ByteSex and Squeak

Tim Rowledge tim@rowledge.org

Squeak was originally written for the PowerMac, a big-endian system. Without wishing to get involved too heavily in the bytesex wars, this was a mistake. If the gods had meant us to do things in a big-endian manner they would have given us warped brains. Many important systems are little-endian, including the Acorn RPC and that obscure Intel x86 CPU that serves so well as a space heater. Given this unfortunate state of affairs, it was neccessary to alter Squeak to be more friendly to little-endian systems. This involved three sets of changes to the Squeak system.

- Loading the image
- Making BitBLT little-endian capable
- Endian swapping the Form bitmaps

I persisted with using this code for a couple of years but since Squeak Central wouldn't adopt it the code was eventually withdrawn. Current Squeak (as of about version 3.4) has much of the functionality built in as part of supporting external bitmaps on little endian hardware so that hardware acceleration can be used.

Loading the image.

Obviously much of an image file is byte-sex dependant -- all the word based values need byte-swapping when an image saved on one variety of machine is loaded on the opposite sort. Byte based values such as strings, large integers etc do not need any swapping. To achieve this we need to:-

1. discover that the image is the wrong endianness for our

loading machine

2. scan the loaded raw image byte-swapping the appropriate items
3. remember to write the header when saving the image such that 1) can be done on the next machine to use this image.

It turns out that we can derive the saver's endianness from the version word in the image header; so long as the image version word is maintained so that a byteswapped image will yield an 'impossible' version number we can do a simple check. Any 16bit number, 6502 for example, will byte-swap to a huge value (1712914432) clearly testable. Once we know we need to byte-swap then the procedure is quite simple;

Scan the entire loaded image in memory, byte-swapping every word. This leaves us with all the pointers, headers and class words in the correct form for our machine, but with Strings etc in the wrong order. Rescan the entire image, making use of the now intelligable headers and class words to re-swap Strings and other byte objects.

Floats are held in Squeak as IEEE format doubles, and machines (such as Intel based PCs) which swap the word order of doubles must handle the word swapping on the fly, rather than complicating the imageload by having to track the double word-order as well as byte-sex.

In the 1.18 release of Squeak this loading algorithm is incorporated in to the Interpreter class code in methods such as `#reverseBytesInImage`

Making BitBLT little-endian capable

Within the `BitBltSimulation`and `WarpBlt`classes, there is some code where the bytesex of the words being manipulated is crucial. In general, anywhere that a mask, shift or scan of a word is done will be bytesex dependant. On a big-endian machine, the high-order bits will form the pixels at the left of the screen word whereas on a little-endian machine they form the right side.

In order to form the left mask for a blt, we need to shift the `AllOnes`mask to the 'pixel-right' so that the affected bits are masked. On a big-endian machine that requires a LEFT shift instruction.

When doing a multi-bit pixel operation such as blting a form to a destination of a different pixel depth, we need to scan the words in the opposite order and carefully reassemble the destination word.

To support this need for two different ways of blting, we tweaked the `CCodeGenerator`to make use of a C macro in the blt code. If `LITTLE_ENDIAN`is #define'd in the makefile, the macro `ifLittleEndianDoelseDo()`will evaluate to the first, little-endian clause and vice-versa. This leads to Smalltalk code that is perhaps less pretty than we would like, but keeping both code-paths together in this way at least helps remind us that the endian issue has to be dealt with in the method. As an example, consider the method

```
!BitBltSimulation methodsFor: 'pixel mapping'!
pickSourcePixels: nPix srcMask: sourcePixMask destMask:
destPixMask
 "This version of pickSourcePixels is for
sourcePixSize <= 8 and colorMap notNil"
 "Pick nPix pixels from the source, mapped by the
color map, and right-justify them in the resulting
destWord."
      | sourceWord destWord sourcePix destPix dstShift
     |
      sourceWord _ (interpreterProxy longAt:
     sourceIndex).
      destWord _ 0.
      self ifLittleEndianDo:
          [dstShift _ 32 -(nPix * destPixSize)]
      elseDo:[].
```

```
    1 to: nPix do: [:i|
        self ifLittleEndianDo:
            [sourcePix _ sourceWord >>
            ( srcBitIndex) bitAnd: sourcePixMask]
        elseDo:[sourcePix _ sourceWord >>
            ((32-sourcePixSize) - srcBitIndex)
            bitAnd: sourcePixMask].

    "look up sourcePix in colorMap"
    destPix _ (interpreterProxy
        fetchWord: sourcePix
        ofObject: colorMap)
        bitAnd: destPixMask.
    self ifLittleEndianDo:
        [destWord _ destWord
            bitOr:( destPix<< dstShift).
            dstShift _ dstShift + destPixSize]
        elseDo:[destWord _
           (destWord<< destPixSize) bitOr: destPix].
    (srcBitIndex _ srcBitIndex + sourcePixSize) > 31
       ifTrue: [srcBitIndex _ srcBitIndex - 32.
    sourceIndex _ sourceIndex + 4.
    sourceWord _ interpreterProxy longAt:
   sourceIndex]].
    ^ destWord!
```

