Game Engine Architecture

Chapter 2

Tools of the Trade

Hooman Salamat

Version Control

- A *version control* system is a tool that permits multiple users to work on a group of files collectively.
- It is sometimes called *source control*
  - provides a central repository from which engineers can share source code;
  - keeps a history of the changes made to each source file;
  - provides mechanisms allowing specific versions of the code base to be tagged and later retrieved; and
  - permits versions of the code to be branched off from the main development line, a feature often used to produce demos or make patches to older versions of the software.

Common Version Control Systems

- *SCCS and RCS*. The Source Code Control System (SCCS) and the Revision Control System (RCS
- The Concurrent Version System (CVS)
- *Subversion*. an open source version control system aimed at replacing and improving upon CVS.
- *Git*. an open source revision control system
- *Perforce*. Perforce is a professional-grade source control system, with both text-based and GUI interfaces (used by many game companies, including *Naughty Dog* and *Electronic Arts*..)
- *NxN Alienbrain*. Alienbrain is a powerful and feature-rich source control system designed explicitly for the game industry.
- *ClearCase*. Rational ClearCase is a professional-grade source control system
- *Microsoft* Team Foundation **Version Control** for centralized version control.

Compilers, Linkers and IDEs

- Source Files, Headers and Translation Units
- Libraries, Executables and Dynamic Link Libraries
- Projects and Solutions
- Build Configurations
- Common Build Options
  - *Preprocessor Settings*
  - *Compiler Settings*
  - *Linker Settings*
- Local and Global Optimizations

Typical Build Configurations

- *Debug*
- *Development*
- *Ship*
- *Hybrid Builds*
  - A hybrid build is a build configuration in which the majority of the translation units are built in development mode, but a small subset of them is built in debug mode.

Project Configuration

- Right-clicking on any project in the Solution Explorer and selecting “Properties…” from the menu brings up the project’s “Property Pages” dialog.
  - Configuration Properties/General,
  - Configuration Properties/Debugging,
  - Configuration Properties/C++, and
  - Configuration Properties/Linker.

Macros

- $(TargetFileName). The name of the final executable, library or DLL file being built by this project.
- $(TargetPath). The full path of the folder containing the final executable, library or DLL.
- $(ConfigurationName). The name of the build config, which will be “Debug” or “Release” by default in Visual Studio, although as we’ve said, a real game project will probably have multiple configurations such

as “Debug,” “Hybrid,” “Development” and “Ship.”

- $(OutDir). The value of the “Output Directory” field specified in this dialog.
- $(IntDir). The value of the “Intermediate Directory” field in this dialog.
- $(VCInstallDir). The directory in which Visual Studio’s C++ standard library is currently installed.

Debugging Property Page

- *C/C++ Property Page*
  - *General Property Page/Additional Include Directories.*
  - *General Property Page/Debug Information Format.*
  - *Preprocessor Property Page/Preprocessor Definitions.*
- *Linker Property Page*
  - *General Property Page/Output File.*
  - *General Property Page/Additional Library Directories.*
  - *Input Property Page/Additional Dependencies.*

Debugging Your Code

- The Call Stack
- The Watch Window
  - The ",d" suffix forces values to be displayed in decimal notation.
  - The ",x" suffix forces values to be displayed in hexadecimal notation.
  - The ",*n*" suffix (where *n* is any positive integer) forces Visual Studio to treat the value as an array with *n* elements.
- Data Breakpoints
  - set a breakpoint that trips whenever a specific memory address is *written to*. Bring up the "Breakpoints" window found on the "Debug" menu under "Windows" and then "Breakpoints"
    - Select the "New" drop-down button in the upper-left corner of the window.
    - Select "New Data Breakpoint."
    - Type in the raw address or an address-valued expression, such as "&myVariable"

Conditional Breakpoints

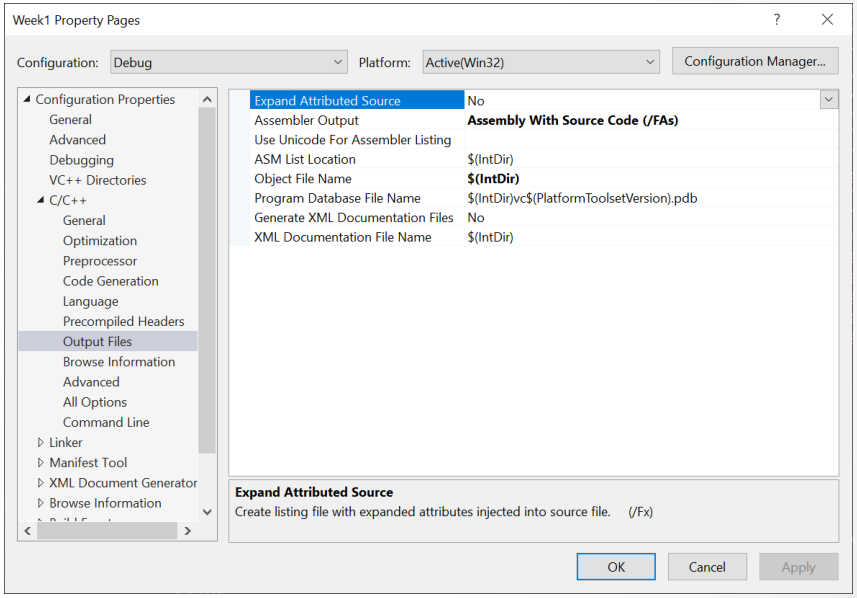
- set conditions on any type breakpoint—data breakpoints or regular line-of-code breakpoints.
- Example: game level with 20 tanks on-screen, and you want to stop your program when the third tank, whose memory address == 0x12345678, is running.
- By setting the breakpoint's condition expression to
- something like "(uintptr_t)this == 0x12345678", you can restrict the breakpoint only to the class instance whose memory address (this pointer) is 0x12345678.
- Conditional breakpoints cause the debugger to evaluate the conditional expression every time the breakpoint is hit, so they can bog down the performance of the debugger and your game.

