth checking code can't work!

`NICK_NCQD`

This one is actually pre-defined if you are building a Carbon app…never finished :(

`TARGET_API_MAC_CARBON`

Define this to include <Gestalt.h> instead of <GestaltEqu.h>, needed for some older/odd compilers

`GESTALT_NOT_GESTALTEQU`

Define this to leave out all pre-Sound Manager code…might work…but there's no replacement for some important things :(

`NEW_MAC_SOUND_ONLY`

Define this to avoid doing any MacOS trap patching

NOTE: This will avoid things like disabling screensavers, but is necessary with some compilers, or Carbon.

`DONT_PATCH_TRAPS`

What I didn't have time to do

- Fully eliminate double buffering. There is still residual code that causes moving things to be drawn twice (in different locations) on the screen…this usually happens too fast to notice.
- Carbon support, started but crashes - will be necessary for a MacOS X PPC and older (up to 10.6?) Intel builds. Possibly 4 vs 2 byte int's? NOTE: It currently runs fine in MacOS X Classic when using QuickDraw. "New" drawing engine does NOT draw correctly on 10.4(?) or newer.
- Pomme support (https://github.com/jorio/Pomme) - will be necessary for modern Intel / Silicon builds.
- Support building with CodeWarrior, started, but crashes for unknown reason. Possibly 4 vs 2 byte int's?
- Native PowerPC (MacOS 7/8/9) support. (M68k versions run fine in emulation)
- Still some text drawing bugs, maybe. Some text still looks a little distorted.

- Smarter board redraw, it's currently redrawn every frame! There is not an easy way around this, and it slows things down, probably quite a bit.
- Aspect ratio bugs in original code, this shows up on certain screen sizes only. This causes overlap and wrong positioning/size of some line and text drawing.
- ALL "spots" and "pixels" are still being computed, but not drawn, even when the NICK_SKIP_'s are defined. This would help with speed, but unknown how much.
- Assembly version of "new" drawing - may be necessary for acceptable performance on 68000 and slow '030 Macs, if done really well. The C is (probably) as good as it can get currently.
- Tighten code, more inlining, maybe make some features (some kinds of enemies and animations?) optional - all probably needed for performance 68000 and slower '030 Macs.
- I believe 68000 performance may be more compute and less drawing bound, so things like changing 32 bit values to 16 bits (the 68000's and some '030's data bus is only 16 bit), and making sure all data is word aligned may be needed.
- Also for 68000 and slow '030's: Eliminate as many multiplies/divisions as possible (change to shifts), and use lookup tables to eliminate computation of predictable values (especially angles!).
- Eliminate or optimize screen size support. Currently everything is automatically scaled. If we only support specific sizes (such as a 512x342 window) this will eliminate all those computations and should speed things up. This would also eliminate many "expensive" multiplies/divisions.
- Add code that'd detect screen depth for the NCQD version and refuse to run if it wasn't set to 1 or 8 bit (depending on which was define'd).