

G54GAM Games

Building Games

Architectures

Now what?

- Given a game design, how do we go about building it?
 - Highly complex interactive system
- We need to plan it otherwise it becomes a mess
 - Difficult to understand
 - Difficult to maintain
 - Difficult to extend
- “Games must be designed, but computers must be programmed”
 - Often mainly native
 - C, C++
 - *Speed and efficiency*
 - However we can write a game in any language we wish

Games as Systems

- Games are *soft real-time interactive agent-based computer simulations*
- A Subset of the real world / an imaginary world
 - Modelled mathematically
 - Approximation, simplification
 - Numerical over analytical – can determine the next state of the system
- Simulation
 - Agent-based - Distinct entities interact autonomously
 - Temporal - The model of the world is dynamic, changes over time
 - Interactive - Respond to user interactions
- Deadline driven
 - The screen has to be updated 30 times per second
 - Soft - Missing a deadline is not catastrophic

Traditional Separation of Work

- Game Engine
 - Software, written by programmers
- Rules and Mechanics
 - Created and scripted by designers, with help from programmers
- User Interface
 - Coordinated with programmer, HCI specialist
- Content
 - Created by designers using tools

Game Engines - Systems

- A collection of sub-systems
- Physics
 - An example of a game *sub-system*
 - Specifies the space of possibilities for a game
 - Not the specific parameters of elements
 - Separates programmer from game designer
 - Programmer creates the engine
 - Designer fills in the parameters
- Defines physical attributes of the world
 - There is a gravitational force
 - Objects may have friction
 - Light may reflect in the following ways
- Does not define precise values or effects
 - The direction or value of gravity
 - Friction coefficients for each object
 - Specific lighting for each material

Game Engines - Characteristics

- Broad, adaptable and extensible
 - Encodes all ***non-mutable*** design decisions
 - Parameters for all ***mutable*** design decisions
- Outlines gameplay *possibilities*
 - Cannot be built independent of game design
 - But only needs high level information
- Data driven design
 - Ideal: No code outside the engine
 - Everything else is data
 - Create game content using editors / authoring tools
 - Art, music in standard file formats
 - Object, level data in XML (e.g.)
 - Character behaviour specified via scripts

Rules and Mechanics

- Fills in the values for the system
 - Parameters
 - Gravity, hit points, power
 - Types of player abilities, mechanics
 - Types of world interactions
 - Types of obstacles, challenges
- Does not include specific challenges
 - Set of challenges that *could* exist
 - Contents of the *pallet* for the level editor

Interfaces

- Engine and mechanics
 - ...describe what the player can do
 - ...do not specify *how* it is done
- Interface specifies
 - How a player does things
 - Player -> computer
 - How a player receives feedback
 - Computer -> player
- Bad interfaces can ruin a game
 - Strive for metaphorical “invisible interface”
 - Flow / immersion



