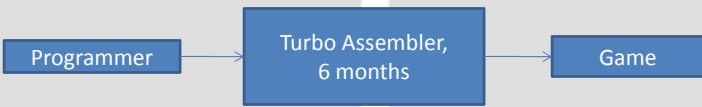




# Game Programming: The BIG Picture

Dan White  
Studio Technical Director  
Pipeworks  
danw@pipeworks.com  
a Foundation 9 Entertainment Studio

## The very old days...

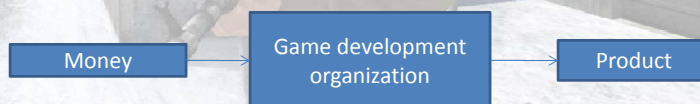


```
graph LR; A[Programmer] --> B[Turbo Assembler, 6 months]; B --> C[Game]
```

30 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, sports, holiday launches, or anything with advertising



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.

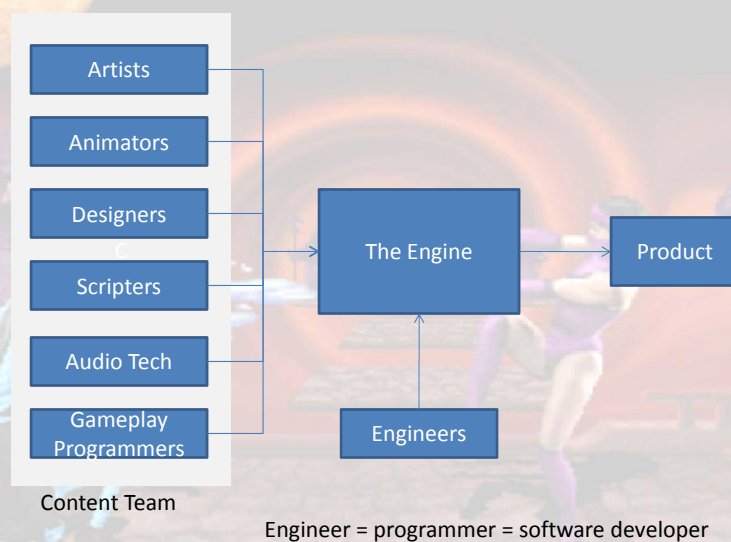
## Doom Engine cemented the idea

Doom released in 1993

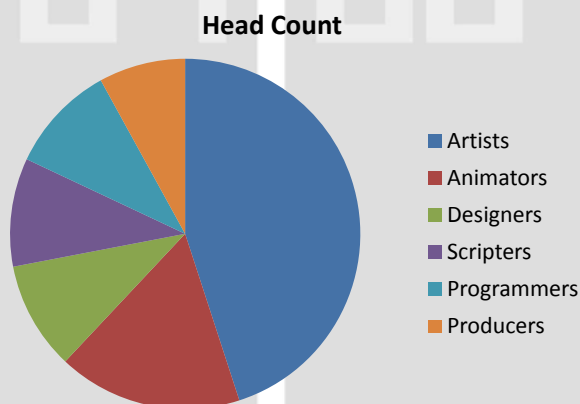
- Spawned HeXen, Heretic and so on
- Started the idea that an engine is potentially valuable on its own



A more detailed view of today...



## Distribution of People (large project)



On large projects, programmers are a small part  
 On smaller projects, a larger part  
 Where is test? Forgotten as usual..

## The real task of engineers

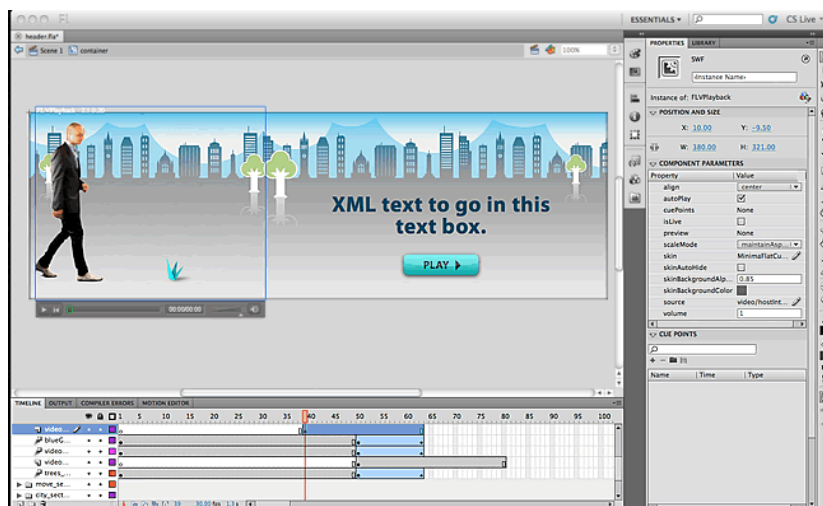
- The main task of engineers in a commercial game company is to make tools and runtime to enable artists and designers to make games.
  - Saves money, but just as important increases predictability.
- This is how most industries work anyway...you don't need programmers to use Autocad.
- How to design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

|| 19:48

## 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.
- Future is unknown: Will HTML 5 replace it, or will Flash succeed with 3d support.

