Game Programming: The BIG Picture

Dan White
Studio Technical Director
Pipeworks
danw@pipeworks.com

The very old days...

25 years ago, a game might have 1 programmer, and maybe 1 artist
View from the CEO’s chair...

• Games are big business
  – Current AAA titles: $30 million budget, 2+ year time frame
  – Project completion date is very important
    • Particularly for licensed properties

This means we make games differently...

When I first noticed change...

Jumpman
Atari 800
circa 1983:

This game had an EDITOR!
At the time, this seemed revolutionary to me.
A more detailed view of today...

Engineer = programmer = software developer

Distribution of People

Head Count

On large projects, programmers are a small part
Where is test? Forgotten as usual..
The real task of engineers

• The main task of engineers in a commercial game company is to make tools and runtime to enable artists and designers to make games.
  – Saves money, but just as important increases predictability.
• This is how most industries work anyway...you don’t need programmers to use Autocad.
• How to design and create the perfect engine is very much unsolved.
• Not to worry, we are decades away from losing the chance to be creative.

Actually it has been sort of solved

• Flash has taken over the market for 2D games.
• It’s a huge success.
• Flash games can be made with only scripters.
• This is possible because:
  • Performance not an issue for 2D
  • 2D games are vastly simpler
• So successful that MS is making their own: Silverlight
Adobe Flash CS4

Flash suggests form of the general solution

• Artists make assets
• Designers use the assets to make levels.
• The levels are packed to make game data.
• The runtime interprets & processes the game data.
• Sound familiar?
  – Source file → Compiled EXE
  – EXE run by an OS
A slightly more detailed view...

Artists

DCC Tools

Art Assets

Designers & Scripters

Level Editor

Level File

Bundler

Aka: Distiller, packer Resource converter

DCC = Digital content Creation.

There will be at least one version for each target platform: PSP, PS2 XBOX, WII, X360, PS3, PC

Runtime

The art pipeline...

Render monkey, FX composer

.fx or .cg files

Photoshop

PSD/Tiff files

Source of:
Meshes
Animations
Materials
Textures
Skins

Maya, 3ds Max, Motionbuilder XSI (commercial products)

Special Pluggins

Export Pluggins

Collada

Proprietary binary files

Originally targeted at film.
The level editing pipeline...

- Art assets
- Scripts
- Special effects

Engine specific level editor → Level file

- Object placement & properties
- Setup lights
- Create light/shadow maps
- Script binding (connect triggers and so forth)
- Script testing
- Create camera paths & cinematic sequences.
- In some cases...creating of the static world geometry (now out of fashion)
How to design a level editor is still unsettled...

- Some engines use the DCC as the level editor
- Some approaches have a special build of the game that is also the level editor.
- In some cases, the level editor also does the bundling.
Other tools

• Special Fx editor
  – Particle systems very important for look and feel.
  – Edit and tweak particle systems
  – Can be really fun to work with & neglected academically

• Sound editor
  – Take sound samples (from Sound Forge or whatever) and create runtime sound-effects by applying patches, filters etc.

• Asset management
  – This is a huge problem with > 10^5 files
Recurring Theme

- Editor
- Previewer
- Runtime
- As usual, the UI (editor) takes most of the work.
On to the Bundler...

- The bundler does the final conversion of assets to a platform specific format.
- Stripify & build meshes
- Convert texture formats
- Compress everything that can be compressed.
- Layout binary data for streaming.
- Handles endian swapping.
- Build large data structures (e.g. BSP or kD trees)
- Do every possible preprocess operation!
  - Among other advantages, finds errors early.

Runtime

- Game runtime is a real-time simulation
  - Predictable performance is important
    - Amortization is not the answer
  - Resources are limited, memory is generally scarce (relative to the PC)
    - Hardware has poor memory bandwidth.
Runtime...some important systems

- Simulation/World (Game object)
- Rigid body Physics
- Cloth/water
- Culling & “Scene Graph”
- Geometry management (e.g. BSP trees)
- Interface & GUI
- Sound
- Video
- Material Management
- Shapes & Meshes
- Special Effects
- Primitive rendering
- Platform SDK (e.g. DX9 + Win32)

Perhaps 80% is cross platform.

Runtime design issues

- Grouping of libraries varies, but functionality is common.
- Biggest unsolved question in runtime design is how to achieve threading.
  - Or, on PS3 how to use the SPE’s
- Note that I have broken the engine down much farther that the tools side. Many of the components exist there as well.
Perspective

• The tools complexity, by line count, will be multiples of the runtime side (at least for a single platform).
• The engineers use the runtime, but all the content people use the tools.
  — Training time on tools is much higher, cause more people are involved.
  — Productivity payoff is much higher for the same reason.
• Runtime changes with platform and fads, but tools can remain forever.
• A lot of what makes a runtime “good” is performance, but the payoff to the user may be low.
  — 2x increase in poly count is often barely noticeable
• We have already reached the sweet spot for most game features:
  — 4 enemies to 8 → big difference
  — 64 enemies to 128 → small difference
• Iteration makes games good, and the tool chain allows iteration.

The point...

• The whole tool chain is very important, and the runtime often gets too much attention.
• Our task as engineers is to think about the WHOLE engine, not just the runtime.
Issues with engine-centric development

- You can make a game without tools, but you can’t make a game without runtime.
  - Just like you can make a game without artists, but no without programmers.
- For financial and technical reasons, you have to make the engine in stages.
- It’s very easy to screw up. The world is filled with failed engine initiatives.
- Making an engine genre agnostic is very hard...potentially impossible.
  - Genres: FPS, RTS, RPG, fighting, puzzle, driving and the all-important 3rd persona action adventure
- In the end, users don’t give a darn...they want a fun game.

Are you excited yet!

- Engine design and implementation is utterly fascinating (at least for me).
- It touches more disciplines that almost any other software development area.
- Entry is through scripting/game programming, which is fun anyway.
The price of admission...

- Most of game & engine programming is just solid CS.
- However, there are areas which are typically missing from a CS program.
  - Mathematical modeling, vector math
  - Simulation & physics
  - Graphics, particularly special effects
  - Low level (not TCP based) networking
- This is why I’m so excited about this class!
- Some CS areas are less important:
  - Web anything
  - Databases
  - This is changing...

High-level unsolved problems

- Should materials be edited inside DCC tools?
- How should art assets be exported from DCC tools?
- How much game logic should be in script, vs. the engine?
- Is there a visual programming solution for scripting?
- How to have a data-driven design and still get predictable performance?
- What language should tools be written in? C++, C#, Java, Python?
- How should cut-scenes be created?
- At what level should network replication occur?
Lower-level problems...

- Engine structure for multiple cores/threads
- Class organization for a complex simulation
- Realistic human animation
- Destructible stuff
- Hair, Fire, Water, Smoke (Fur is mostly under control)
- Lighting
- Shadows
- Convincing & predictable AI
- Non-creepy human rendering
- Non-interior environments: Foliage, urban scenes