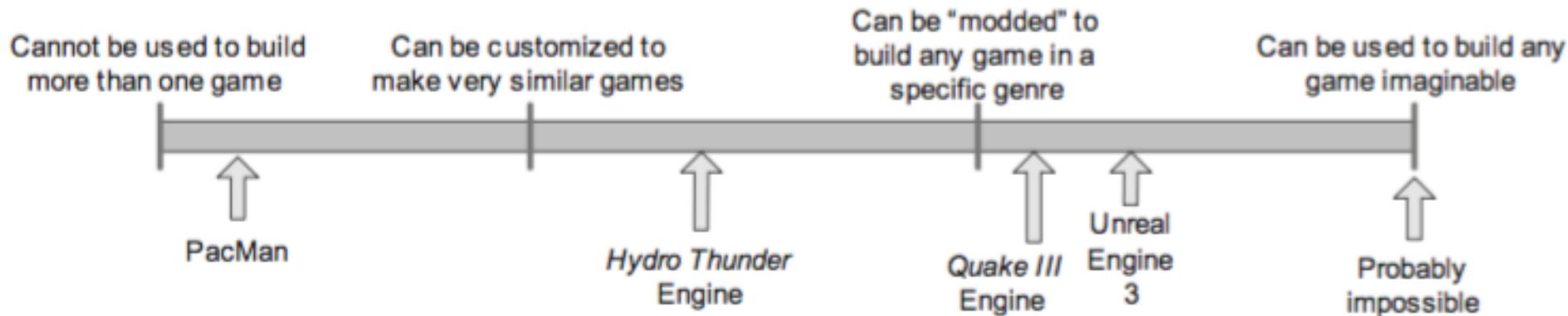
Content

- Content is *everything else*
- Gameplay content defines the actual game
 - Goals and victory conditions
 - Missions and quests
 - Levels
 - Interactive story choices
- Non-gameplay content that affects player experience
 - Graphics and cut-scenes
 - Sound effects and background music
 - Non-interactive story

Why the division?

- These components are not developed sequentially
 - Content may require changes to the game engine
 - Interface may change throughout
- A way of organising the design
 - Engine
 - Decisions to be made early on, hard-coded
 - *Unreal*
 - Mechanics
 - Mutable design decisions
 - Interface
 - How to shape the user experience
 - Content
 - Specific gameplay and level-design

Why Game Engines?

