Practical game-programming

Overview

- Consoles in general
- PC/Console difference
- Development environments
- Memory management
- Memory fragmentation
- Multiplatform development
- Optimization
- TRC/LOT-Guidelines

Consoles in general

- XBox
  - **CPU**: 733 MHz chip (Pentium III Coppermine-based); 32 KB L1 cache, 128 KB on-die L2 cache
  - **GPU**: 233MHz custom chip (similar to NVidia Geforce 3)
  - **Memory**: 64 MB (shared memory)
  - **I/O**: 4x DVD, 10GB hard disk

Consoles in general

- PlayStation 2
  - **CPU**: 128-bit „Emotion Engine“ clocked at 294 MHz
    - Vector Units VU0 and VU1
    - 16 KB Instruction cache, 8 KB Data cache, 16 KB Scratchpad
  - **Graphics**: "Graphics Synthesizer" clocked at 147 MHz
  - **I/O Processor, SPU2, IPU**
  - **Memory**: 32 MB
  - **VRAM**: 4 MB (framebuffer, z-buffer, textures)
  - **I/O**: 4x DVD, no hard disk

Consoles in general

- Nintendo DS
  - **CPU**: Main ARM946E-S (67 MHz); Sub: ARM7TDMI (33 MHz)
  - **Graphics 2D**: BG: 4 Background-Layer per Display, OBJ: max. 128 Sprites
  - **Graphics 3D**: max. 120.000 polygons/second
  - **Memory**: 4 MB
  - **VRAM**: depends on which mode is used
  - **I/O**: cartridge, size varies

Consoles in general

- Wii
  - **CPU**: PowerPC-based „Broadway“
  - **Graphics**: ATI „Hollywood“
  - **Memory**: 24 MB „internal“ 1T-SRAM, 64 MB „external“ GDDR3 SDRAM
  - **VRAM**: can use internal and external, but speed varies
  - **I/O**: double-layer DVD

Consoles in general

- XBox 360
  - **CPU**: Triple-core PowerPC-based IBM Xenon
  - **Graphics**: ATI Xenos
  - **Memory**: 512 MB (unified memory)
  - **VRAM**: „10 MB eDRAM“
  - **I/O**: 12x DVD, hard disk optional

Consoles in general

- PlayStation 3
  - **CPU**: Cell
  - **Graphics**: NVidia RSX (based on G70)
  - **Memory**: 256 MB
  - **VRAM**: 256 MB
  - **I/O**: 2x Blu-ray, hard disk

PC/Console difference

- CPU/GPU/cache sizes differ
  - Know your hardware, use it efficiently
- Memory size differs greatly
  - Data, assets, and code size; everything plays a role
  - Amount of memory is carved out of stone; there is no virtual memory
  - OS/SDK often needs a lot of precious memory
- VRAM size differs greatly
  - New strategies needed, exploit hardware
  - Assets must cope with that as well

PC/Console difference

- Multithreaded/parallel programming essential for PS2/XBox360/PS3
  - EE, VU0, VU1, GS, IOP, IPU perform „independently“ on PS2, DMA acts as link
  - Entirely new algorithms needed for PS3 (concept of „SPU Shaders“)
  - still not mainstream in PC development
- Seek-times and bandwidths differ
  - Hard disk on PC, DVD on consoles
  - Cartridges on handhelds (Nintendo DS)
    - No seek times

PC/Console difference

- Some algorithms simply won't work out-of-the-box
  - (Predicated) Tiled Rendering on XBox360 -> Framebuffer doesn't fit into eDRAM
    - Render in tiles
    - Vertex shaders get executed for every tile -> Problems with memexport!
  - Many different strategies for known problems
    - E.g. there are more than 7 ways to skin a character
  - Moving stuff traditionally done on the GPU to the CPU/SPU (backface culling, PP effects)

Development environments

- Dev-kits
  - Most of them have more memory than the real system
    - Can be used for development/prototyping
  - Provide DVD emulation
    - Don't have to burn a CD/DVD everytime
  - May provide LAN connection for e.g. real-time editing or sharing amongst team-members
  - May have hardware bugs
  - May contain extra-hardware (e.g. Profiler)
  - Cost more than the real system

Development environments

- Different compilers
  - Mostly a specific GCC is used on consoles
  - First revisions may not at all be bug-free
    - Helps to understand assembly code to see what the compiler does
- Different compiler feature-sets
  - May not compile hardcore C++ features at all
  - May have problems with some features, e.g. templates
  - Work with different warning-levels, complaining about different things

Development environments

- SN Systems
  - Provides Windows development tools for consoles (PS2, PSP, GC, DS, PS3)
  - ProDG
    - Compiler
    - MS Visual Studio integration
    - Debugger
  - PS2/GC compiler based on older GCC (2.95.5)
  - DS compiler is a complete rewrite
  - Additional console-specific features (e.g. vector unit code for PS2, 128bit datatypes)

Development environments

- Console tools
  - Statistical profiler
  - Debugger
  - Console I/O
  - GFX profiler (PIX, Shader debugger, etc.)
  - PS2 Performance analyzer: captures the complete console status (DMA, GS, etc.)
    - Provides insight into inner workings, produces **a lot** of data
    - Needs experienced people only to understand the data

Memory management

- Crucial in console development
  - Remember: no virtual memory!
  - Xbox has MMU for "virtual memory", can't swap however
  - GameCube has "virtual memory", actually swaps into Sound-RAM
  - Main missing part is still a harddisk

