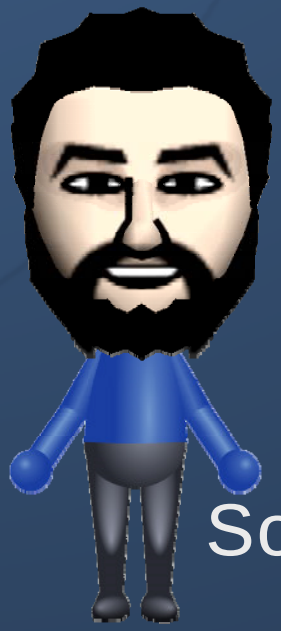
WiiProfiler v3.0

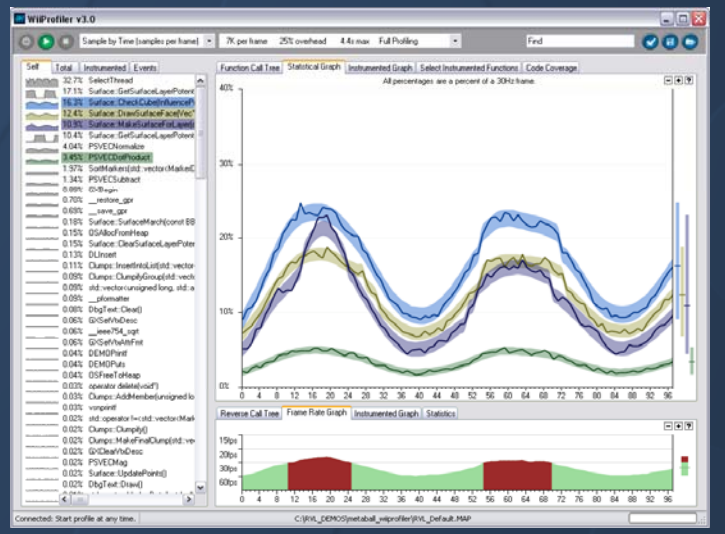


Steve Rabin
Principal Software Engineer

Software Development Support Group

Agenda

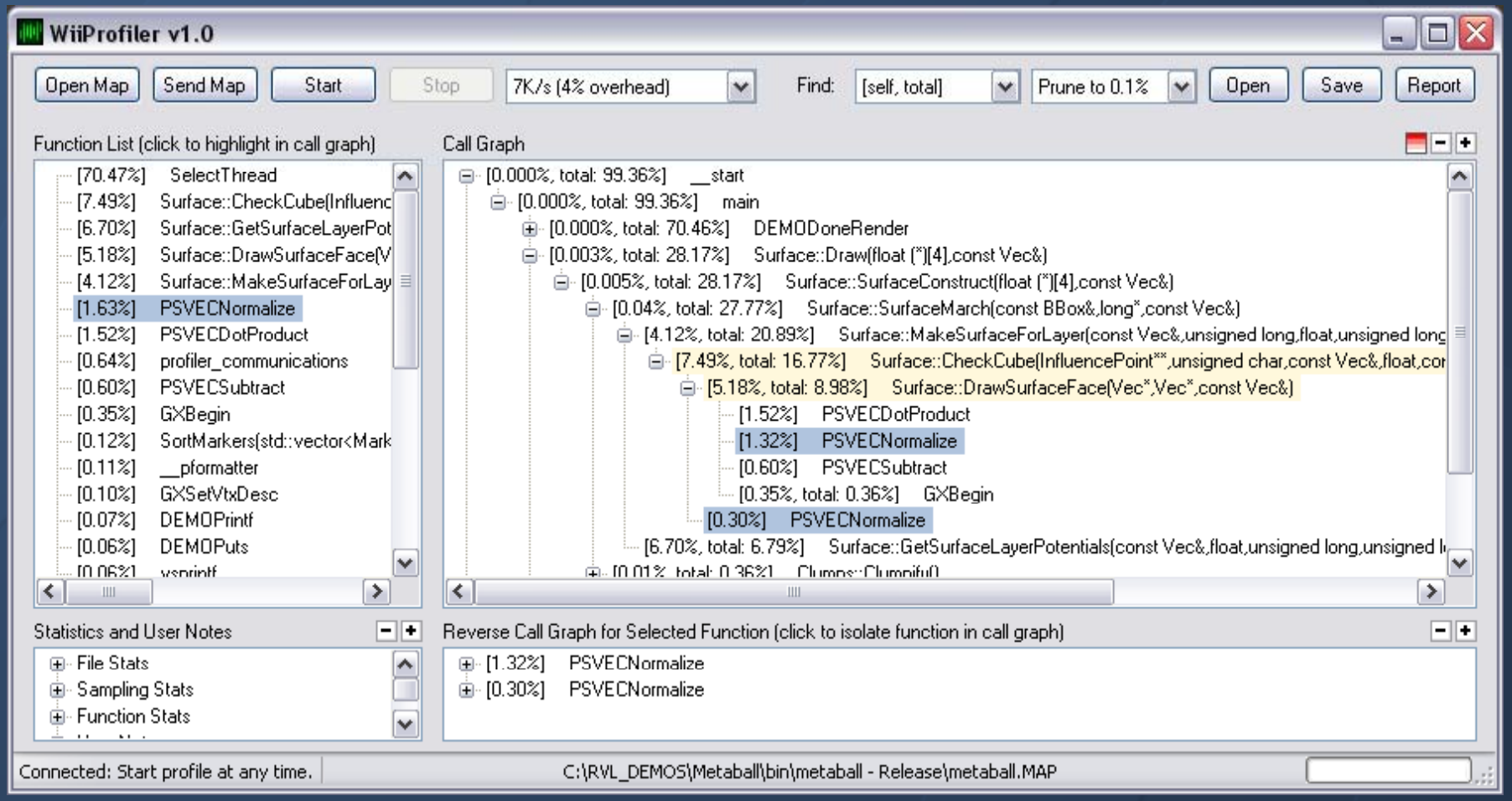
- WiiProfiler introduction
  - What it provides
- WiiProfiler Methodology
  - Game integration
  - V2.0 features
- WiiProfiler v3.0 features
  - Sampling based on performance counters
  - Instrumenting using performance counters
  - Tracking user data
  - Code coverage