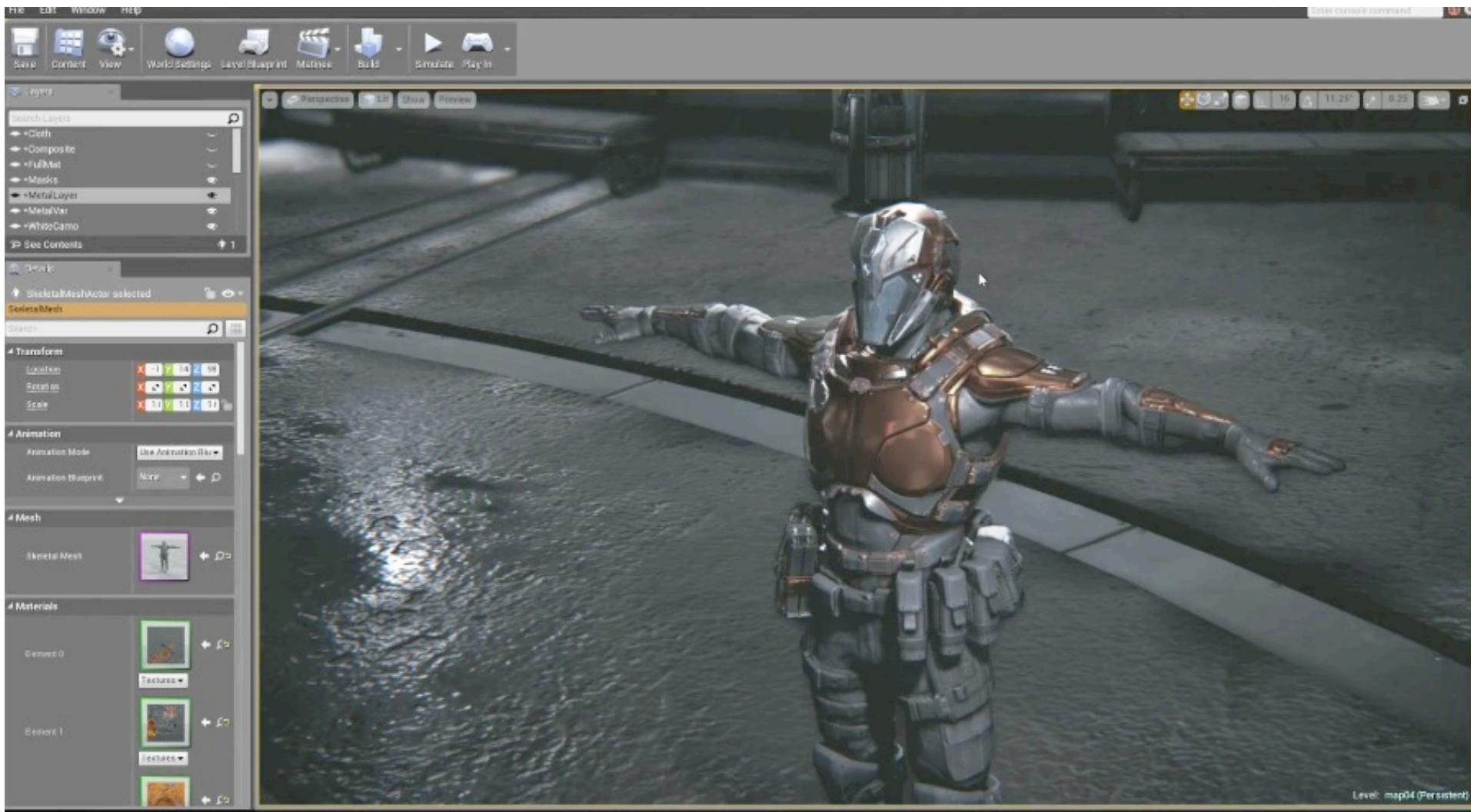


- Initially games were completely bespoke
 - Software specialised for individual application and hardware
- Now software aims to be fully featured, reusable and “kit” oriented
 - Robustness improves in the long term
 - Faster to develop new games
 - Abstraction over multiple platforms
 - Secondary revenue from licensing game engines
 - But do games made with the same engine all look alike?