Memory management

- Avoid allocations every frame
  - Shouldn't do that on the PC either, really
- Avoid small allocations, prefer large ones instead
  - Helps with fragmentation as well
  - Use own allocation scheme for small allocations you can't get rid of (e.g. STL, Strings)
- Small allocations
  - Use group allocations
  - Use freelists
  - Use memory pools

Memory management

- STL usage
  - Choose the right tool for the job
  - Try to reserve memory in advance, e.g. myVector.reserve(1000);
  - Is/can be used on consoles, although some people/studios roll their own containers
    - Don't do this before profiling!
  - Use hash tables if possible

Memory management

- STL usage
  - Choose the right tool for the job
  - Try to reserve memory in advance, e.g. myVector.reserve(1000);
  - Is/can be used on consoles, although some people/studios roll their own containers
    - Don't do this before profiling!
  - Use hash tables if possible

PROFILE!

Memory management

- You can never have enough memory
  - Try to reduce memory consumption at the algorithmic level, rather than brute-force optimization
  - Code size can get in your way too, make sure to study the compiler settings
  - Write your own memory dump
  - Make use of map files
  - Don't store what you won't need
  - Use member/method reordering as a last resort!

Memory management

- You can never have enough memory
  - Try to reduce memory consumption at the algorithmic level, rather than brute-force optimization
  - Code size can get in your way too, make sure to study the compiler settings
  - Write your own memory dump
  - Make use of map files
  - Don't store what you won't need
  - Use member/method reordering as a last resort!

PROFILE!

Memory fragmentation

- What is it?
  - Memory gets fragmented during run-time
  - E.g. A,B,C get allocated; A,C get freed; B leaves a gap (Bad Idea™)
  - Not something PC developers worry about
- How does it emerge?
  - Due to allocation/deallocation during run-time
  - Depends on order of allocations, can't be eliminated completely (only under specific circumstances)

Memory fragmentation

- Why is it bad?
  - Longer playtime means less memory
    - Not really less memory, but the largest available block gets smaller
  - Less memory suddenly becomes no memory
    - Largest available block might not be big enough anymore
  - No memory becomes (random) game crash
    - But the memory **would** be there
  - Random crashes can be **extremely** difficult to find

Memory fragmentation

- What to do against it?
  - Use memory pools
  - Use own allocation schemes
  - Avoid small allocations
  - Write your own heap implementation (different pools for small allocations, coalescing/defragmenting heap for other allocations)
  - Last resort: Use memory images (your team will end up in a sea of fire for that…)
  - Be clever!
    - Rearrangement of allocations can be a life saver

Multiplatform development

- Different APIs for development
  - DirectX on Xbox
  - More "down to the metal" on PS2
  - Library routines provided by Microsoft, Sony, Nintendo
  - Own large codebase for file-I/O, input-devices, memory tracking, etc.
- Different CPUs and GPUs
  - Capabilities differ, one platform might have more features than the other
  - One platform might outperform the other (CPU- or GPU-wise)

Multiplatform development

- Platform-specific features
  - Use them, but stay multiplatform (especially in high-level code)
- Up to 10 different builds
  - Hard to make sure that a build doesn't break when there just isn't time for rebuilding all of them
  - This is where an automated build system comes in handy
- Input devices differ
  - Mouse and keyboard vs. Wii-Mote

Multiplatform development

- Output devices differ
  - High-resolution monitor vs. Pal-TV
  - User-defined Hz vs. PAL/NTSC 50/60hz
  - Colors are different on PAL and NTSC systems
- Requires different assets
  - Remember the 4 MB VRam on the PS2?
    - Might be hard to fit XBox/PC-textures in there
  - Assets might be graphics, sounds, as well as code
- Requirements/guidelines differ (more on that later)

Optimization

- Most important rule:
  - Don't optimize without prior profiling!
    - No need to optimize your math-library when it's not going to be the bottleneck anyway
    - Optimization requires finding a bottleneck
  - Improving 20% of the code is responsible for 80% of the results (80/20 law, Pareto principle)
  - 90% of the execution time is spent executing 10% of the code (90/10 law)

Optimization

- Most important rule:
  - Don't optimize without prior profiling!
    - No need to optimize your math-library when it's not going to be the bottleneck anyway
    - Optimization requires finding a bottleneck
  - Improving 20% of the code is responsible for 80% of the results (80/20 law, Pareto principle)
  - 90% of the execution time is spent executing 10% of the code (90/10 law)

Optimization

- Only optimize when really necessary
  - Try to optimize at the algorithmic level first
    - O(n) will almost always outperform O(n^x)
  - Try to help the compiler when he can't optimize (e.g. because of aliasing)
  - Use assembly language and know your hardware!
    - Changes will get tedious (Assembly -> High-level language -> Assembly)
    - But a tremendous speed-up is possible

TRC/LOT-Guidelines

- Games must fulfill guidelines for the respective platform(s)
- These guidelines come from the console-manufacturer (Microsoft, Sony, Nintendo)
- Contain requirements e.g. about
  - button handling in menus
  - areas where to display graphics on the screen
  - safety-messages
- Can be tedious to implement
  - Must be implemented 100%, otherwise you're simply not allowed to ship your game!

TRC/LOT-Guidelines

- Are sometimes not that easy to understand
  - Might be contradictory
- Every game must go through platform-QA
  - Can't develop whatever you want on consoles, must first get approval for this
- Approval process
  - Concept, game design
  - Technical requirements
  - Several stages to go through

Questions ?