The sections in the `ifLittleEndianDo:elseDo:`are arguably ugly, but at least the entire implementation is gathered in one place, helping overall understanding and maintenance.

The changes to the VM have so far been restricted to destMaskAndPointerInit the mask1 & mask2 bit masks have to be formed by shifting in opposite directions sourceSkewAndPointerInit the skew direction has to be reversed copyLoopPixMap again, the skew direction has to be reversed pickSourcePixels:* the order in which the source pixels are extracted and the result pixels are inserted is different between endians warpLoop again, reverse the skew direction sourcePixAtX:y:pixPerWord: extract the source pixel from opposite end of the word warpSourcePixels:xDeltah:yDeltah: extract and reassemble the pixels differently

Endian swapping Form bitmaps

Form bitmaps are stored as byte objects within the image and so get to be re-reversed during image loading. It might seem to be possible to do a further scan for all the Form objects, track down their bitmap ByteArrays and correct them for byte endianness but bitmaps need to be pixel reversed rather than byte reversed. Not all Forms are eight bits per pixel and so a more flexible reversing algorithm is needed.

We chose to do the Form reversal in the image startup code so that this pixel depth information could be more easily discoverd, and so that other classes might be able to perform the same work, or subclasses of Form could do something different. DisplayScreen for example need not do a pixel reversal since it will get redrawn during the startup sequence.

Two primitives are added to the VM

- primitiveReverseForPixelOfDepth which takes a positive 32 bit number receiver and reverses it for argument deep pixels

- primitiveIsVMLittleEndian which works out if the machine running the VM is little or big endian

If the endianness stored in the image does not match the endianness of the VM, then all Forms are pixel reversed by enumerating their bitmap and calling the primitiveReverseForPixelOfDepth for each word. This allows subclasses to redefine the `#pixelReverse`method when required a d also allows unrelated classes to make use of the same capbility.

In pidgin-code this is:-

```
SystemDictionary startUp
```

```
... self isVMLittleEndian = self isImageLittleEndian
 ifFalse: [ Form startUp.
self isImageLittleEndian: self isVMLittleEndian]. ...
```
leads to
```
Form startUp self withAllSubclasses do:[:cl| cl
 allInstancesDo:[:i| i pixelReverse]]
```
leads to
```
Form pixelReverse array _self bits.
1 to: array size do: [ :i| array
 at: i
 put: ((array at: i)
 reversePixelsOfDepth: depth)]
```
which leads to
```
Integer reversePixelsOfDepth:
primitiveReverseForPixelOfDepth
```
The latest version of the LEBitBlt source files has added some changes that should improve the places where pixel bits are being manipulated by Smalltalk code and that were therefore not fixed by the primitive changes. It should now be possible to read/write Forms to filestream properly, and specifying a Form via the `#extent:fromArray:`messages now understands endianness. Methods such as #pixelAt: now work ok, so users such as #colorAt: should also present no problem. There is still a problem when inspecting Bitmap arrays, since they should really be able to swap the numbers we see while inspecting, but they know nothing of the pixel depth of their owning form. For the moment this is left as an open question awaiting resolution.