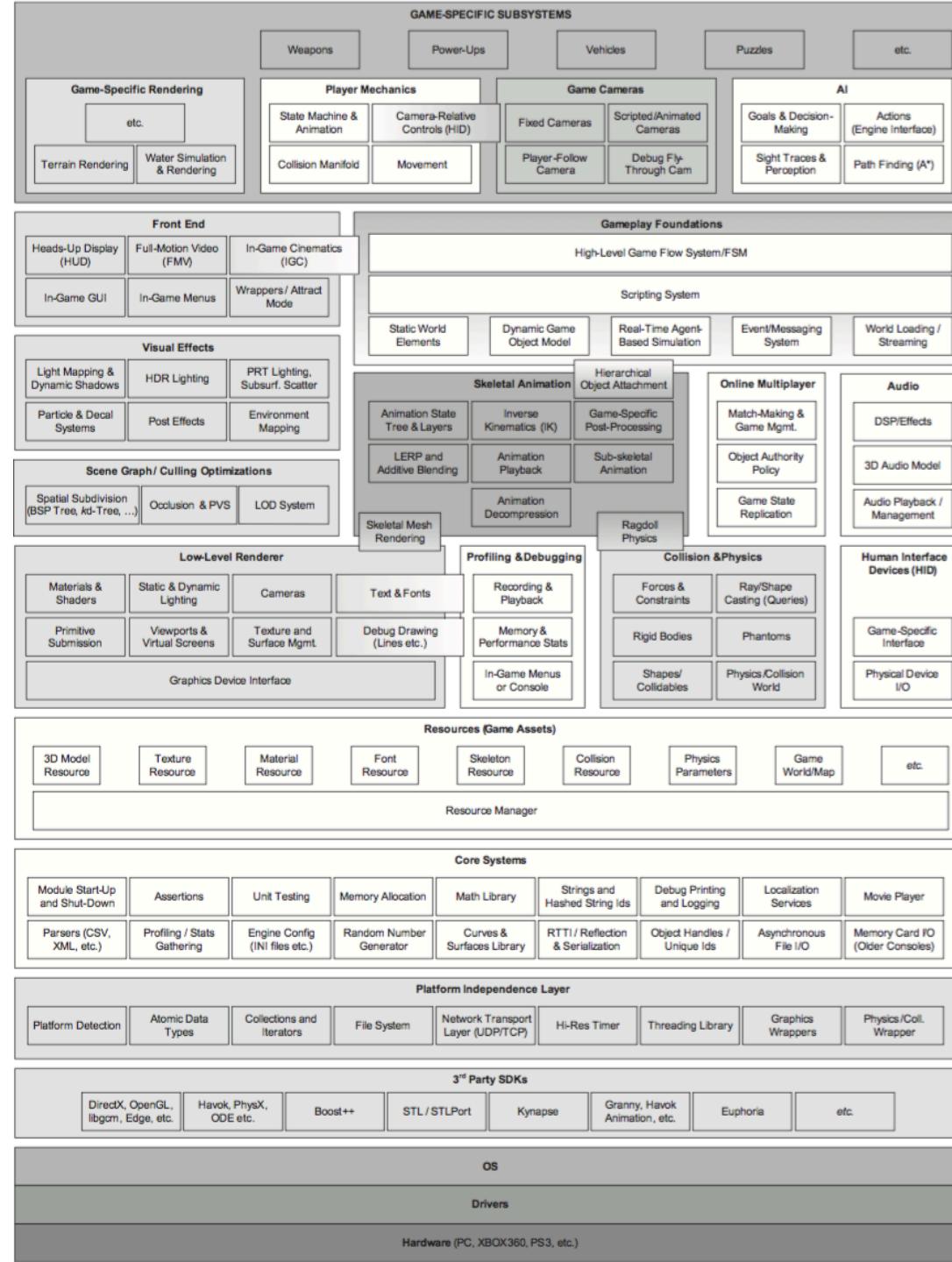
Game Engines across Genres

- Engines are typically *genre specific*
 - But with a large overlap – e.g. user input, visual rendering
- FPS
 - Efficient rendering, physics-based animation, AI of NPCs
- Platformers
 - Dynamic world, animated sprite characters, collision detection
- Fighting games
 - Large set of animations, accurate user input, character animation
- Racing games
 - Level of detail, rendering, rigid body physics and deformations
- RTS
 - Crowds, evolving environment, complex AI
- MMORPGS
 - Large scale network support (servers), data optimisation, persistent worlds