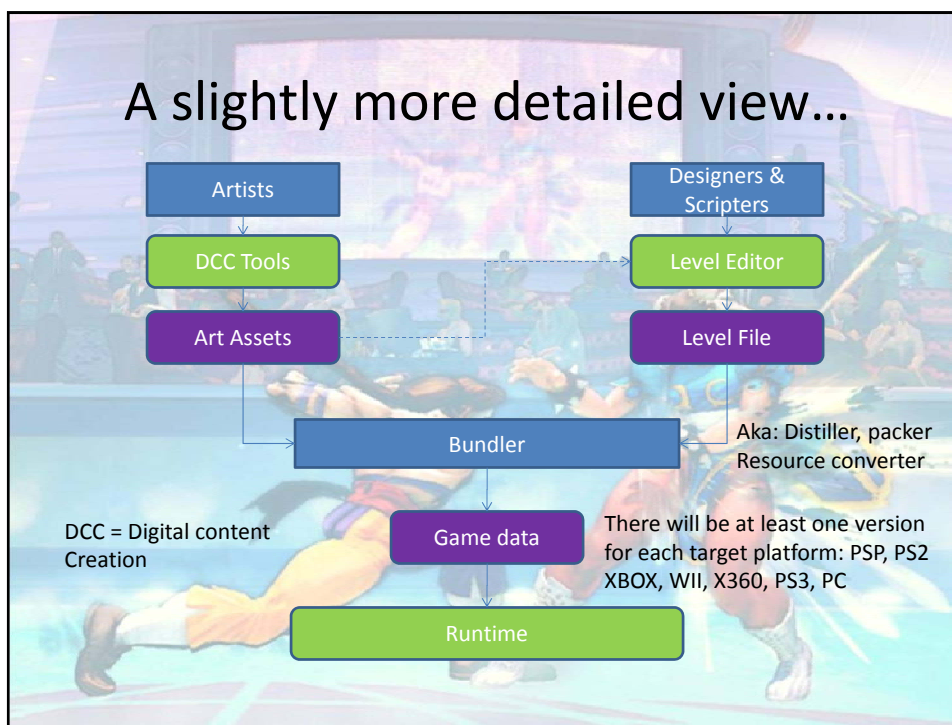
## Adobe Flash CS5

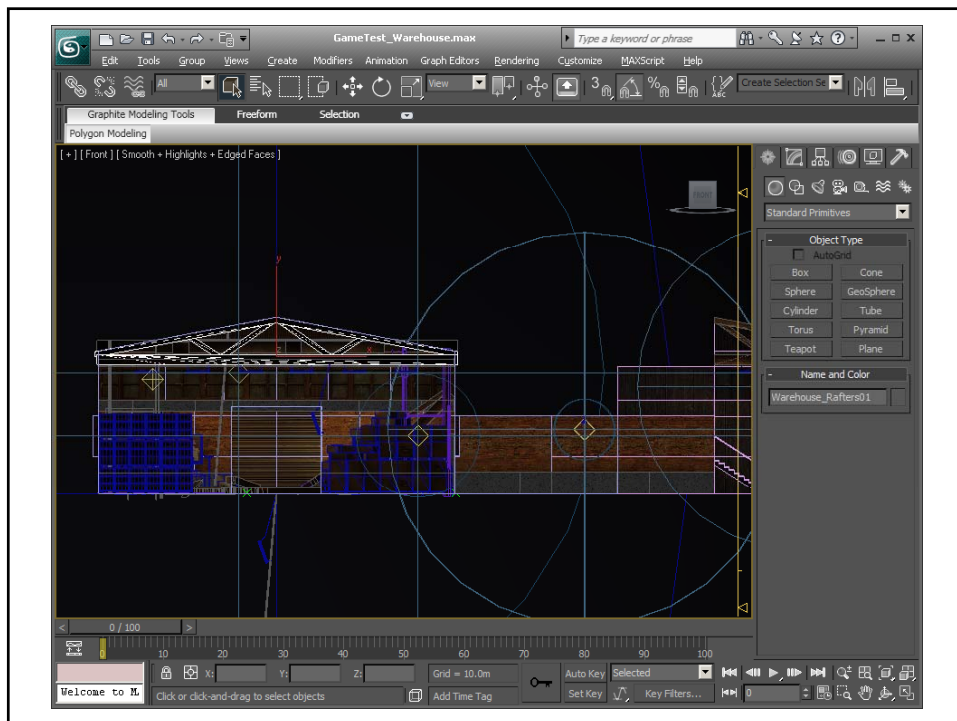
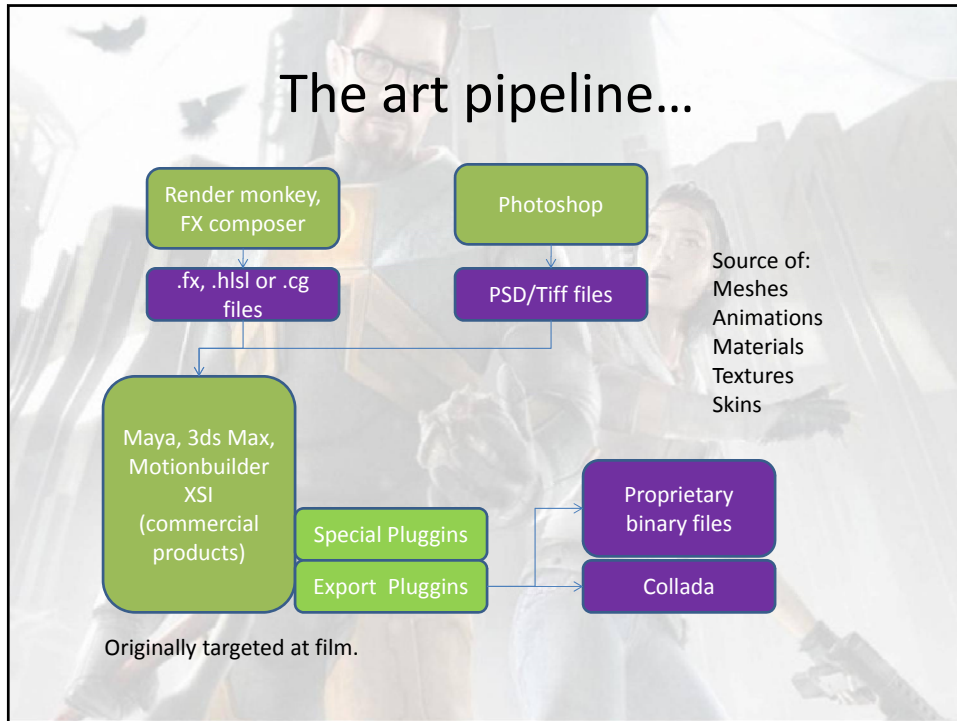


