Esoteric - A Java Sega Dreamcast Emulator

- Acknowlegments
- Challenges
- Technological demands
- Technical Approachs
- Optimization Strategies
- Applications
- Future Development

Acknowledgments

- Developers of swirly, an opensource dreamcast emulator in C (swirly.sf.net).
- Mr. Ivan Toledo for having developed ”dcemu” which served as a basis for this project
- Authors of LWJGL library for their help and support.
- Author of the dreamcast emulator NullDC for his help, and insight
- Professor Salvador Pinto Abreu for mentoring this project
- JNA (Java Native Access) developers/users community.

Challenges

- The Sega Dreamcast is substancially complex system.
  - Renesas SH-4 processor running at 200 Mhz
  - Third Generation PowerVR card
  - 16 Megabytes of main system memory,8 Megabytes of video memory.
  - Etc...!!!
- Considering the above the challenge to implement a compatible and performant emulator capable of executing open source software is cleary a demanding task for the Java (Tm) platform. Let's see how it turned out.

Technological demands

- Renesas SH-4 processor
  - Very fast Floating Point unit
  - Fast Int → Float , Float → Int conversions
  - 200 Mhz of clock speed
  - Numerous on-chip feautures (Timers, Interrupts,Real Time Clock)
- PowerVR graphics card
  - Deferred Rendering architecture
  - Order Independent Transluncency
  - Supports several kind of texture formats which are not supported by OpenGL (VQ compression)

Technological Demand (cnt)

- Large Memory Areas
  - There is the need to implement a fast way to acess the several memory areas.
  - Read/Write byte,word,double word or quad word.
  - MMIO Handling
- USB like protocol to access the devices which can be connected to the system (Controllers,Keyboards,Mouse)
  - Each connected device requires separate implementation.
  - Large number of possible devices to connect
  - Up to four devices at the same time

Technical Approachs

- Esoteric does extensive use of the NIO architecture
  - Each Memory Area is implemented using a ,declared final, ByteBuffer.
    - For each read/write operation there is a View of underlying ByteBuffer Area, in order to optimize the access to those Buffers.
  - Direct Byte Buffers to simulate the Video Memory in order to optimize the texture uploading.
  - Floating Point registers are implemented using Views over ByteBuffers in order to optimize the float->int conversions.
  - Direct Byte Buffers are used to handle cdrom data transfers
  - Zero-Copy Architecture for the data transfers from the connected devices

Technical Approachs (cnt)

- Use of a C shared library,wrapped around libcdio, for multiplatform cdrom drive access
  - JNA greatly simplifies access to native shared libraries.
  - Libcdio supports all the main operating systems.
- Multi-threaded
  - In order to take advantage of the current,almost ubitiquous,multi core processors,the graphical processing of Esoteric takes place on a separate thread.
- Use of the Substance pluggable look and feel for Swing for better integration with the underlying Operating System
- GCJ compliance in order to deploy Esoteric as an AOT built application if the user/developer so whishes.

Technical Approachs (cnt)

- Several emulation approaches of the Hitachi SH4 processor
  - Interpreter – Slow but more compatible.
    - Implemented as a switch statement over the available opcodes.
  - Fast Interpreter/Dynarec – Builds code blocks of the code being executed.Reduces the dispatch overhead of the interpreter, and creates self contained code units (CodeBlocks).
    - Implemented in pure Java taking advantage of anonymous inner classes to build a representation of the context of the correspondent opcode in that code block.
    - Each anonymous inner class implements a private defined Interface (NativeOpcode).

Technical Approachs(cnt)

- Example of the generation of a ”Native Opcode”

```
case  NativeOpcodeConstants.MOVI :
        args[RM] = Sh4Context.RM(instruction);
        args[RN] = Sh4Context.RN(instruction);
        CurrentCodeBlock.InstructionList.add(new NativeOpcode(){
        final int m = args[RM];
        final int n = args[RN];
        public void call() {
              if ((m&0x80)==0) Emu.sh4cpu.registers[n]=(0x000000FF & m);
              else Emu.sh4cpu.registers[n] =(0xFFFFFF00 | m);
        }
    });
```

Optimization Strategies

- Several procedures could be employed to further optimize Esoteric:
  - Idle Loop Detection for the SH-4 (short loops with no memory accesses – data within loop is ”created” within the loop it self)

```
dt R1  decrement and test
bfs <dt_address>  if false goto dt address/execute addi
addi R0 0x1  add 1 to R0
```

  - The above block would be executed R1 number of times till the value of R1 is 0.It could be replaced by adding to R0 0x1 multiplied by R1.
- Render To Texture and Pixel/Vertex shaders support , would take the burden of the graphical emulation of the Sega Dreamcast console out of the CPU.

Applications

- Esoteric in its current state,albeit not serving as a full featured Sega Dreamcast emulator ,can be used as:
  - A tool for developers, writing software for the Sega Dreamcast console.
  - As an applet to make opensource games and applications made originally for the Sega Dreamcast available also throught the World Wide Web
  - The SH-4 emulation is modular by design and its implementation can be used in other projects.
  - Deploy opensource software for the Sega Dreamcast as a Java Web Start bundle

Future Work

- Future development on Esoteric will focus on:
  - Adding sound support, through OpenAL
  - Increase the compatibily, through better and more complete emulation of the several devices present on a Sega Dreamcast gaming console.
  - Further optimizing the application
  - Add support for more devices which can be connected to the Sega Dreamcast through the Java API