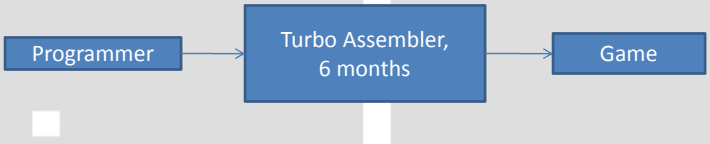
# Game Programming: The BIG Picture

Dan White
Studio Technical Director
Pipeworks
danw@pipeworks.com



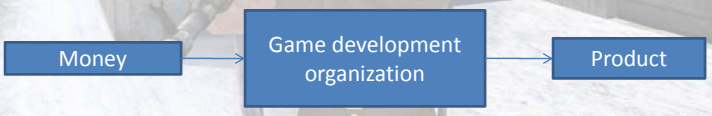
## The very old days…




25 years ago, a game might have 1 programmer, and maybe 1 artist

# View from the CEO's chair…

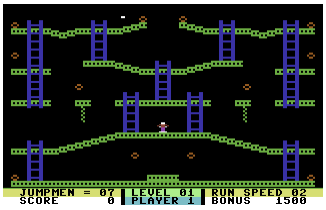
- Games are big business
  - Current AAA titles: $30 million budget, 2+ year time frame
  - Project completion date is very important
    - Particularly for licensed properties




This means we make games differently…

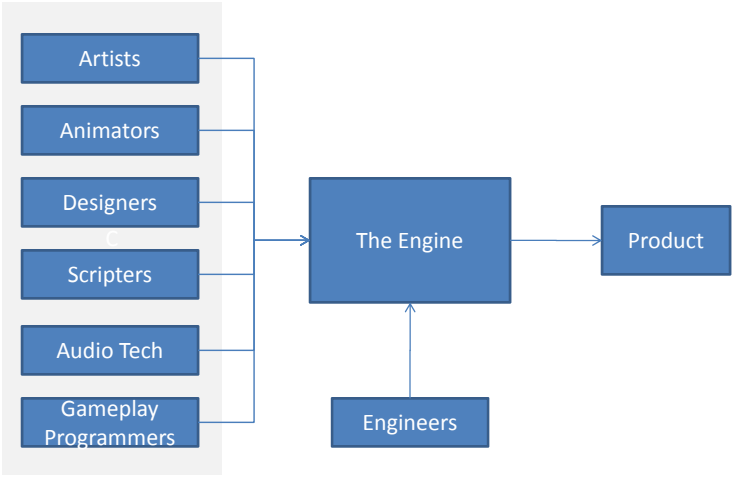
# When I first noticed change…

Jumpman
Atari 800
circa 1983:



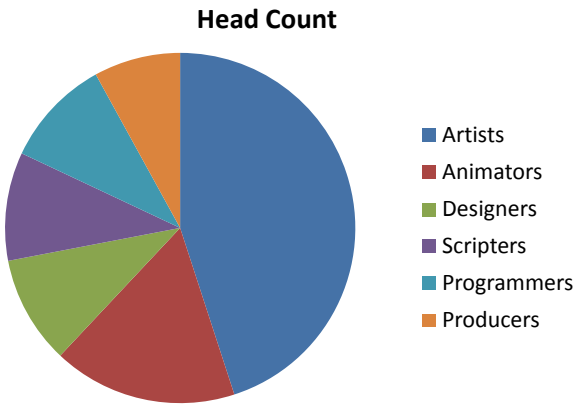

This game had an EDITOR!
At the time, this seemed revolutionary to me.