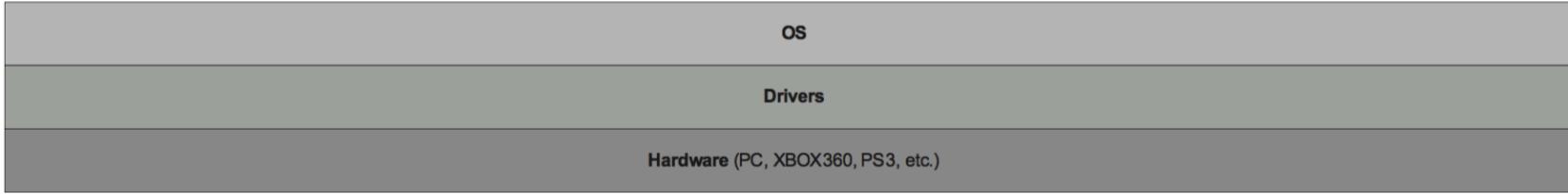






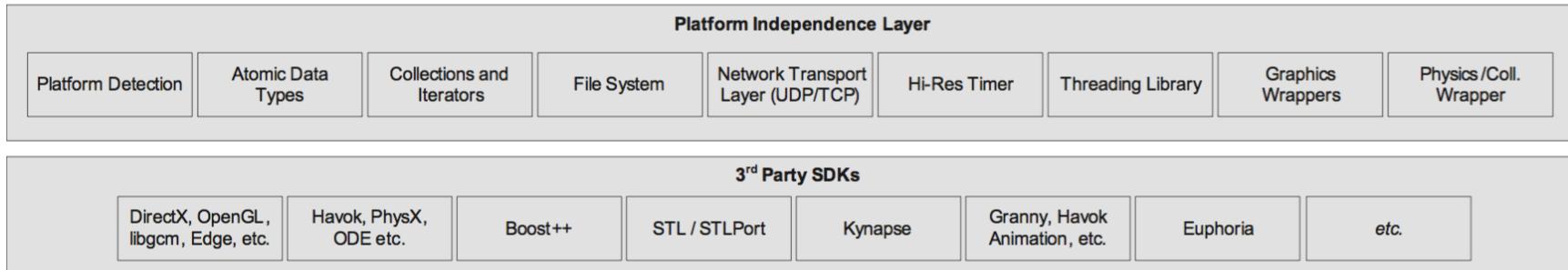


OS/Hardware layer



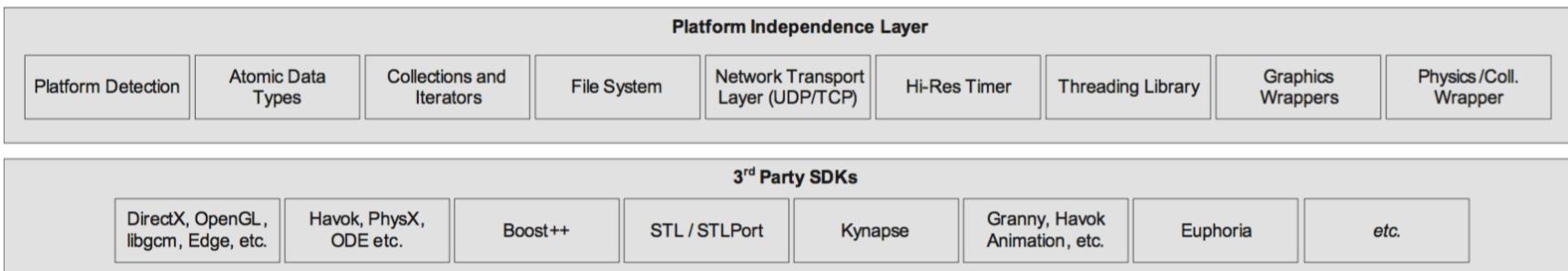
- **Hardware**
 - PC, PS2,3,4, Xbox,360,One, Nintendo...
 - Graphics card
 - Sound card
 - (Physics card?)
 - Input devices
 - Keyboard, mouse, game pad, controller, camera...
- **Drivers**
 - Low level interface to the above
 - Provided by hardware vendor
- **OS**
 - Time-sliced (Windows, Linux)
 - Console – thin library layer compiled into the game

3rd party SDKs



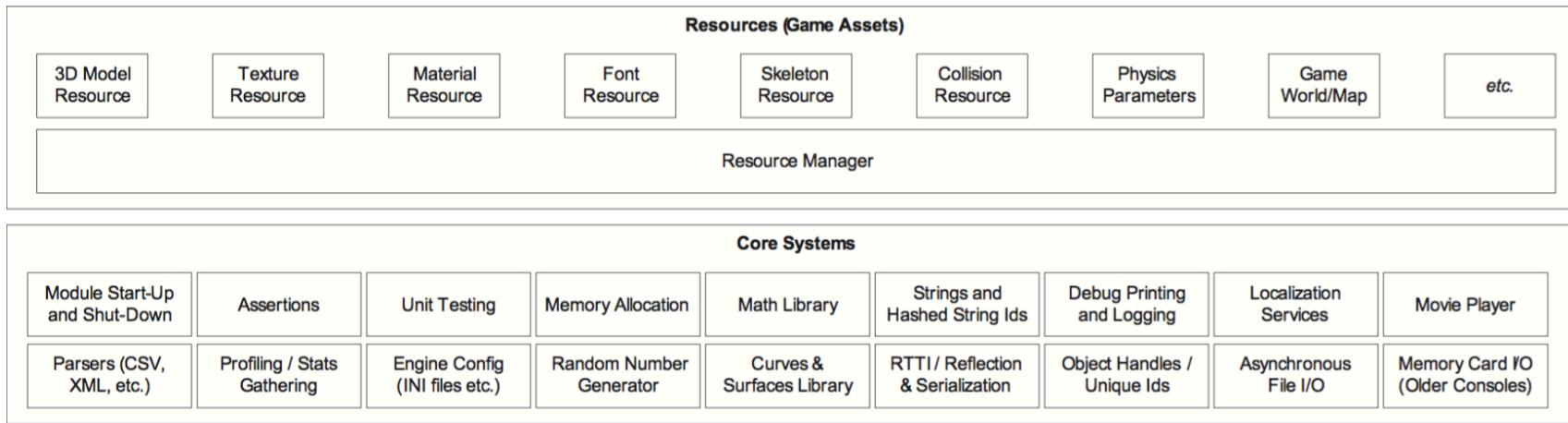
- Hardware interface libraries
 - OpenGL - Widely used, multiplatform
 - DirectX - Microsoft's 3D graphics SDK
 - Libgcm + Edge - PS3 graphics interface
- Physics
 - Havok, PhysX
- Data structures and algorithms
 - Boost, STL