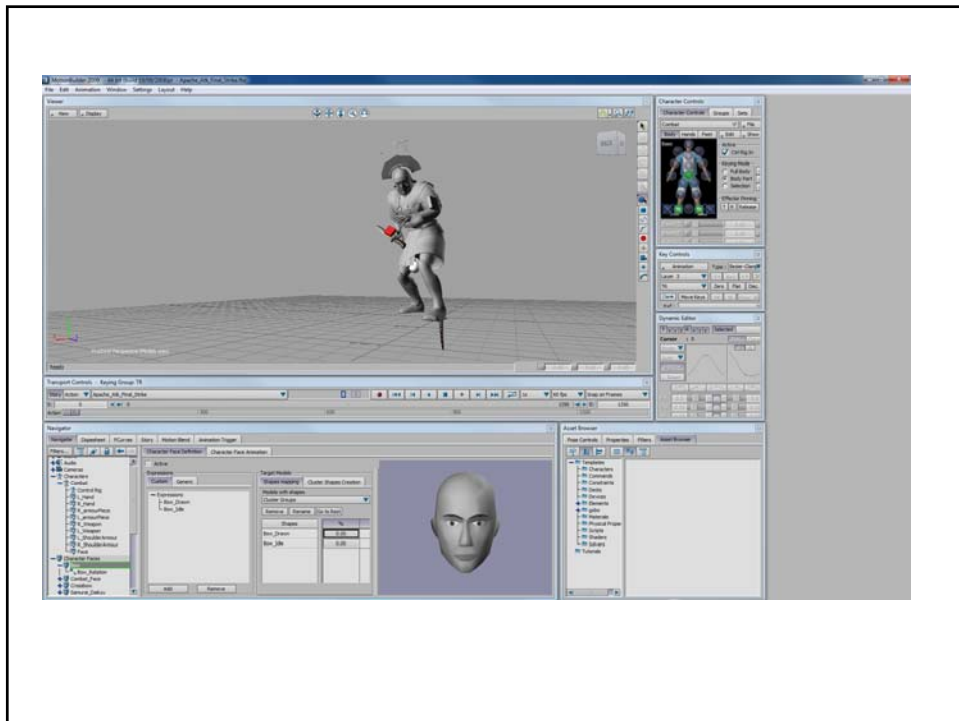
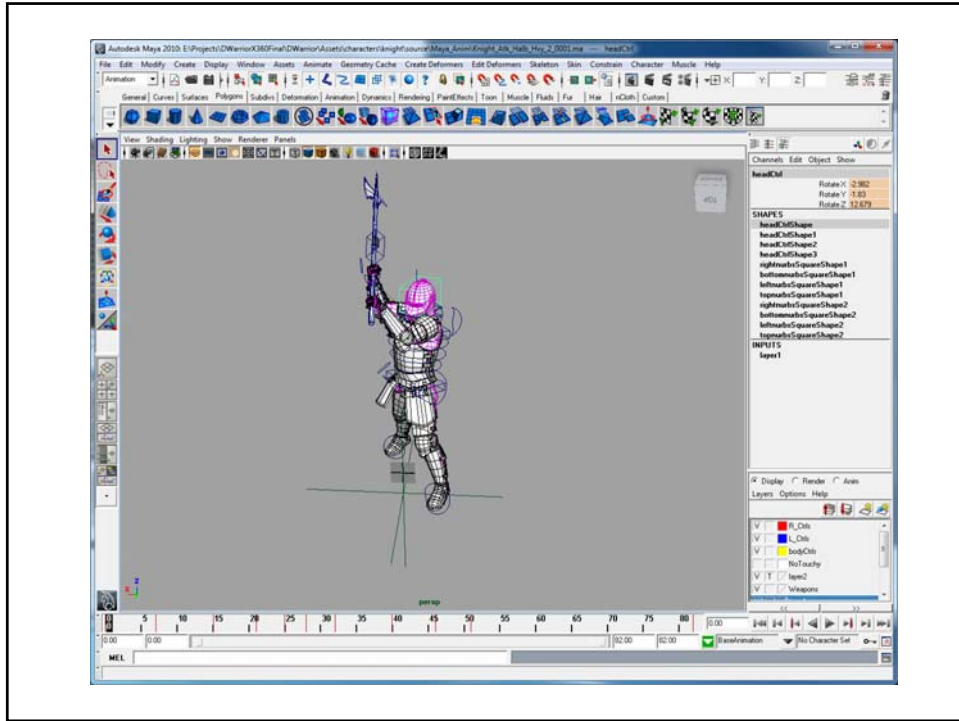
## Flash suggests form of the general solution