# A more detailed view of today…




Engineer = programmer = software developer

# Distribution of People




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

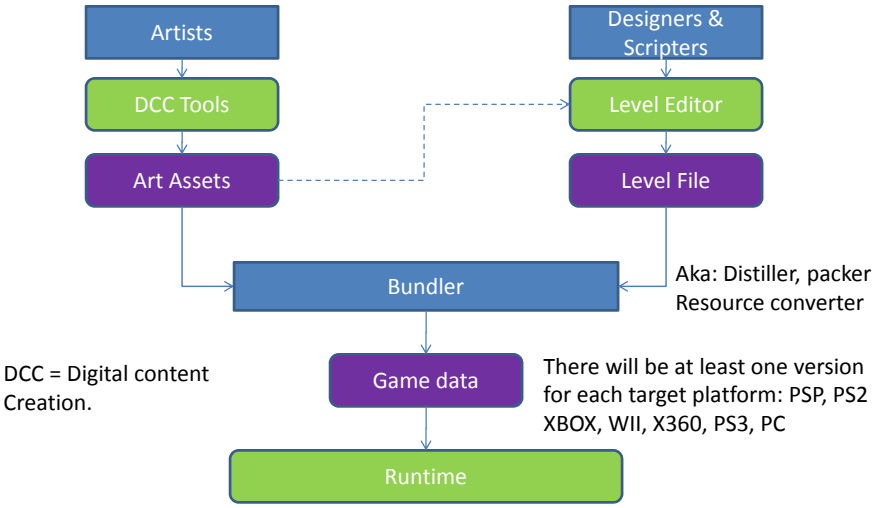
- Flash has taken over the market for 2D games.
- It's a huge success.
- Flash games can be made with only scripters.
- This is possible because:
  - Performance not an issue for 2D
  - 2D games are vastly simpler
- So successful that MS is making their own: Silverlight