Platform Independence



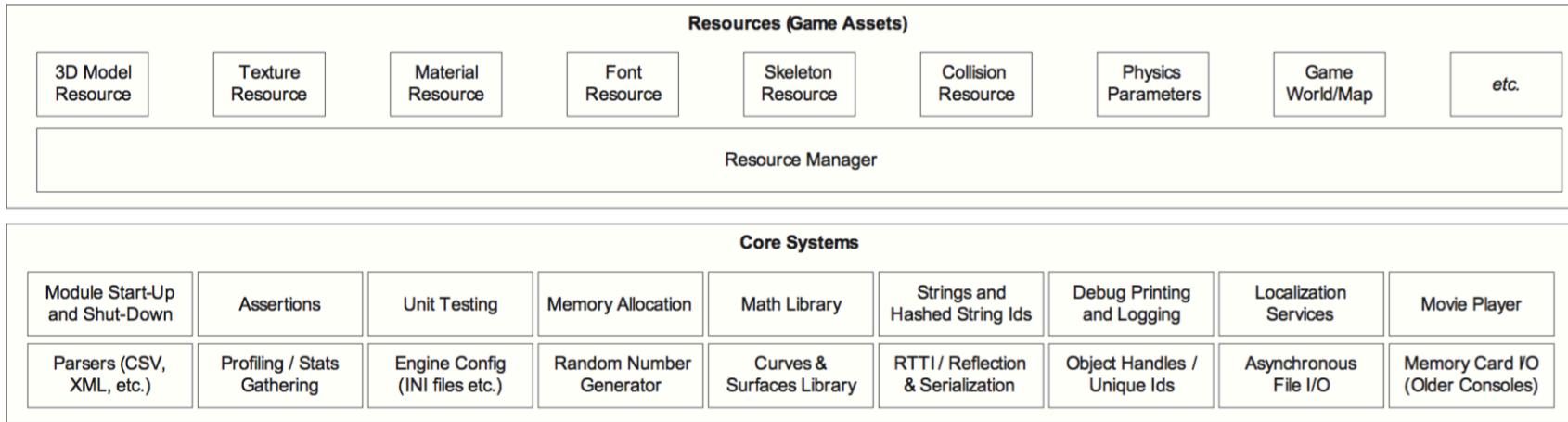
- Most game engines target multiple platforms
 - Notable exceptions are first-party studios
 - Naughty Dog / Sony
- Wrap / replace standard OS functions
 - Platform independent API
 - A lot of variation between platforms
 - Threading, Filesystem, Time, Network