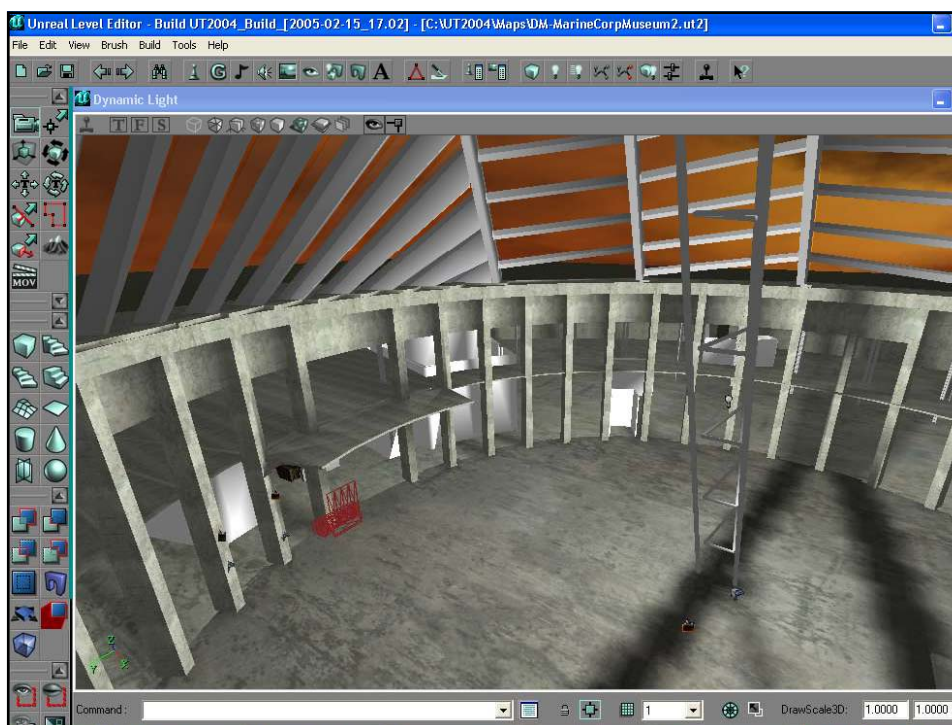
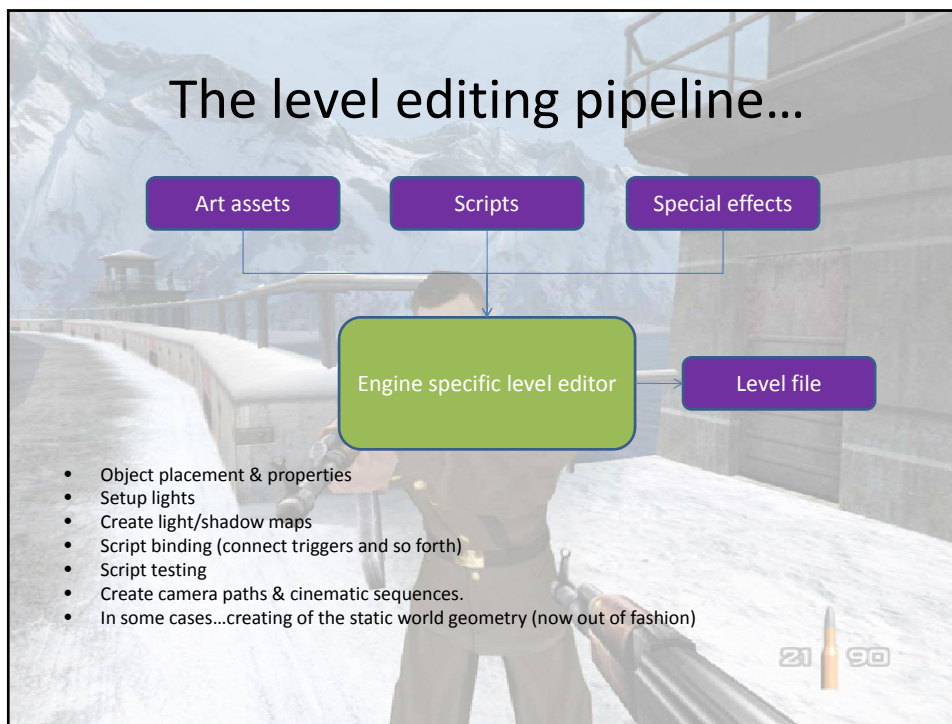
- Artists make assets.
- Designers use the assets to make levels.
- The levels are packed to make game data.
- The runtime interprets & processes the game data.
- Sound familiar?
  - Source file → Compiled EXE
  - EXE run by an OS

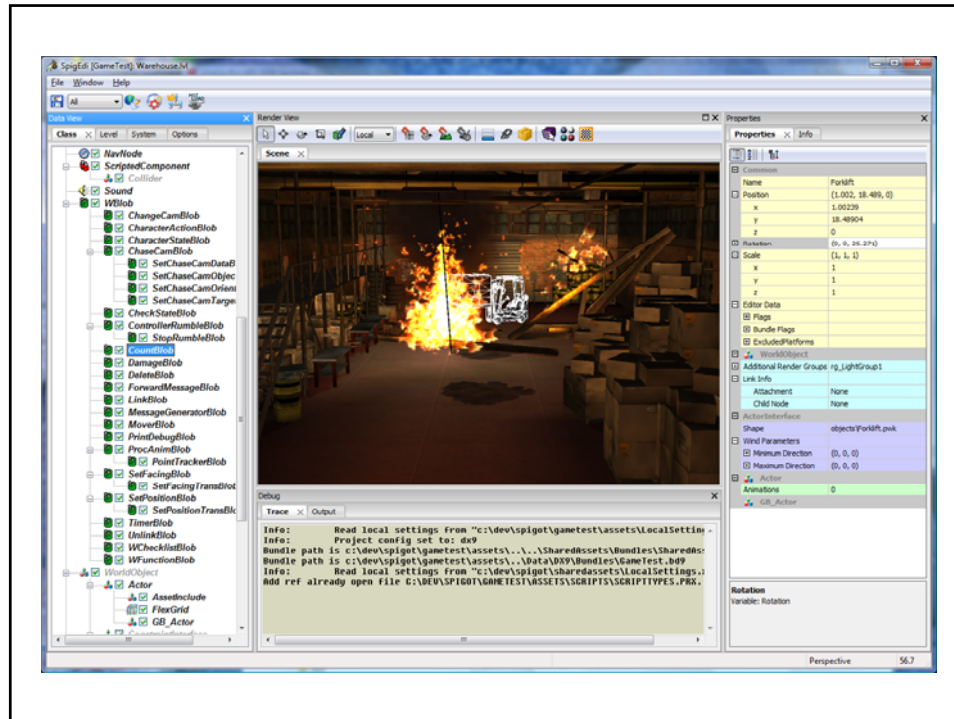
## A slightly more detailed view...