# Adobe Flash CS4

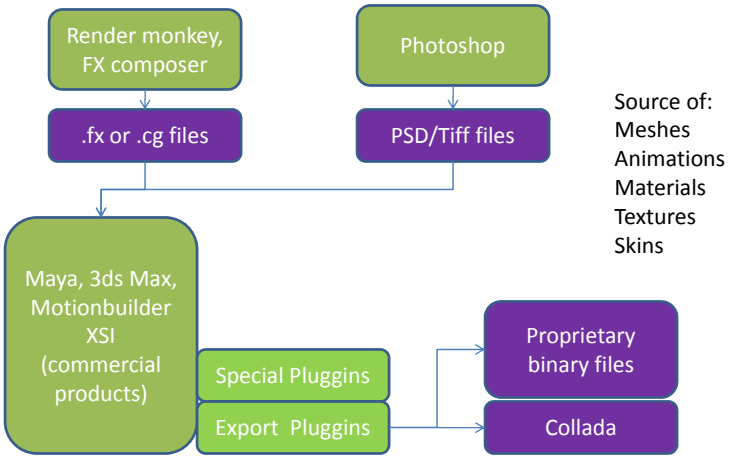

# Flash suggests form of the general solution

- 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

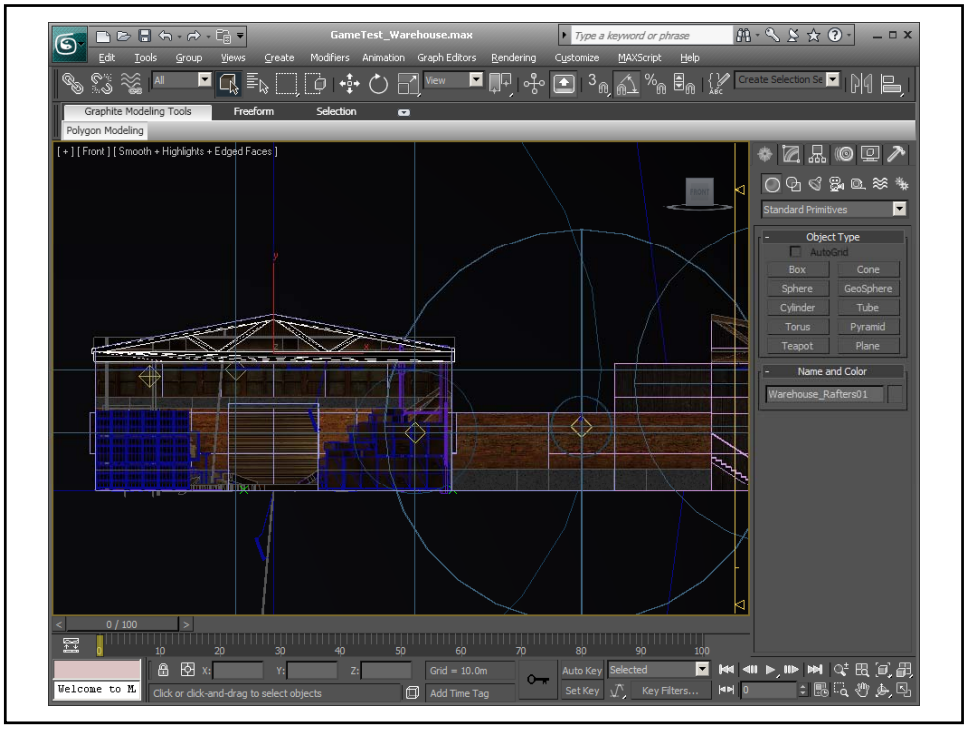
# A slightly more detailed view…

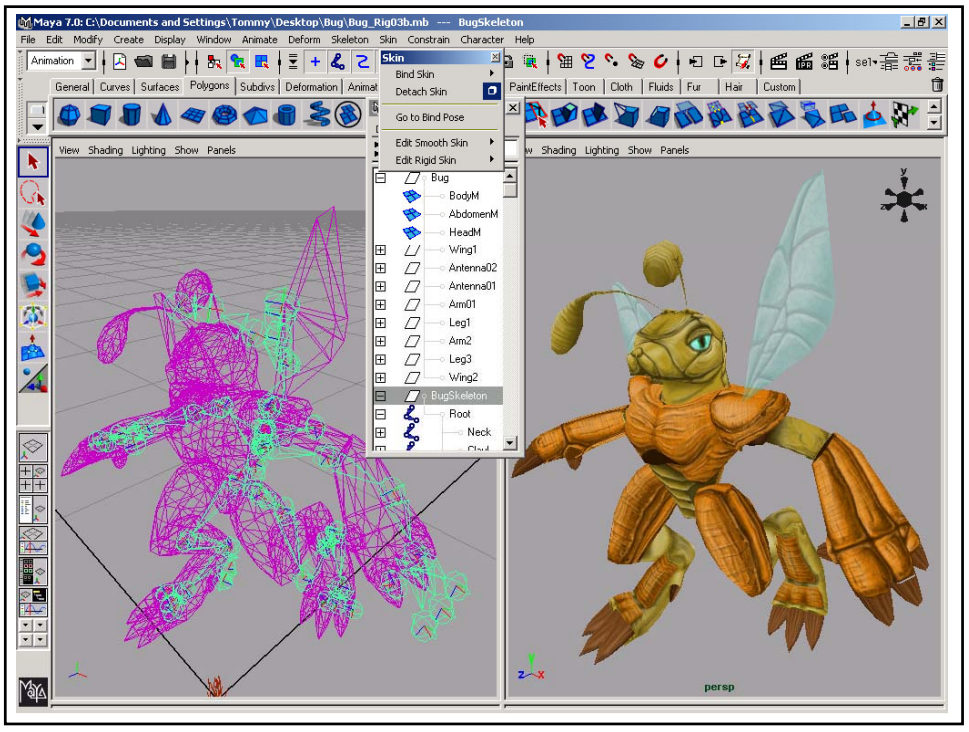



# The art pipeline…

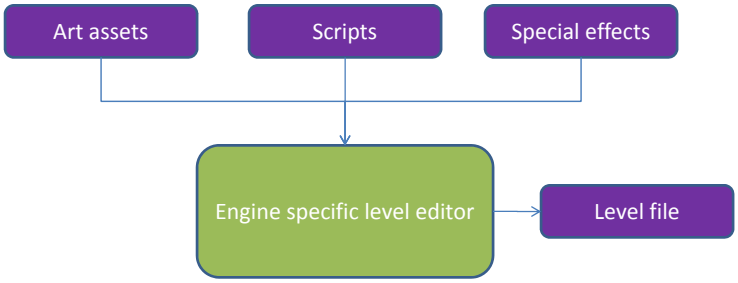



Originally targeted at film.