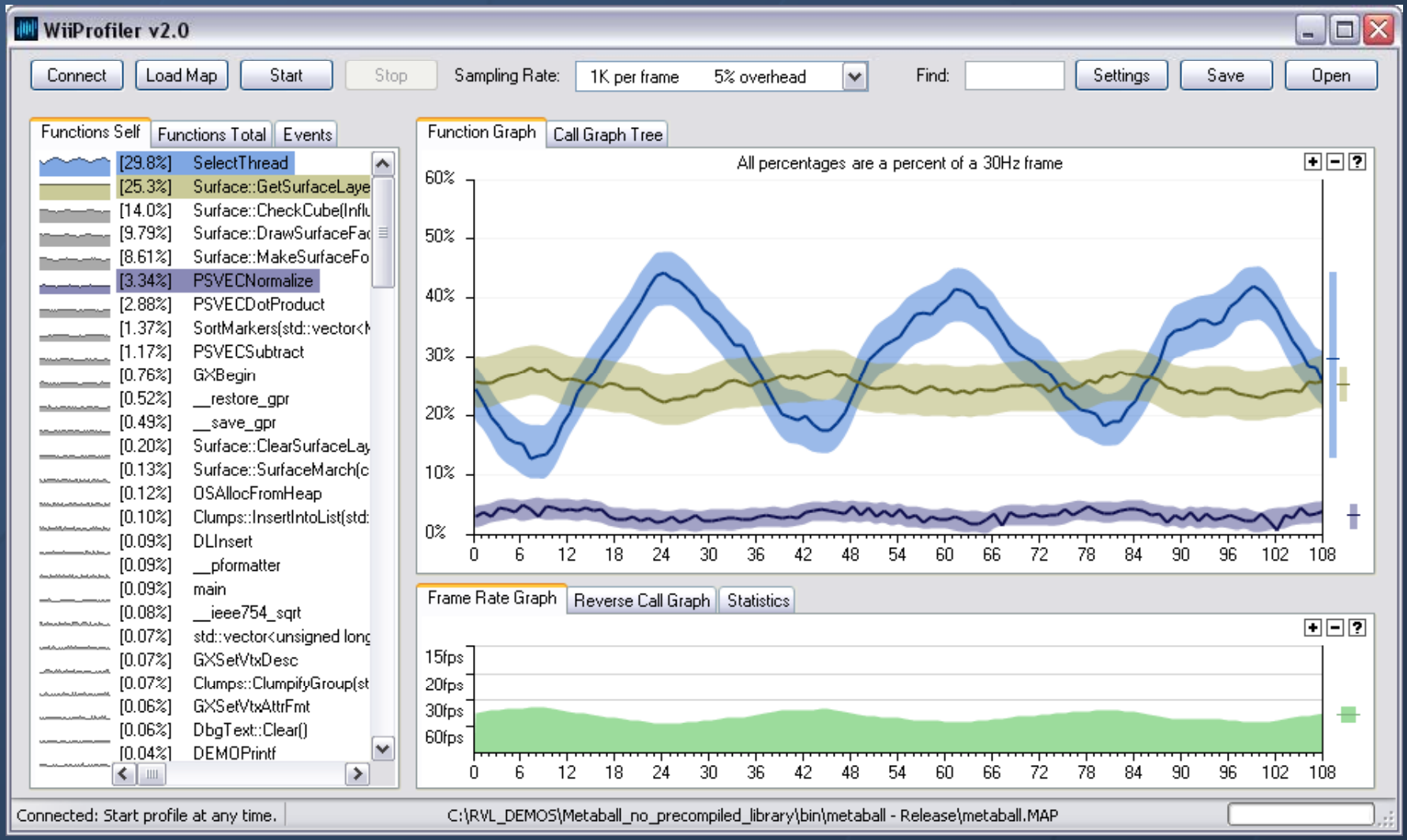
Introduction

- Measures CPU function performance
  - How much time spent in each function
  - Cycles, instructions, branches, cache misses
  - Function call tree
  - Function code coverage
  - Frame rate performance
- Free tool created exclusively for Wii
  - Version 1.0 (May 2007)
  - Version 2.0 (April 2008)
  - Version 3.0 (Open BETA now, Final Summer 2009)
- Requirements
  - NDEV and minor programmer integration