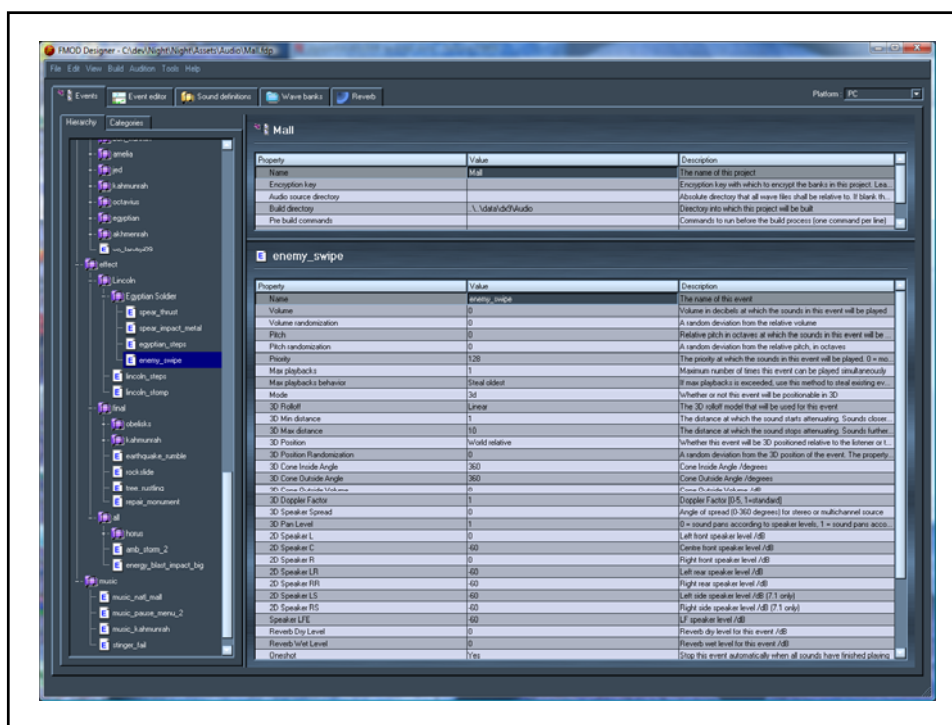


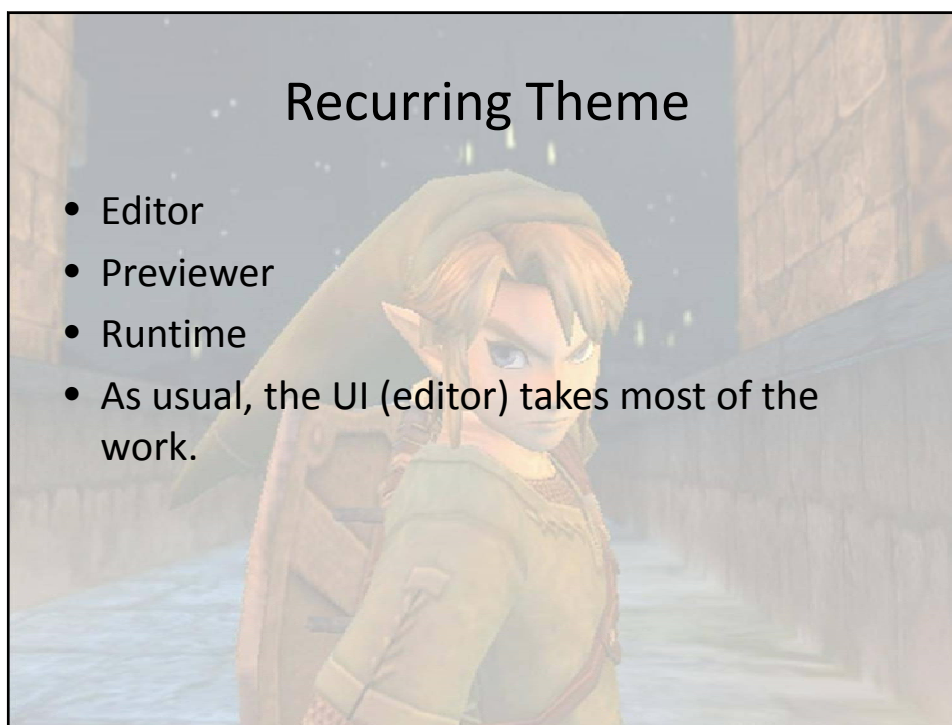
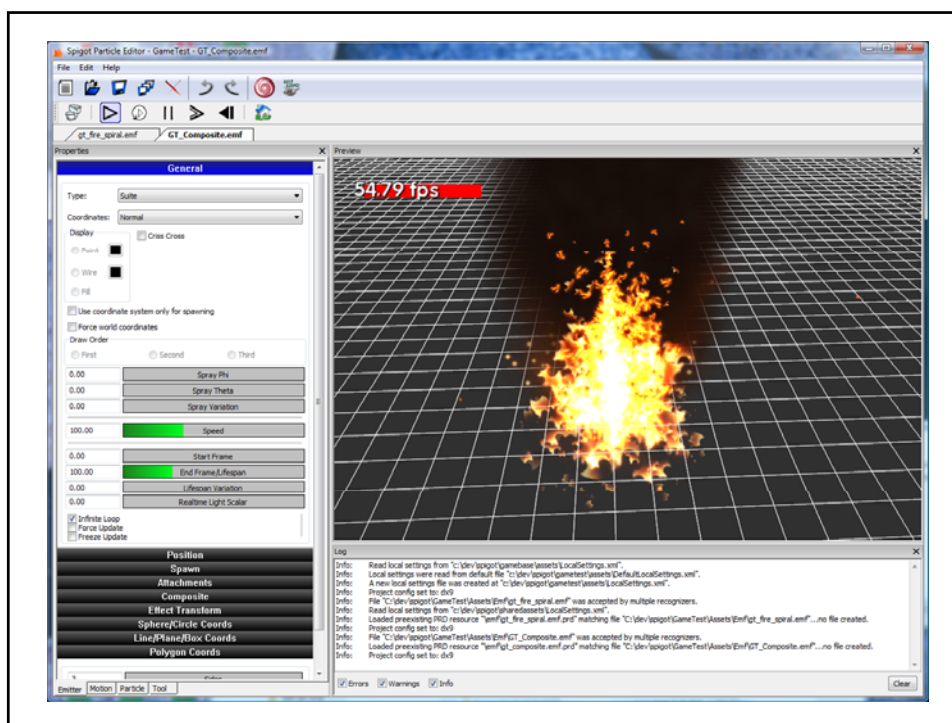
## How to design a level editor is still unsettled...