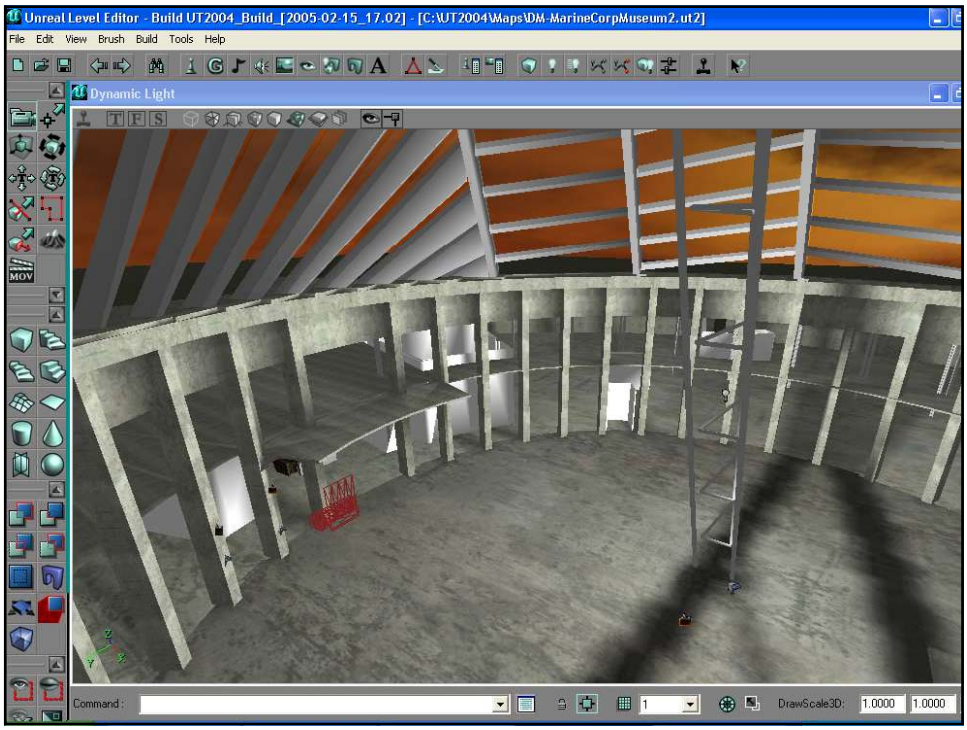

GameTest_Warehouse.max
Type a keyword or phrase
File Edit Tools Group Views Create Modifiers Animation Graph Editors Rendering Customize MAXScript Help
Graphite Modeling Tools
Freeform
Selection
Polygon Modeling
[ + ] [ Front ] [ Smooth + Highlights + Edged Faces ]
Standard Primitives
Object Type
AutoGrid
Box
Cone
Sphere
GeoSphere
Cylinder
Tube
Torus
Pyramid
Teapot
Plane
Name and Color
Warehouse_Rafters01
0 / 100
Grid = 10.0m
Auto Key
Selected
Set Key
Key Filters...
Add Time Tag
Welcome to M
Click or click-and-drag to select objects

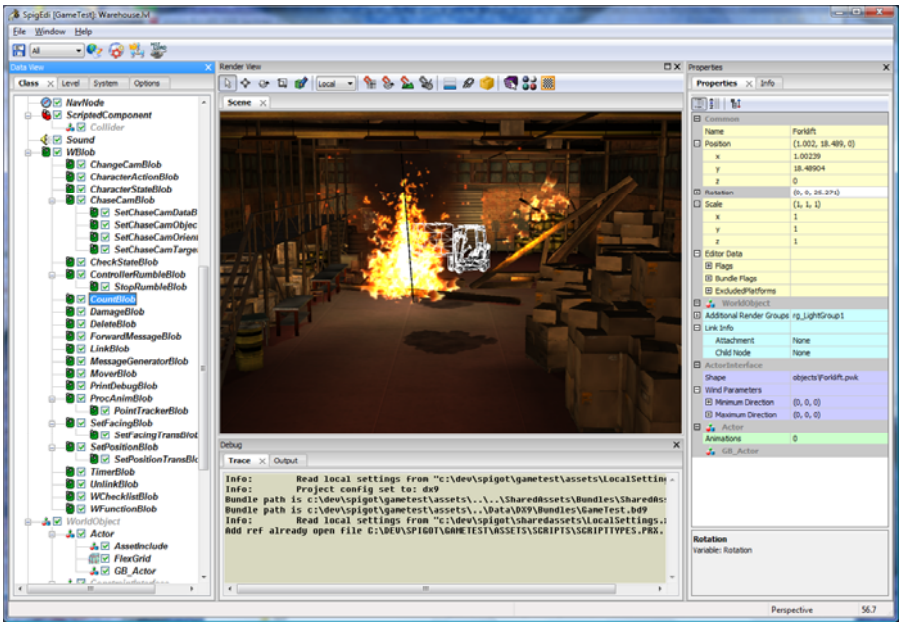

Maya 7.0: C:\Documents and Settings\Tommy\Desktop\Bug\Bug_Rig03b.mb --- BugSkeleton
File Edit Modify Create Display Window Animate Deform Skeleton Skin Constrain Character Help
Animation
General Curves Surfaces Polygons Subdivs Deformation Animat
PaintEffects Toon Cloth Fluids Fur Hair Custom
View Shading Lighting Show Panels
Skin
Bind Skin
Detach Skin
Go to Bind Pose
Edit Smooth Skin
Edit Rigid Skin
Bug
BodyM
AbdomenM
HeadM
Wing1
Antenna02
Antenna01
Arm01
Leg1
Arm2
Leg3
Wing2
BugSkeleton
Root
Neck
persp