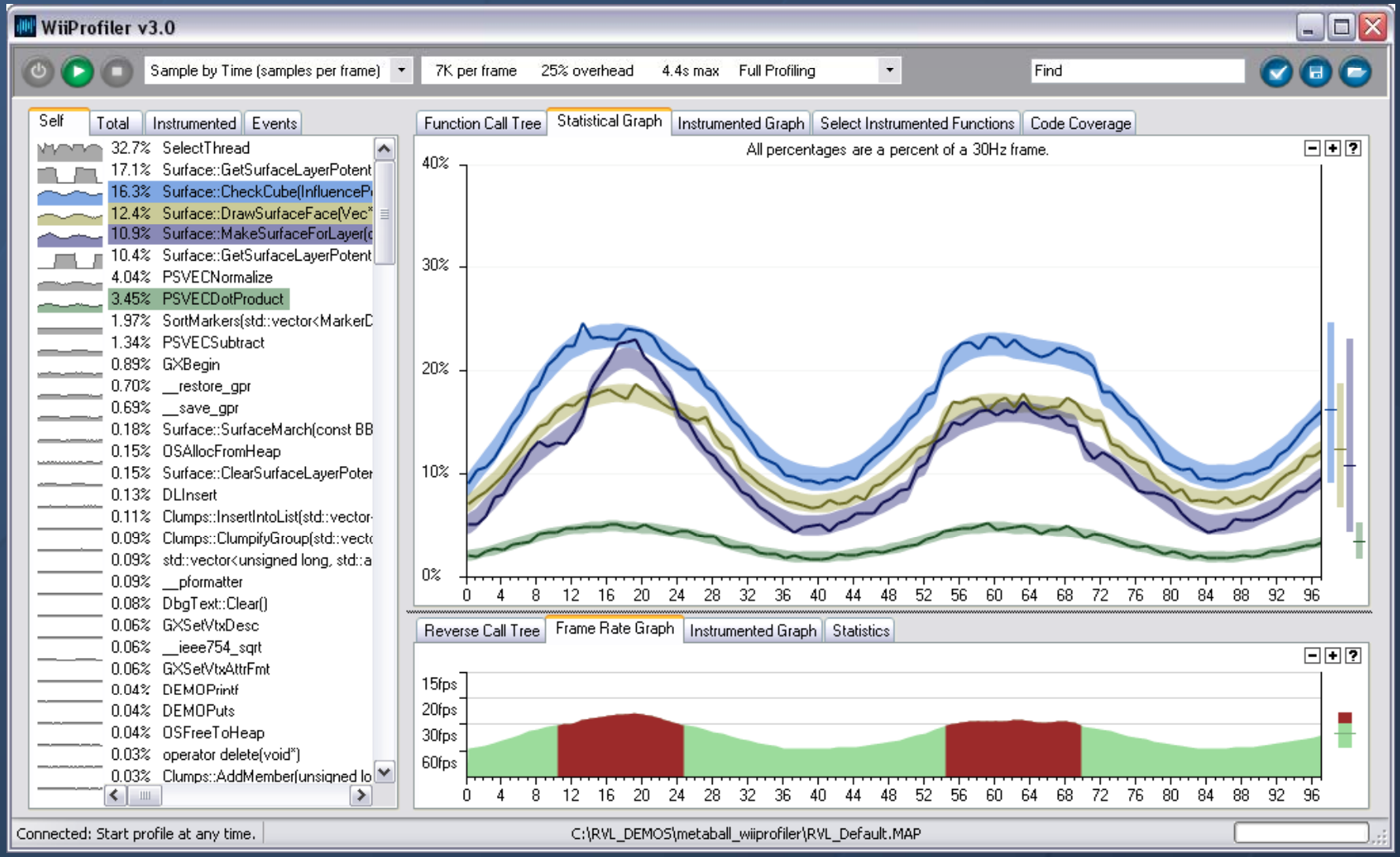
WiiProfiler v1.0



WiiProfiler v2.0



WiiProfiler v3.0



WiiProfiler Design Methodology

- Extremely fast and easy to integrate
  - Only a couple required function calls to library functions (10 minute integration)
- Extremely fast and easy to operate
  - Minimalist interface that just works
  - Deep functionality with little cognitive overhead
- Effortless visual exploration of data
  - Use graphs to maximize comprehension
  - Frame-based graphs show problem frames
  - Easy to compare and interpret

Methodology:
Fast and easy to integrate

Code Integration: Step 1

- Link against "wiiprofiler.a"

Code Integration: Step 2

Include the header file:

```
#include <revolution/wiiprofiler.h>
```

Code Integration: Step 3

```
WIIPROFILER_Init(void * bufferMEM2,u32 sizeInBytes,
                 BOOL doesGameWaitForRetrace);
```

- Call init function with a MEM2 buffer
  - At least 8MB, as large as 100MB
  - Larger buffer = longer profiling
- Answer the question:
  - Does your main loop wait for the vertical retrace?