- 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<sup>4</sup>5 files



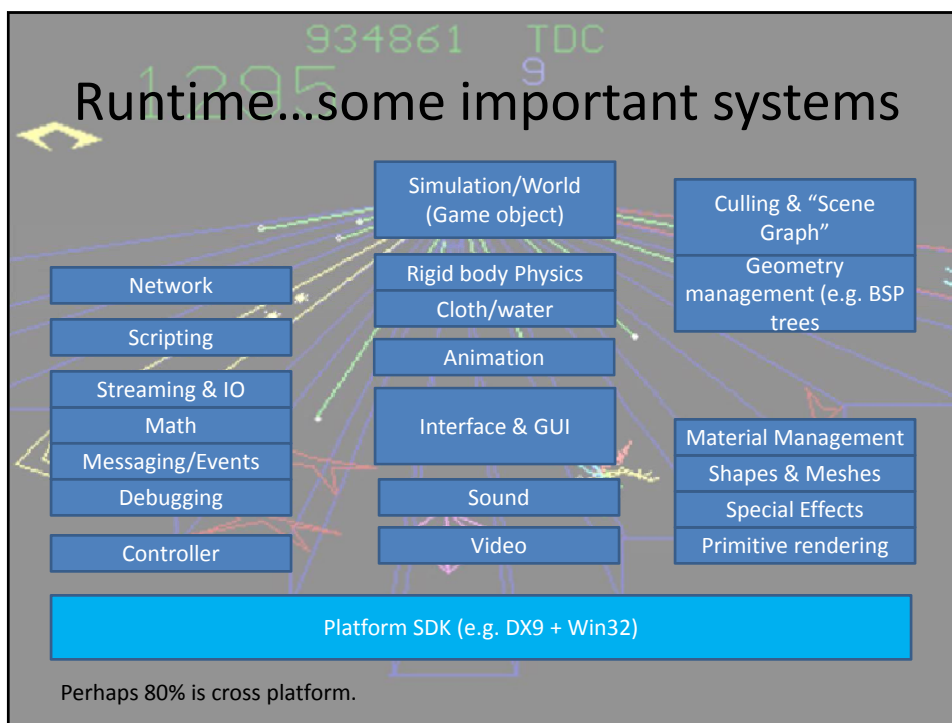


## On to the Bundler...

- The bundler does the final conversion of assets to a platform specific format.
  - Like the compiler
- Stripify & build meshes
- Convert texture formats
  - Also build texture atlases.
- 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.

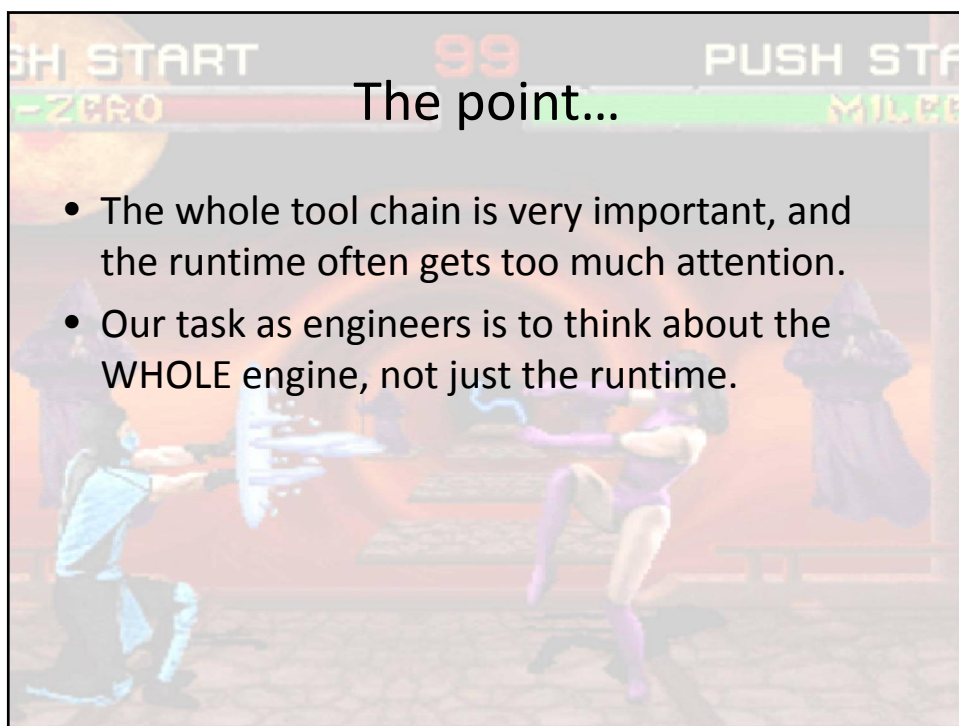
## Runtime

- Game runtime is a real-time simulation
  - Predictable performance is important
    - Amortization is not the answer
    - Garbage collection is...garbage
  - Resources are limited, memory is generally scarce (relative to the PC)
  - Hardware has poor memory bandwidth & small caches



## 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 than the tools side. Many of the components exist there as well.



When it all goes together, you get:



|| 19:48

## 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
- **Iteration makes games good, and the tool chain allows iteration.**

## Iteration – DW1 vs DW2



## 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 is 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 3<sup>rd</sup> persona action adventure
- In the end, users don't give a darn...they want a fun game.

