BUILDING AND LEVERAGING A CROSS PLATFORM VFX/ANIMATION DEVELOPMENT ENVIRONMENT

by Colin Doncaster (colin@peregrinelabs.com)
INTRODUCTION AND BRIEF OVERVIEW OF THE TALK

- making life easy via a simple concept
- controlled environments
- cross platform build management
WHAT DOES CROSS PLATFORM MEAN IN THE CONTEXT OF THIS TALK?

• Multiple OS platforms (Linux, Windows, OSX)
• Multiple Third Party applications (Maya, Houdini, Nuke & myriad of renderers)
• Cross platform Shaders and Shader DSO’s (Arnold c++, Renderman DSO’s etc.)
• Cross Platform APIs (Boost, Qt, etc.)
YOUR ENVIRONMENT

• where you work ( home, office, on set, traveling )
• what you’re working on ( Windows, Linux, Mac )
• the current state ( variables, installation locations, versions )
• production requirements
FORESHADOWING (IE. PROBLEMS WITH UNCONTROLLED ENVIRONMENTS)

- versions of applications
  - production vs. testing betas
  - staging new builds
- R&D
- dependencies
- deployment
- per show, sequence and shot dependencies (and how to stage control over these)
HOW CAN THIS BE MANAGED?
( ONE PERSONS OPINION )

- always start fresh
- use the shell! ( or at least wrap it up nicely )
- break down each chunk of information into manageable pieces
- pick your weapon ( python )
- make it a requirement to work this way
BUILD THE FOUNDATION
(REQUIRED CHEESY SLIDE)

• a common means of resolving non cross platform requirements

• defining each applications environment needs ( packages )

• resolving dependencies

• easy to execute
NON CROSS PLATFORM CONSIDERATIONS

- make decisions on how to represent non-cross platform requirements
  - base directories
  - users
  - system libraries, shared libraries etc.
- build a myStudioCrossPlatform.py library which resolves directory locations and wraps up global environment variables that lets you do:

  ```python
  myPath = myStudioCrossPlatform.getPluginsDirectory()
  ```
PACKAGES

• a package contains a description of environment needs
  • platforms supported (Windows, OSX and Linux)
  • the version this represents
  • environment variables (path to binaries, libraries, etc.)
  • dependencies and their specific environment needs
    (unique to this package)
RESOLVE DEPENDENCIES

• each package should list dependencies and/or define how to behave in another package has been requested
• dependencies can also include other environment variables (base directories etc.)
• both dependencies and environment variables should be “resolved” and unrolled for a clean environment
EXAMPLE PACKAGE

```json
{
  'tool': 'nuke',
  'version': '6.3',
  'platforms': [ 'windows', 'linux', 'darwin' ],
  'requires': [],
  'environment': {
    'NUKE_MAJOR_VERSION': '6.3',
    'NUKE_MINOR_VERSION': '8',
    'NUKE_BASE': '${PG_SW_BASE}/thefoundry/${NUKE_VERSION}',
    'NUKE': { 'darwin': '${NUKE_BASE}/${NUKE_VERSION}.app/Contents/MacOS',
              'linux': '${NUKE_BASE}',
              'windows': 'C:/Program Files/Nuke${NUKE_MAJOR_VERSION}v${NUKE_MINOR_VERSION}',
    'NUKE_VERSION': 'Nuke${NUKE_MAJOR_VERSION}v${NUKE_MINOR_VERSION}',
    'NUKE_EX_VERSION': 'NukeX${NUKE_MAJOR_VERSION}v${NUKE_MINOR_VERSION}',
    'PATH': { 'darwin': '${NUKE_BASE}/${NUKE_VERSION}.app/': '${NUKE_BASE}/${NUKE_EX_VERSION}.app/',
              'linux': '${NUKE}',
              'windows': '${NUKE}'}},
  'optional': { 'dev':
    { 'NUKE_PATH': '${DEV_BUILDS}'
    },
  },
}
```
EXECUTION

[ insert inappropriate image here ]

• user requests the versions of software they need

• list of dependencies are built and packages sourced

• the environment and dependencies are resolved

• a flattened (unrolled) environment is stored

• this is used to “set” the current working environment

• success!
A TOOL BY MANY NAMES

• need (Weta, originated at POP)

• use (Tippett and others)

• fuse (Fuel “use”)

• rez (Dr. D, open sourced and very verbose)

• ecosystem (Peregrine)
ECOSYSTEM= AN EXAMPLE -
TO THE SHELL!
CAVEATS

• python doesn’t let you set the current environment

• on unix platforms the environment can be stored to a temporary file and sourced

• on windows the environment can be launched/set for each invocation of applications wrapped in a .cmd file
SOME BENEFITS
( IF NOT OBVIOUS )

• it’s easy to push out new package descriptions to support newly installed software

• no environment clashes, especially on Windows ( Maya 2012 and 2013 plugins fighting for resolution etc. )

• easy to separate development and release version where staging and testing is extremely easy

• control over sequence and shots dependencies, may be controlled by artists or supervisors
Now we can think about build systems

- A system to wrap up the dependencies and steps needed to build (compile) source code to produce a tool
- Generally referred to as tool chains; compiler, linker and other build tools
- Visual Studio, XCode, makefiles, Scons and CMake
CMAKE FOR THE WIN

- cross platform
- mature and becoming more widely supported
- generators for different toolchains
- a module concept, very useful for defining dependencies
CMAKE GENERATORS

• used to build intermediate files for the target platform all from a common source

• Makefiles on Linux

• Makefiles/XCode Projects on OSX

• NMake files/MSVC Projects on Windows ( Jom for NMake builds )
CMAKE MODULES

• Call FindPackage.cmake, ie. FindQT4, FindTiff, FindHDF5 etc.

• We can derive our own, ie. FindMaya, FindNuke, FindPRMan, FindHoudini, FindArnold

• having a common environment makes this much easier to resolve/implement the desired version and dependencies

• let’s look at one!
CMAKE TARGETS & CONFIGURATIONS

• each cmake project can contain multiple targets

• each target inherits a global environment with the option of specifying/overriding target specific options

• a configuration can be one of Debug, RelWithDebInfo and Release

• each configuration controls global build options to provide a specific style of binary output
CMAKE EXAMPLE - BACK TO THE SHELL!
I DON’T WANT TO BUILD C++, SHOULD I STILL CARE?

• no one said you have to do any of this, but it’s hard not to argue the merit in having a controlled environment across multiple platforms

• managing software dependencies and show/seq/shot requirements it’s probably worth leveraging beyond any sort of development

• python, perl, tcl, your language of choice, still requires modules to be installed - virtual env and other tools can be used but it still good to have a studio wide means of managing these
TAKING IT A STEP FURTHER

• easy to set an environment on the render farm (keys become need dependencies to be sourced)

• store environment “needs” in EXR meta data for historical purposes

• easy to expand to external locations (on set equipment and outsource)

• integrate license and resource management (choose to access different pools of a render farm based on dependencies)
THE CLOUD
(“ON A BEACH IN COSTA RICA”)

• leverage version control for global access to the toolset (GitHub)

• S3 for distribution

• a new tag in a package, bundle://

• bundle encapsulates the cross platform installation for the package based on hdf5 and fuse file system (like a dmg or custom image)

• a nice side effect is easy disaster recovery
RE-ITERATING GOAL

- simple control
- clear understanding of context
- flexibility of tools
- a common environment
QUESTIONS/DISCUSSION ?