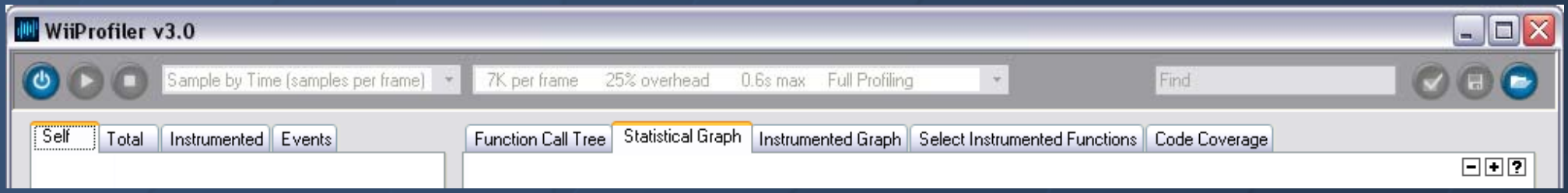
Code Integration: Step 4

```
while(true)
{  //Top of main loop

   WIIPROFILER_MarkFrameBegin();

   //Game code, etc.
}
```

Methodology:
Fast and easy to operate

Only Two Choices

- Connect to NDEV
- Open a profile



Demo:
Fast and Easy to Operate

- Statistical sampling
  - Various rates available, Simple vs Full
  - Accuracy vs Overhead/Size tradeoff
- Start and Stop

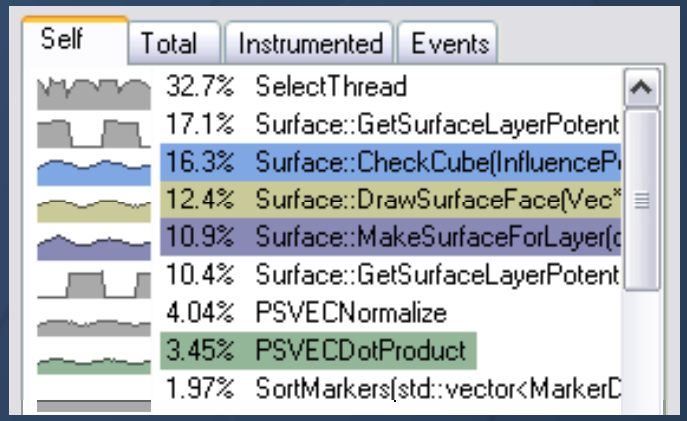

- Open and Save

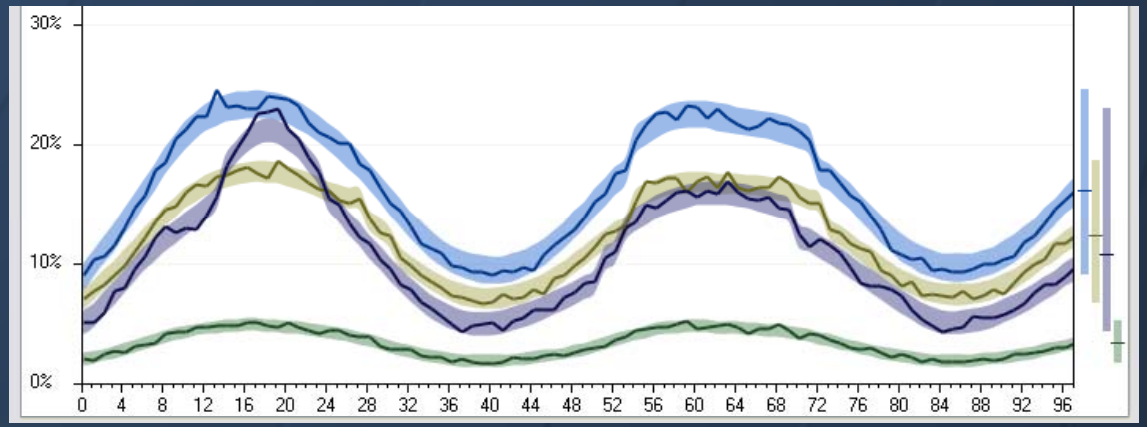

- Settings and right click menus

Methodology: Effortless Visualization

Demo: Effortless Visualization

- Functions
  - Sparklines
  - Self vs Total
  - Hide insignificant
- Call tree exploration
  - Reverse call tree
- Statistical graph
  - Click functions
  - Zoom, scroll, choose frame
  - Highlight Band
  - Range and average