## Are you excited yet!

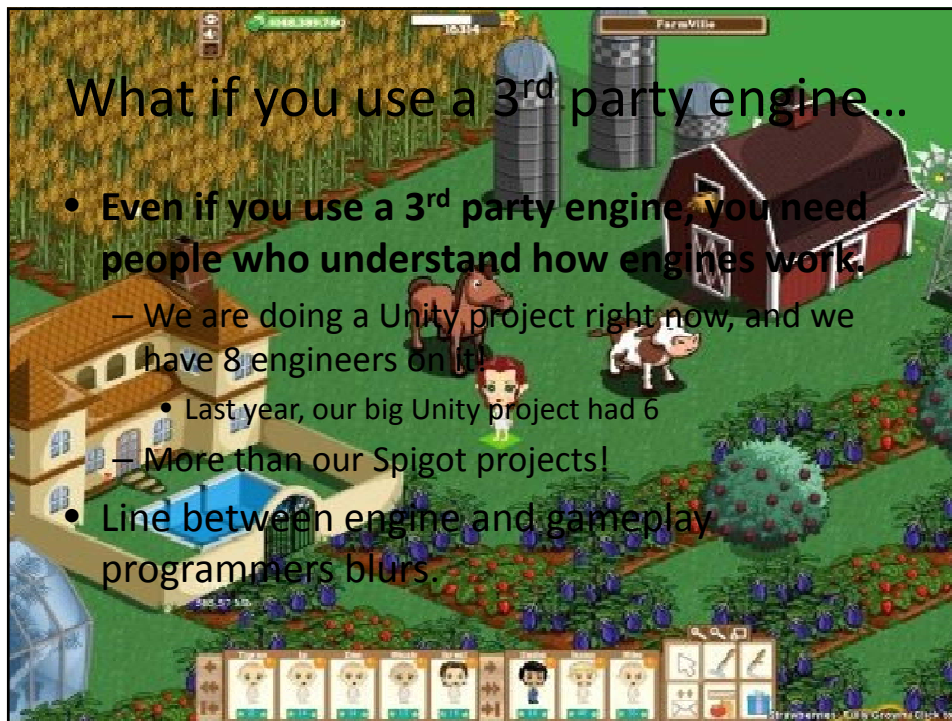
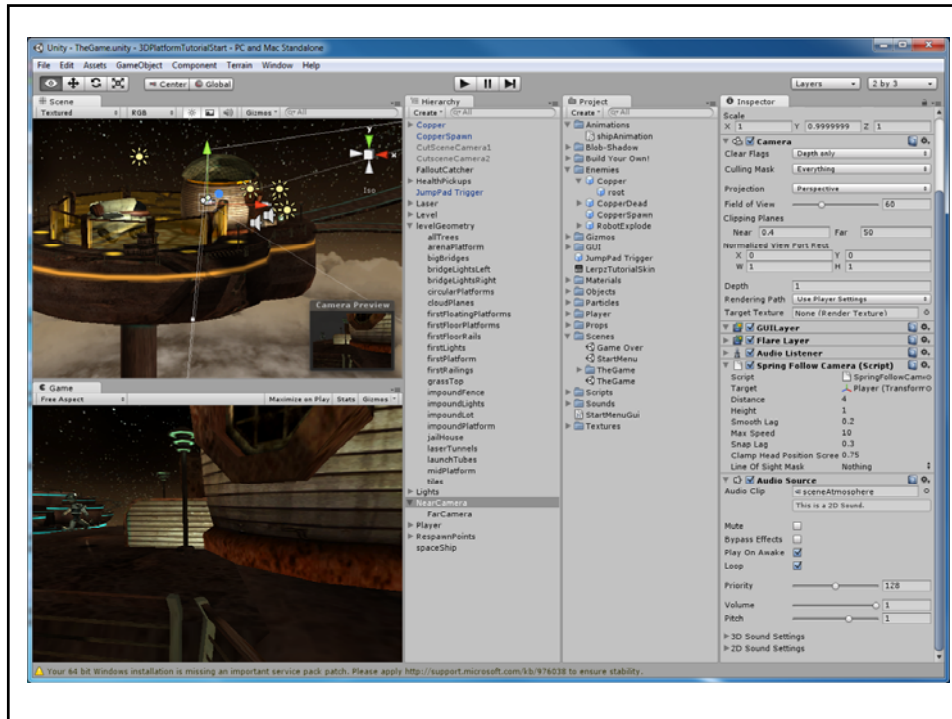
- Engine design and implementation is utterly fascinating (at least for me).
- It touches more disciplines that almost any other software development area.
- Entry is through scripting/game programming, which is fun anyway.

## What about small developers?

- Until recently, needing an engine was a barrier to entry.
- A small company can't maintain an engine team.
  - Pipeworks, at 54 people is probably too small.
    - Our secret is that we've been at it for 12 years!
    - Key to this is software engineering...another topic.
  - "Small" gets larger every year.
- Viable 3<sup>rd</sup> party engines now exist.

## Unity

- Combines in one tool:
  - Level editor
  - Particle editor (weak)
  - Previewer
- Missing UI system...but they are working on it
- Targeted at mobile & browser.
  - In fact, owns this market.
  - Moving quickly: support for larger teams and so on.
  - The world is Unity's to lose.
- Unlike Ogre and some others, used to make many commercial games.
- Unity is an excellent thing to study!