Hit count

- Specifying a *hit count* for a breakpoint causes the debugger to decrement a counter every time the breakpoint is hit
- and only actually stop the program when that counter reaches zero!
- inspect what's happening during the 100th iteration of the loop (e.g., the 100th element in an array).
- Don't hit the F5 key 100 times!

Debugging Optimized Builds

- *Learn to read and step through disassembly in the debugger.*
  - You can normally see assembly code while debugging C++ in visual studio For this in Visual Studio put a breakpoint on code in question and when debugger hits it rigth click and find "Go To Disassembly" (or press CTRL+ALT+D)
  - Second approach is to generate assembly listings while compiling. For this go to project settings -> C/C++ -> Output Files -> ASM List Location and fill in file name. Also select "Assembly Output" to "Assembly With Source Code". The file *.asm will be saved in the same location as $(IntDir)



Debugging Optimized Builds

- *Use registers to deduce variables' values or addresses.*
  - The debugger will sometimes be unable to display the value of a variable
  - trace back through the disassembly to where the variable is first loaded into a register

```
ShaderInfo shaders[] = {
{ GL_VERTEX_SHADER,
"triangles.vert" },
{ GL_FRAGMENT_SHADER,
"triangles.frag" },
{ GL_NONE, NULL }
};
```

```
ShaderInfo shaders[] = {
{ GL_VERTEX_SHADER, "triangles.vert" },
00945CC2  mov         dword ptr [shaders],8B31h
00945CC9  mov         dword ptr [ebp-28h],offset
string "triangles.vert" (094FDA4h)
00945CD0  xor         eax,eax
00945CD2  mov         dword ptr [ebp-24h],eax
{ GL_FRAGMENT_SHADER, "triangles.frag" },
00945CD5  mov         dword ptr [ebp-20h],8B30h
00945CDC  mov         dword ptr [ebp-
1Ch],offset string "triangles.frag" (094FDB8h)
00945CE3  xor         eax,eax
00945CE5  mov         dword ptr [ebp-18h],eax
{ GL_NONE, NULL }
00945CE8  mov         dword ptr [ebp-14h],0
00945CEF  mov         dword ptr [ebp-10h],0
00945CF6  xor         eax,eax
00945CF8  mov         dword ptr [ebp-0Ch],eax
};
```

Debugging Optimized Builds

- *Inspect variables and object contents by address*
  - if an instance of the Foo class resides at address 0x1378A0C0,
  - We can type "(Foo*)0x1378A0C0" in a watch window,
  - and the debugger will interpret that memory address as if it were a pointer to a Foo object.
- *Leverage static and global variables*
  - Even in an optimized build, the debugger can usually inspect global and static variables.
  - Example: find the address of an internal object within the physics system,
  - discover that it is in fact stored in a member variable of the global PhysicsWorld object.

Profiling Tools

- *Pareto principle: 80/20 rule*, because it states that in many situations, 80% of the effects of some event come from only 20% of the possible causes
- 80% of the perceived bugs in a piece of software can be eliminated by fixing bugs in only 20% of the code
- how do you know *which* 20% of your code to optimize?
- A profiler is a tool that measures the execution time of your code.
  - how many *times* each function is called

Profilers Categories

- *Statistical profilers.*
  - the target code runs at almost the same speed, whether or not profiling is enabled.
  - sampling the CPU's program counter register periodically and noting which function is currently running.
  - Example: Intel's *VTune*
- *Instrumenting profilers*
  - The most accurate and comprehensive timing data possible, but at the expense of real-time execution of the target program
  - These profilers work by preprocessing your executable and inserting special prologue and epilogue code into every function.
  - Example: IBM's Rational Quantify

LOP

- low-overhead profiler by Microsoft
- is a hybrid between the two approaches.
  - uses a statistical approach
    - sampling the state of the processor periodically, which means it has a low impact on the speed of the program's execution
  - it analyzes the call stack, thereby determining the chain of parent functions that resulted in each sample.
- On the PlayStation 4, SN Systems' *Razor CPU* is the profiler of choice for measuring game software running on the PS4's CPU
  - Support both statistical and instrumenting profilers
- https://en.wikipedia.org/wiki/List_of_performance_analysis_tools

Memory Leak and Corruption Detection

- Blame for both of these problems falls squarely on the language feature known as the *pointer*.
- A memory leak occurs when memory is allocated but never freed.
  - leads to a potentially fatal out-of-memory condition.
- Memory corruption occurs when the program inadvertently writes data to the wrong memory location
  - overwriting the important data that was there
- IBM's Rational Purify:
  - instruments your code prior to running it, in order to hook into all pointer dereferences and all memory allocations and deallocations made by your code.

Other Tools

- *Difference tools:*
  - A difference tool, or *diff tool*, is a program that compares two versions of a text file and determines what has changed between them. http://en.wikipedia.org/wiki/Diff
- *Three-way merge tools:*
  - When two people edit the same file, two independent sets of diffs are generated. A tool that can merge two sets of diffs into a final version of the file that contains both person's changes is called a three-way merge tool. https://en.wikipedia.org/wiki/Merge_(version_control)#Three-way_merge
- *Hex editors:*
  - A hex editor is a program used for inspecting and modifying the contents of binary files.
  - The data are usually displayed as integers in hexadecimal format
  - tracking down problems with binary file formats or when reverse-engineering an unknown binary format
  - https://hexed.it/