# The level editing pipeline…




- 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)



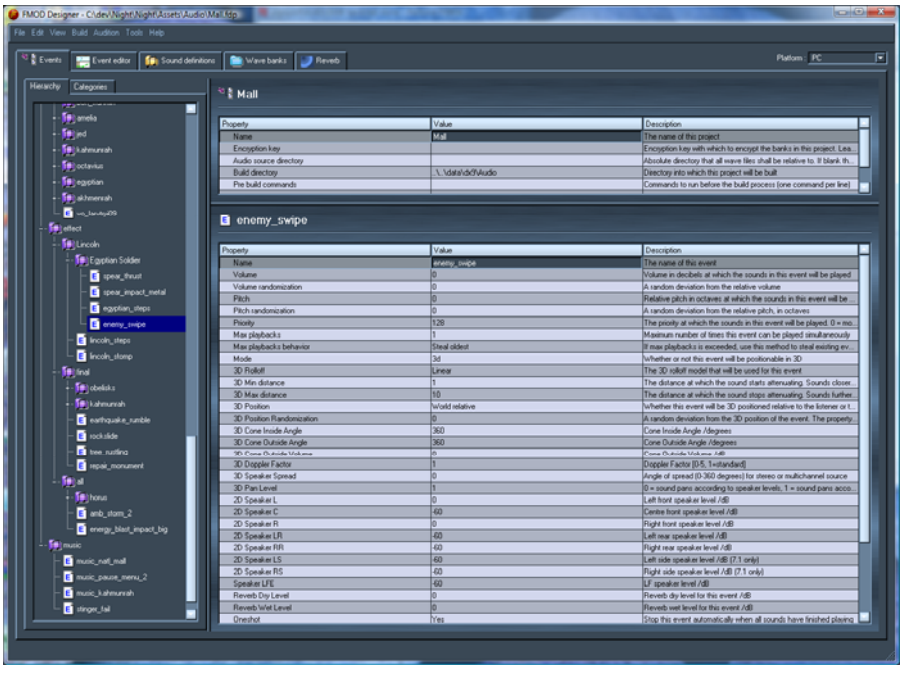


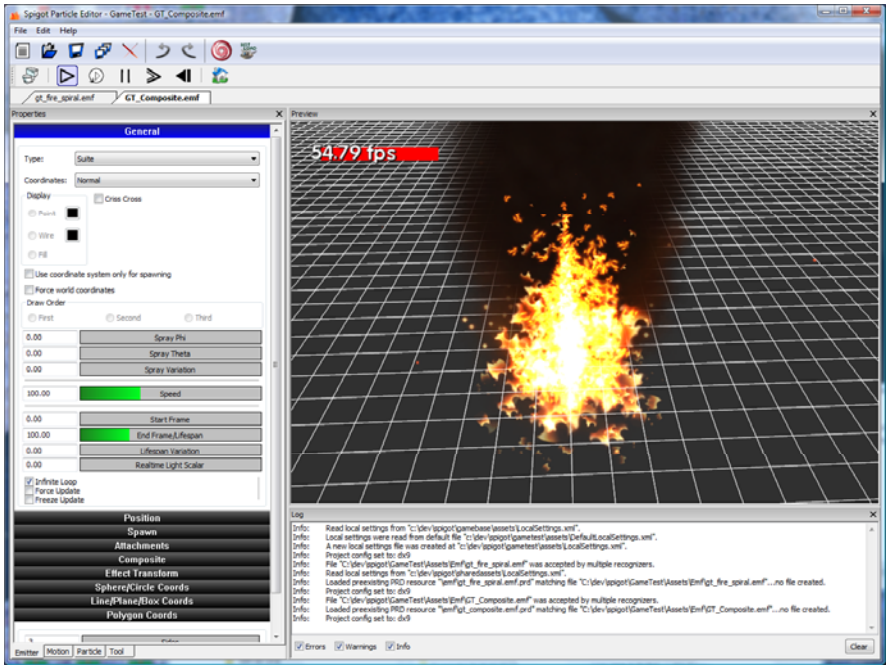
## How to design a level editor is still unsettled…

- 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




Spigot Particle Editor - GameTest - GT_Composite.emf
File Edit Help
gt_fire_spiral.emf
GT_Composite.emf
Properties
General
Type:
Suite
Coordinates:
Normal
Display
Criss Cross
Use coordinate system only for spawning
Force world coordinates
Draw Order
First
Second
Third
Spray Phi
Spray Theta
Spray Variation
Speed
Start Frame
End Frame/Lifespan
Lifespan Variation
Realtime Light Scalar
Infinite Loop
Force Update
Freeze Update
Position
Spawn
Attachments
Composite
Effect Transform
Sphere/Circle Coords
Line/Plane/Box Coords
Polygon Coords
Emitter
Motion
Particle
Tool
Preview
54.79 fps
Log
Errors
Warnings
Info
Clear


Recurring Theme
• Editor
• Previewer
• Runtime
• As usual, the UI (editor) takes most of the work.

