DConf 2016, Berlin

Ethan Watson, Senior Generalist Programmer


QUANTUM BREAK
AAA GAMING WITH SOME D CODE
REMEDY

WHAT THIS TALK WILL COVER

- Integrating D
- A major use case
- Getting it shipped
- Where to for the future?

QUANTUM BREAK

WHAT IS IT?

- Third person cinematic action game with integrated live action TV show
- Xbox One, Windows 10
- #1 selling game on week of release in 8 countries including UK, Italy, France, and Switzerland
- Biggest selling new Microsoft IP this console generation


INTEGRATING D

PREVIOUSLY AT DCONF 2013

USING D ALONGSIDE A GAME ENGINE

CODE AS DATA

DYNAMIC BINDING BETWEEN C++ AND D

- Code as data
  - D code exists as data in our pipeline, allowing new logic to be shipped out without creating a new build
- Export functions between C++ and D
  - Use D language features to avoid the pain that doing the same in C++ would introduce
- Reloading code for rapid iteration
  - Compile time code inspection and generation to serialise and deserialise D objects

D'S COMPILE TIME FEATURES

WHAT DID I GET MYSELF IN TO?

VERSIONING

BECAUSE CODE AND DATA NEVER MATCH NICELY

- Mark up functions/interfaces with version numbers
  - Variable inside @(Export) and @(Import) UDA in D, #define parameter in C++
  - Matching versions has the nice benefit of solving Windows DLL Hell

VERSIONING

BECAUSE CODE AND DATA NEVER MATCH NICELY

- Stagger submitting D code until new build is published
  - Shelve D code
  - Submit C++ code
  - Email team
  - Publish C++ code
  - Submit D code
- Pain for programmers, seamless for everyone else
  - (Well, mostly seamless, still requires a data sync with a new build which isn't always done)
  - Not a problem with Unity/Unreal thanks to going whole hog with treating code as data
  - There is a better solution to be found for our needs

BINARY COMPATIBILITY

IT JUST WORKS!

```
@( Version( 3 ) ) struct DebugGraph
{
    @( AddedVersion( 2 ) ) Vector2 m_vTopLeft;
    @( AddedVersion( 3 ) ) Vector2 m_vBottomRight;

    @( Import ) void start( const( char )* pLabel, ref const( Vector2 ) vTopLeft, ref const( Vector2 ) vBottomRight );
    @( Import ) void plot( const( Vector2 )* pPoints, int iPointCount, ref const( Color ) color );

    final void start( string label, ref const( Vector2 ) vTopLeft, ref const( Vector2 ) vBottomRight ) { ... }
    final void start( in Vector2[] points, ref const( Color ) color ) { ... }
}

class SomeObject
{
    version( SomeObjectDebug ) DebugGraph graph;

    mixin ExportClass;
}
```

BINARY COMPATIBILITY

IT JUST WORKS!

```
class DebugGraph
{
    Vector2 m_vTopLeft;
    Vector2 m_vBottomRight;

    enum { Version = 3, RequiredVersion = 2 };

    OSP_BEGIN( DebugGraph, Version, RequiredVersion )
        OSP_VARIABLE( m_vTopLeft, 2 );
        OSP_VARIABLE( m_vBottomRight, 3 );
    OSP_END;
};

OSP_DEFINE( DebugGraph );
```

PLUGINS AND BINDINGS FOR RAPID ITERATION

or MAINTENANCE SUCKS

- Creating bindings to do code in D – 30+ minutes and the staggered submit hassle
- Doing the same code in C++ - 5 minutes and everyone can just wait for the new build
- Catch 22!
- A problem with the plugins/binding system, not the D language itself
  - (But we can solve some of the problems with D by using a compile-time parser that reads C++ header files and auto-generates binary compatible structs and bindings for us…)

A MAJOR USE CASE

ANIMATION – NO ONE MOVES WITHOUT IT

LIFE BEFORE D

- Our Morpheme setup required code and data to be in sync
- Code publish time of a day or more is bad

ANIMATION – NO ONE MOVES WITHOUT IT

D + PLUGINS = WIN!

- D plugins? Well, they solve some problems!
  - Generic component system written, specialised for animation networks
  - Manually go through and port C++ code to D components
  - Roll out the system, keep C++ code for fallback, then nuke the C++ code from orbit
  - Submitting new D code with new data became a regular occurance, and minor maintenance of code logic didn't need a whole new build. Win!

