Game Developers Conference™ Europe 2011
August 15-17, 2011 | Cologne, Germany
www.GDCEurope.com

Handling Many Platforms with a Small Development Team

Dietmar Hauser
Head of Console Technology, Sproing

About Sproing

- Based in Vienna, Austria
- Work for hire on all platforms
- Independent for over 10 years!
- More than 50 titles shipped!

sproing

PS3 PSP PSVITA XBOX 360

Wii NINTENDO DS. NINTENDO 3DS PC CD-ROM

What the publisher wants

- An awesome game that sells very well
- Sell it on as many platforms as possible
- Get it delivered on time or earlier
- Spend as little money as possible

What we have

- A small team
  - 1-n Designers, Artists, Producers
  - 1-10 Programmers
- Many platforms
  - 2-n Consoles / PC / Phones / Tablets…
- A tight schedule
  - And all platforms finished at the same time

Possible Solutions

- Spend more money
  - By extending time and/or growing team
  - Very risky for everyone involved
  - Results in death marches and bankruptcies
- Develop all platforms simultaneously
  - Pretty difficult, but possible
  - Everything needs to be shared between platforms as much as possible

Challenges

- Different platform capabilities
  - Several asset fidelities required
  - Gameplay differences (i.e. controls)
  - Code performance differences
- Different APIs
  - No good standards (not even POSIX)
  - „Close to the metal“ APIs

Sharing Code / Abstraction

- Share only what can be shared well
- Don‘t force abstraction
  - Can get too complicated
  - Hides actual behaviour
  - Hides bugs / performance problems
- Seperate what doesn‘t fit together

Examples

- Easy to abstract
  - File I/O, Memory, Network, Threading,…
  - Textures, Meshes,…
- Tricky, but possible
  - Rendering, Controls,…
  - Save Data, DLC,…
- Very tricky, almost impossible
  - Peripherals like Kinect, Move, Wiimote,…
  - Unique platform features

Ways to abstract code

- Preprocessor directives
  - Most obvious and basic way
  - Unused code is discarded
  - Hard to read → Error prone

```
#if PLATFORM_A
        DoStuffPlatformA();
#elif PLATFORM_B
        DoStuffPlatformB();
#else
        #error Unsupported Platform
#endif
```

Ways to abstract code

- Pointer to Implementation (PIMPL)
  - Common, clean and useful
  - A lot to type and virtual function call overhead

```
class Foo {
        void DoIt() { impl->DoIt(); }
        FooImpl* impl;
};

class FooImpl {
        virtual void DoIt() = 0;
};

#if PLATFORM_A
class FooImplA {
        virtual void DoIt() { DoItPlatformA(); }
};
#endif
```

Ways to abstract code

- Templated PIMPL
  - Similar functionality to classic PIMPL
  - Less to type, less overhead

```
template <class FooImpl>
class FooBase {
        void DoIt() { impl->DoIt(); }
        FooImpl* impl;
};

#if PLATFORM_A
class FooImplA {
        void DoIt() { DoItPlatformA(); }
};

typedef FooBase<FooImplA> Foo;
#endif
```

Ways to abstract code

- Master header files
  - Platform specifics can be exposed
  - Least to type, zero overhead
  - Danger of code multiplication

```
// Foo.h
#if PLATFORM_A
        #include „FooA.h“
#elif PLATFORM_B
        #include „FooB.h“
#else
        #error Unsupported Platform
#endif
```

Ways to abstract code

- There are many many other ways…
  - .inl files
  - Layered abstractions
- No best way, so mix and match
- Don‘t be afraid to change it while you can

Compiling & Linking

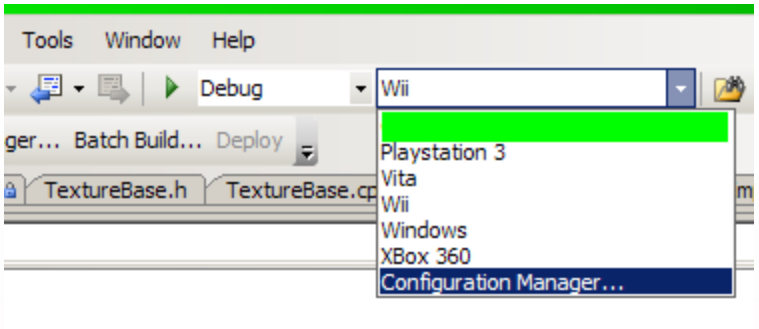
- Each platform has a different toolchain
  - Luckily, they‘re all command line tools
  - Accept the same source code (mostly)
  - Need different parameters
  - Print different diagnostics
- Some platforms have a VS integration
  - Which is usually pretty bad…

Compiling & Linking

- Use a build tool
  - Make, omake, jam, ant, scons,…
- Or roll your own VS integration
  - This is what we did…
  - … and it‘s simpler than you might think

Introducing ClWarrior

- Lets Visual Studio compile all platforms
  - Replaces cl.exe, link.exe and lib.exe
  - Translates arguments
  - Translates diagnostics



- Very straightforward for new employees
  - Code on windows
  - Change platform to anything
  - Build!

CIWarrior Challenges

- Visual Studio dependency check fails
  - Because it relies on a proprietary file (.idb)
  - So we wrote our own (it‘s pretty simple)
  - And hooked it up using an Add In
- Adding new platforms is not supported
  - Except for smart phones…
  - „WCE.VCPlatform.config“ can be hijacked
- Debugging still needs the platform tool
  - Unfortunate, but acceptable

Keeping the code alive

- There are a lot of code configurations
  - We use Debug, Release and Master
  - With 4 platforms that‘s 12 builds
- Checking all of them is difficult
  - Build is broken very often
  - Everyone gets frustrated
  - Code commits slow down

Solutions

- Continous Integration (CI)
  - All code is built all the time
  - Broken builds get reported immediately
  - And hopefully fixed immediately
- Unit Tests
  - Can be run automatically
  - Can detect regressions
  - Are very useful when porting to new platforms

More Solutions

- Static Code Analysis
  - Can also be run automatically
  - Can detect a lot of potential and real bugs
  - But can be difficult to set up
- Production QA
  - Ensures quality during production
  - Problems are uncovered at an early
  - Can prepare for final QA
  - Usually an overall cost reduction

Assets

- Dependent on the platform combination
- Start with the highest fidelity assets
  - Scaling down is easier than scaling up
- Split assets at the latest possible time
  - Most changes affect all versions
- Leverage automated downscaling
  - Textures are obvious candidates

Asset Conversion

- Automate it!
  - Everything is else is too error prone!
  - The build tools mentioned earlier can help
  - We use a selfmade rule based system
- Speed it up!
  - To decrease iteration time
  - Easiest way is to do it on one machine and distribute it to everyone else‘s

Surviving Certification

- Learn, implement and test the requirements early
- Make sure everyone in the team is aware of the requirements
  - Most of them are actually NOT technical
- Take advantage of pre-cert passes

Certification Tips

- Never stall the render thread
  - File I/O is usually to blame
- Make sure your game can be paused at any time
  - And resumed, of course…
- Watch out for memory fragmentation
  - Everything will work ok until the end

This is it...

Questions?