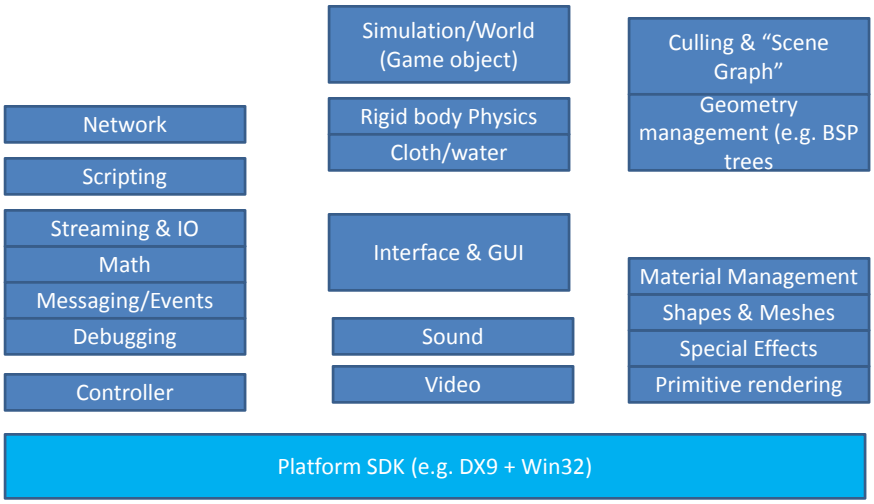
## On to the Bundler…

- 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



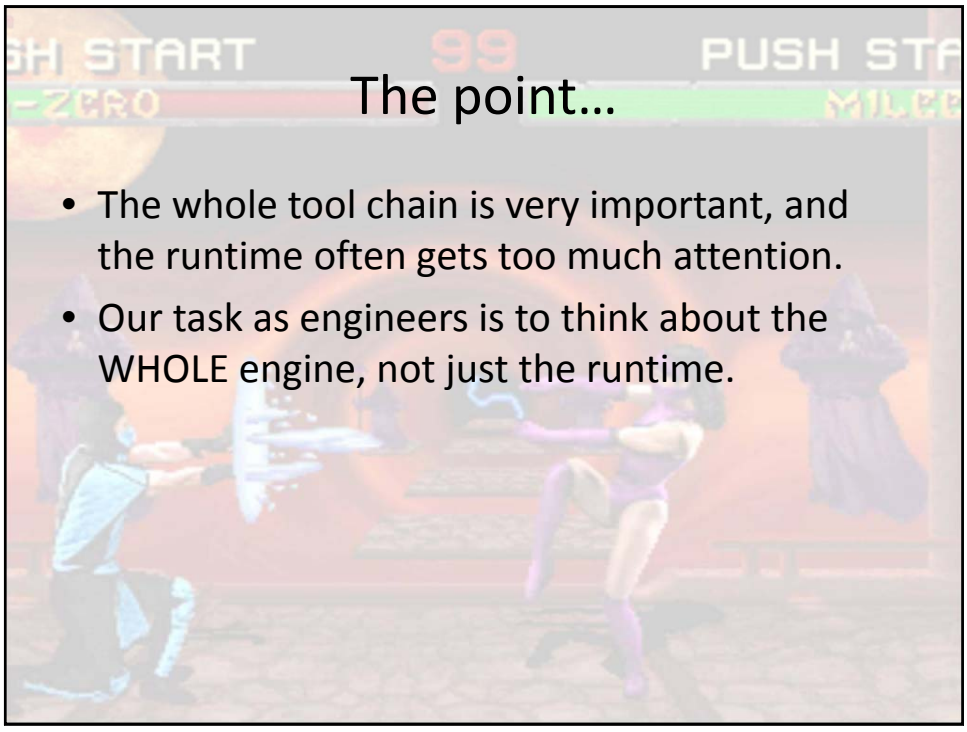

Perhaps 80% is cross platform.

# 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.
- Its 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