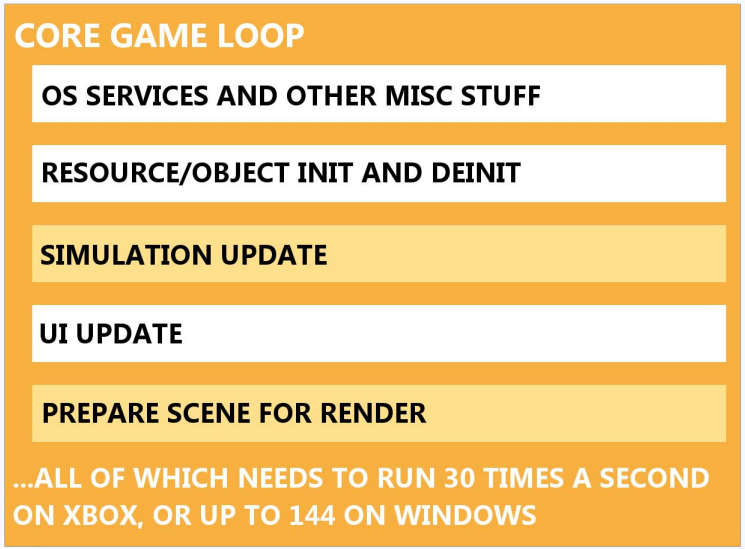

GETTING IT SHIPPED

- core.stdc.stdlib alloc/calloc/realloc/free
  - gc_rawAlloc, gc_rawCalloc, gc_rawRealloc, gc_rawFree
  - Hooking up our engine's allocation functions required staggering DLL initialisation

```
extern( Windows ) BOOL DllMain( HINSTANCE hInstance, ULONG ulReason, LPVOID pvReserved )
{
    final switch( ulReason )
    {
    case DLL_PROCESS_ATTACH:
        g_hInstance = hInstance;
        break;
    case DLL_THREAD_ATTACH:
        // Regularly called before the Setup function with multiple threads active!
    }
}

export extern( Windows ) void Setup( AllocFunc allocMem, CAllocFunc callocMem, ReallocFunc reallocMem, FreeFunc freeMem )
{
    setAllocFunctions( allocMem, callocMem, reallocMem, freeMem );
    dll_process_attach( g_hInstance, true );
}
```

- The GC itself wasn't "solved"
  - Far stricter memory requirements than normal programs
  - Industry standard is to have clear construction and destruction phases and budget time accordingly



- The GC itself wasn't "solved"
  - Automatic Reference Counting is our preferred method
    - Attempted to add compiler frontend support myself
    - Wasn't confident that I caught everything, put it to the side

- The GC itself wasn't "solved"
  - GC has 32MB, never collects, increments in 8MB chunks
    - This is quite clearly rubbish and needs a proper solution

SHIPPING WITH D

UNIVERSAL WINDOWS PLATFORM REQUIREMENTS

- Runtime porting still needs work
- LoadLibrary -> LoadPackagedLibrary
  - Auto-packaged files as data only work from deployment project FFFFFFFFFFFFFFFFFFFFFF
- std.datetime reghacks instead of using Windows APIs
- core.sys.windows.threadaux needs a new implementation
  - We didn't do it, but we get by without it


QUANTUM BREAK
XBOX ONE & WINDOWS 10
FIRST AAA GAME WITH D CODE TO SHIP ON XBOX ONE AND WINDOWS 10

WAS IT WORTH IT?
LOOKING TO THE FUTURE

- Wasn't used enough, could have done it in C++
  - More the fault of the plugin system, statically linking code would have been far simpler
- For the future though? High amount of interest
  - Natural threading boundaries make a task-based system safer to implement
    - No solid way to enforce boundaries in C++
  - AI wants to "script" behaviours with it, code gen around it to fit in to frameworks
  - Render effects in D
  - Speaking of scripting...
    - Internal scripting language used by level designers
    - Lacks modern features and no debugger
    - Why have a scripting language at all when we already treat code as data?

D FOR AAA GAMING
IS IT READY?

- Almost!
- Few areas that need tightening up
  - ARC support please please please please please
- “Official” console support (PS4, Xbox One)
  - PS4 especially is critical for AAA gaming
- Single instance of D runtime for entire application would be very beneficial
- Open sourcing our binding system?

QUESTIONS?