Demo: Effortless Visualization

- Frame rate graph
  - Examine frame rate spikes
  - Events
- Resort functions (new in v3.0)
  - Sort based on selected frame
  - Sort based on average (default)
  - Sort alphabetically
  - Continuously resort

Performance Counter Factoid Theater

- 4 CPU performance counters in Broadway CPU
  - Reset, start, stop, and read in code
- Reset counters
  - PPCMtpmc1(0); PPCMtpmc2(0); PPCMtpmc3(0); PPCMtpmc4(0);
- Start counters
  - PPCMtmmcr0(<counter1> | <counter2>);
  - PPCMtmmcr1(<counter3> | <counter4>);
- Stop counters
  - PPCMtmmcr0(0);
  - PPCMtmmcr1(0);
- Read counters
  - PPCMfpmc1(); PPCMfpmc2(); PPCMfpmc3(); PPCMfpmc4();

Performance Counter Factoid Theater

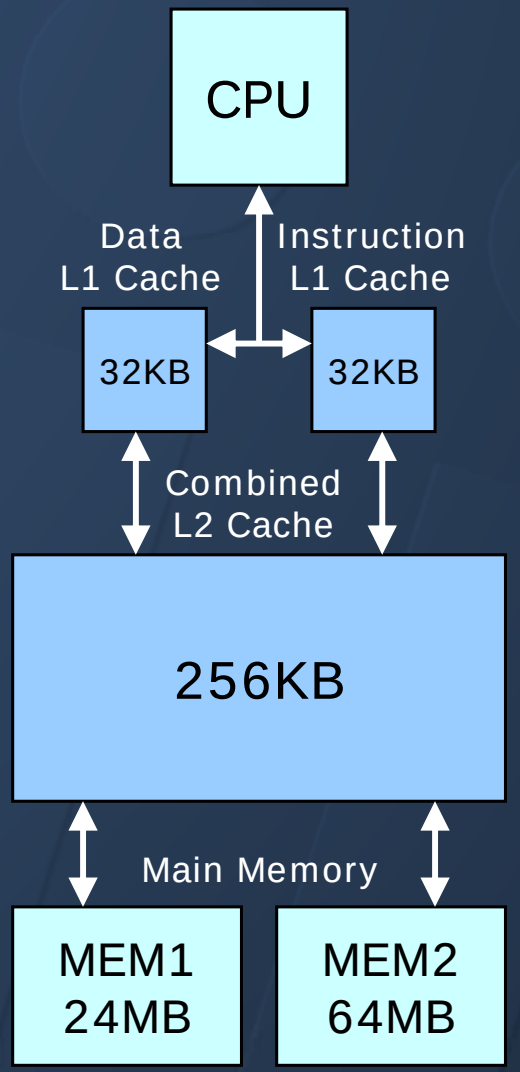
- Performance counter event examples (~60 total)
  - PMC1_CYCLE # processor cycles
  - PMC1_L2_HIT # of accesses that hit L2
  - PMC1_L1_MISS # of accesses that miss L1
  - PMC1_Bx_UNRESOLVED # of branches unresolved
  - PMC1_Bx_STALL_CYCLE # of cycles stalled due to branches
  - PMC2_CYCLE # processor cycles
  - PMC2_INSTRUCTION # of instructions completed
  - PMC2_IC_MISS # of L1 instruction cache misses
  - PMC2_L1_CASTOUT # of L1 castouts to L2
  - PMC2_Bx_FALL_THROUGH # of fall through branches
- Select one PMC1, PMC2, PMC3, PMC4 at a time
- Bracket code (Reset, Start, Stop) and measure results