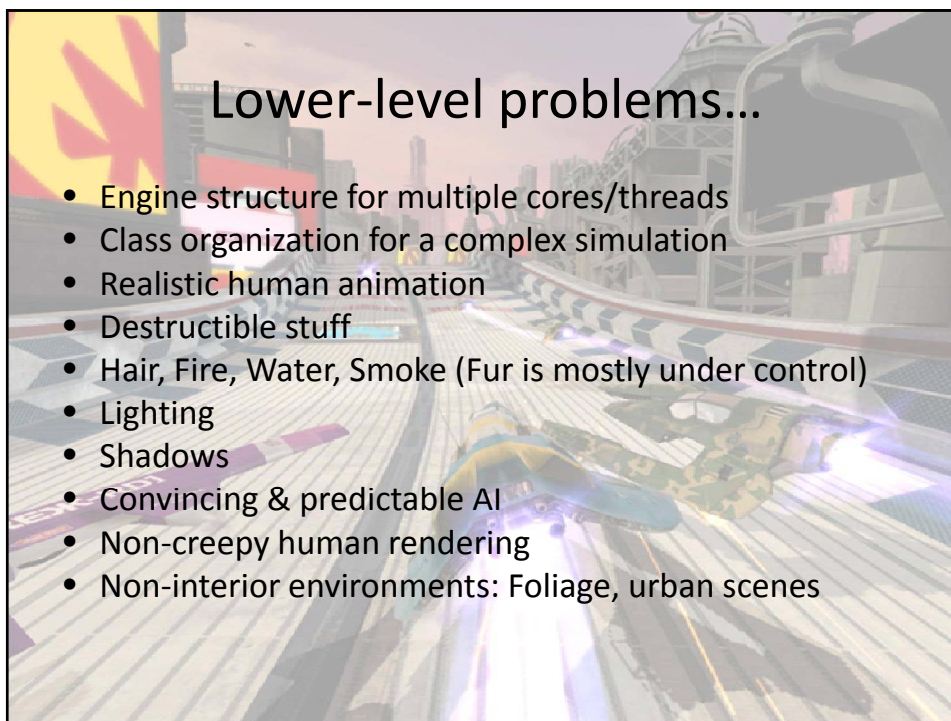


## The price of admission...

- 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!
- Database and server programming are now important as well.

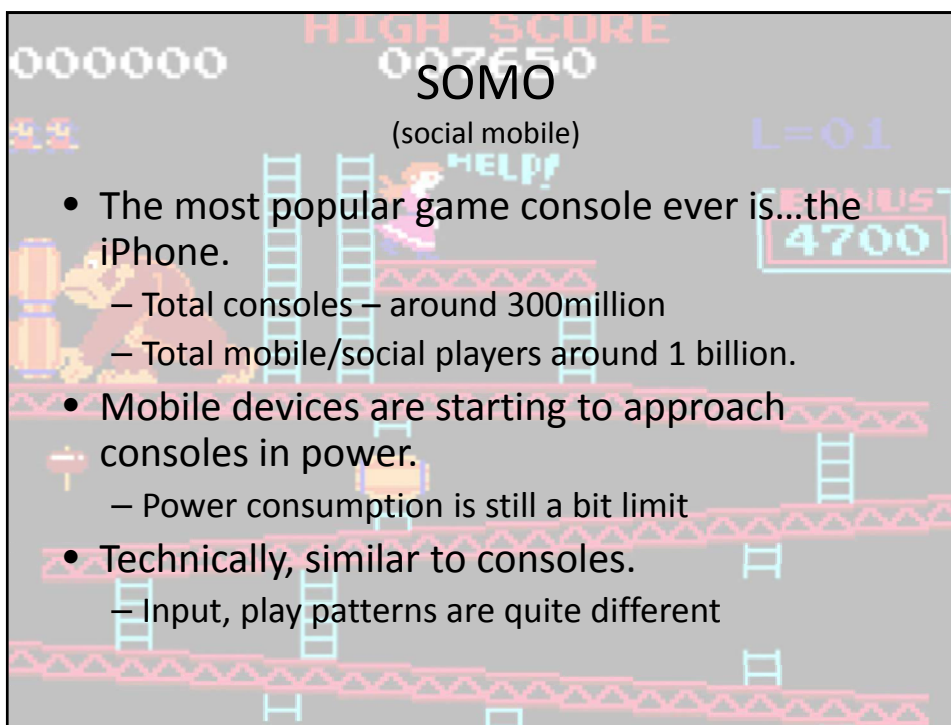
## High-level unsolved problems

- 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?
- How to analyze and tweak a freemium economy.



## 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



## SOMO

(social mobile)

- The most popular game console ever is...the iPhone.
  - Total consoles – around 300million
  - Total mobile/social players around 1 billion.
- Mobile devices are starting to approach consoles in power.
  - Power consumption is still a bit limit
- Technically, similar to consoles.
  - Input, play patterns are quite different

## Back End

- Almost all games now have a back end
  - Cloud based save load
  - Easy multiplayer implementation
  - Social integration
  - Freemium economy & transactions
  - Piracy protection
    - Also used games
  - Hosting services (Amazon, Rackspace) make it simpler
- Typically implemented via a HTTP/HTTPS & REST
  - Most common back ends are PHP, but all kinds used
  - Most common database is MySQL, but nosql is gaining
    - Games are much more write heavy than other Web apps
  - Scalability is a problem

## Console Evolution

	NES	PS1	PS2	PS3	Current PC
CPU	MOS Technology 6502 1.79 MHz	MIPS R3000A-32-bit RISC chip running at 33.8688 MHz	294 MHz MIPS "Emotion Engine"	3.2 GHz POWER-based PPE with seven 3.2 GHz SPEs	8 Cores at 3 GHz
GPU	-	66 MIPS vector math unit on CPU	147 MHz "Graphics Synthesizer"	550 MHz based on Nvidia G70	Gerforce 650 1058 Mhz 384 Cores
Storage	Cartridge	CD	DVD	Blu-Ray	Internet

## Content is King

- We have already reached the sweet spot for most game features:
  - 4 enemies to 8 → big difference
  - 64 enemies to 128 → small difference
- PS2 was the tipping point between ability to display content and ability to make it.
  - We are now firmly in the era where content creation cost is the driving factor.
    - Ironically, main content limit is DVD->Memory transfer