Core Systems



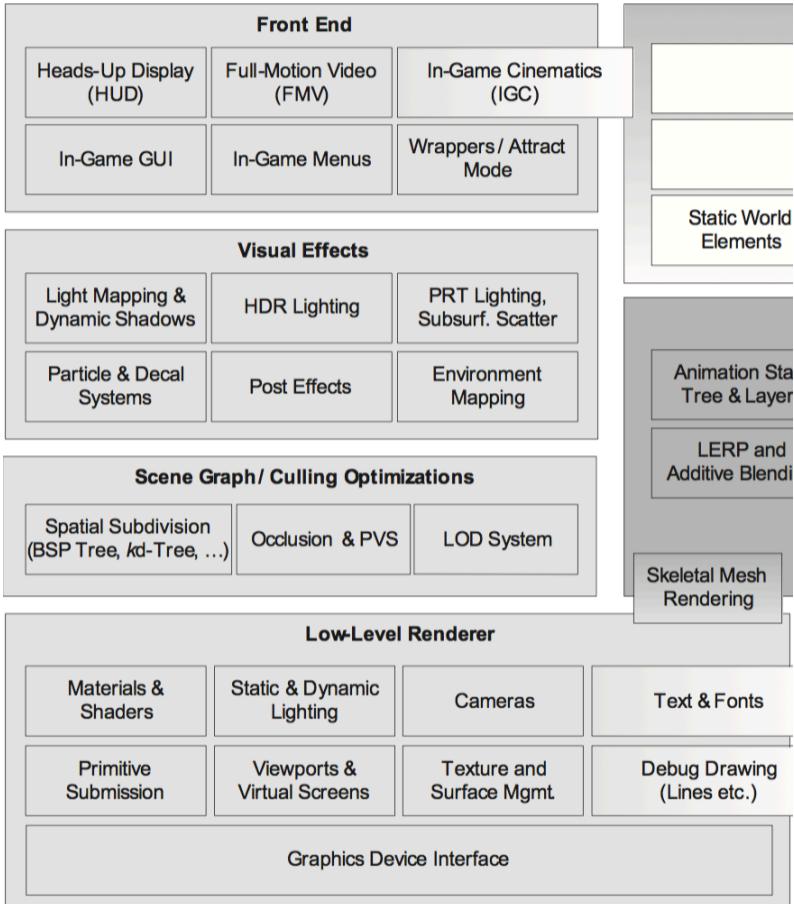
- Useful software utilities
 - Memory management
 - Localisation
 - Module start-up, shutdown
 - Configuration
 - RNG

Resources



- Unified interface for accessing assets
 - Abstraction over filesystem
- Packaged in .uasset, .pak, .pk3, .wad