Performance Counters in WiiProfiler v3.0

- Use performance counters to
  - Statistically sample functions
  - Instrument individual functions

Performance Counter Statistical Sampling

- Sample by
  - Mispredicted branches
  - Undecided branches
  - Floating point instructions
  - L1 or L2 instruction misses
  - L1 or L2 data misses
  - L1 writes to L2
  - L2 writes to memory



Performance Counter Statistical Sampling

- Choose a sampling rate
  - Between every 10 and every 100K
- Too often (every 10 to 100)
  - Large overhead
  - Can be less accurate (cache pollution)
  - Fills up buffer fast
- Often (every 100 to 1K)
  - Medium overhead
  - Good accuracy
- Less often (every 1K to 100K)
  - Least overhead
  - Most accurate overall (less accurate per frame)

Instrumenting Functions

- Choose a class of performance counters
  - Cycles only
  - Cycles and instructions
  - Branch prediction performance
  - Why branch prediction failed
  - Cache and memory performance
  - L1 cache performance
  - L2 cache performance
  - Outbound cache writes
- Explanation of selected in big gray box
- Decide:
  - Whether or not to also statistically sample by time

Instrumenting Functions: Branch Prediction Performance

- Performance counters selected
  - PMC1_Bx_UNRESOLVED
  - PMC2_Bx_FALL_THROUGH
  - PMC3_Bx_TAKEN
  - PMC4_Bx_MISSED
- Data teased out from these 4 counters
  - % of correctly predicted branches
  - % of incorrectly predicted branches
  - Correctly predicted branches
  - Incorrectly predicted branches
  - Skipped branches based on prediction
  - Taken branches based on prediction
  - Branches predicted by hardware
  - Branches unconditionally taken
  - All branches

Instrumenting Functions: L1 Cache Performance

- Performance counters selected
  - PMC1_L1_MISS
  - PMC2_IC_MISS
  - PMC3_DC_MISS
  - PMC4_CYCLE
- Data teased out from these 4 counters
  - Cycles
  - Cycles waiting for memory
  - Instruction not found in L1
  - Data not found in L1
  - Memory not found in L1
  - Average cycles waiting for memory
  - % of time waiting for memory

Instrumenting Functions: Selecting Functions

- Up to 10 functions profiled at a time
- 3 ways to select a function
  - Choose a Self or Total function
  - Drop down list of all game functions
  - Choose a function from Code Coverage
- Data captured is similar to "Total"
  - Function call and child calls

Instrumenting Functions: Profile and Explore

- # function calls tracked
- # recursive calls tracked
- Performance counters
  - Total count for performance counter
  - Range per frame (max, ave, min)
  - Raw call data (might graph slowly)

- Helpers
  - Expand top level
  - Auto-select similar

Tracking User Data

- Track any data you want in code
  - Track floating point values
- WIIPROFILER_TrackValue(name, value);
  - Will track multiple values per frame
- WIIPROFILER_TrackAccumulatedValue(name, value);
  - Will track one accumulated value per frame
- WiiProfiler on PC
  - Appears in Instrumented tab
  - Graphs in Instrumented Graph tab

Code Coverage

- During a profile (or over multiple)
  - Which functions get called
  - Which functions don't get called
- Filter
  - Exclude SDK and platform libraries
  - Exclude functions with certain prefixes
  - Include functions with certain prefixes
- Reset button
- Instrument button

WiiProfiler v3.0 Release

- Open BETA for next 1-2 months
  - Sign up and we'll send it to you:
    https://www.warioworld.com/wii/wiiprofiler
- Final release v3.0 early Summer
  - More robust communications layer
  - Instrumenting functions
    - Allow RSO and REL functions
    - Remove interrupts from data

WiiProfiler Summary

- Statistical sampling profiler
  - Time and performance counters
- Instrument functions
  - Using performance counters
- Track and graph arbitrary data
- Function-based code coverage

Questions?

Ask me after the presentation

Or e-mail support@noa.com