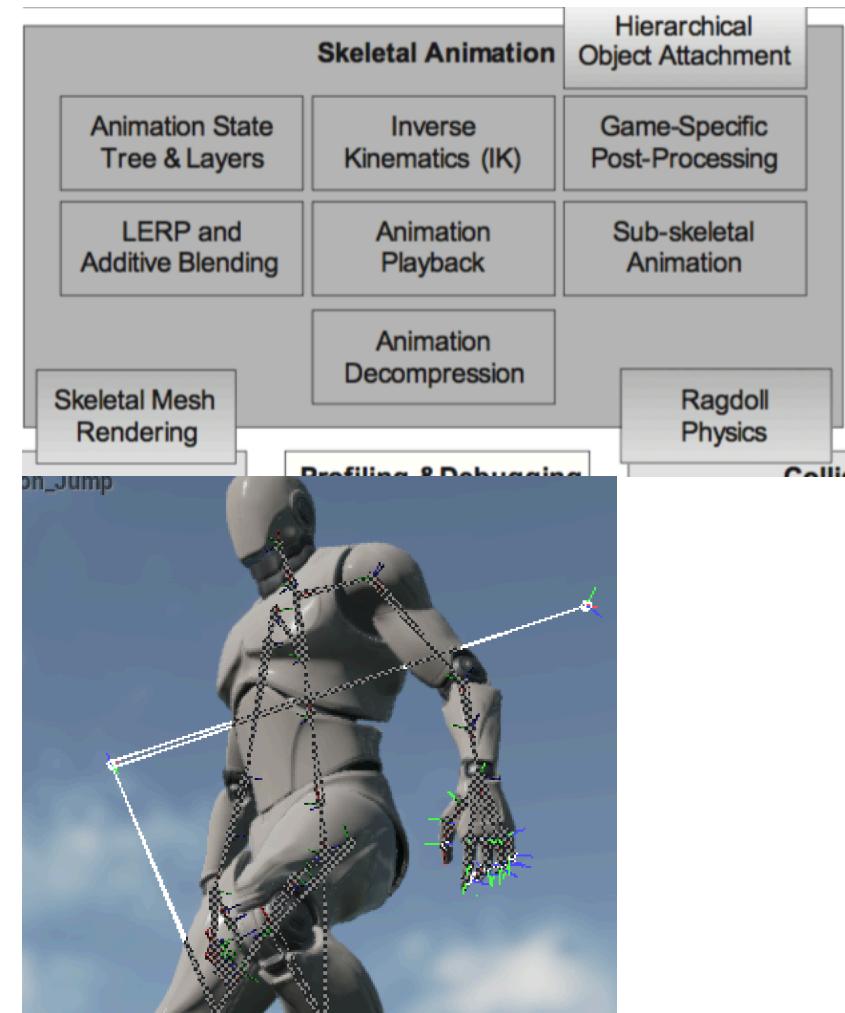
Graphics



- The visual representation of game data
 - HUD, GUI, 2d interfaces
 - Fancy visual effects
 - Decide what to draw
 - Frustum culling, spatial subdivision
 - Draw geometric primitives as quickly / richly as possible
 - Set up drawing context

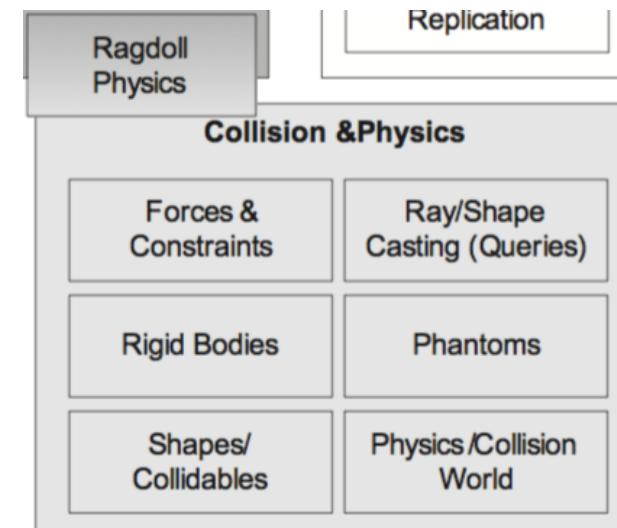
Skeletal Animation

- Pose a 3D character mesh using a simple system of bones
 - "Skin"
- Bones driven
 - By animation
 - By inverse kinematics
 - Figure out where the joints should be to put the foot *here*
- Blend animations together
 - Shoot while jumping

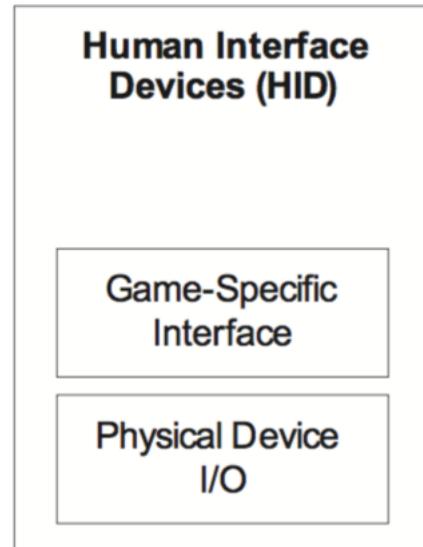


Collisions and Physics

- Handles the simulation of the world
 - Physical behaviours
 - Gravity, laws of motion
 - Kinematics, dynamics
 - Collisions
 - Ragdoll characters
 - Object breaking and destruction
- Physics is increasingly tightly integrated into gameplay
 - Physics-based animation
 - Interaction with objects using physics
 - But limited in simple games
- Physics is hard
 - Off-the-shelf sub-systems – Havok, ODE, PhysX



HID – Human Interface Devices



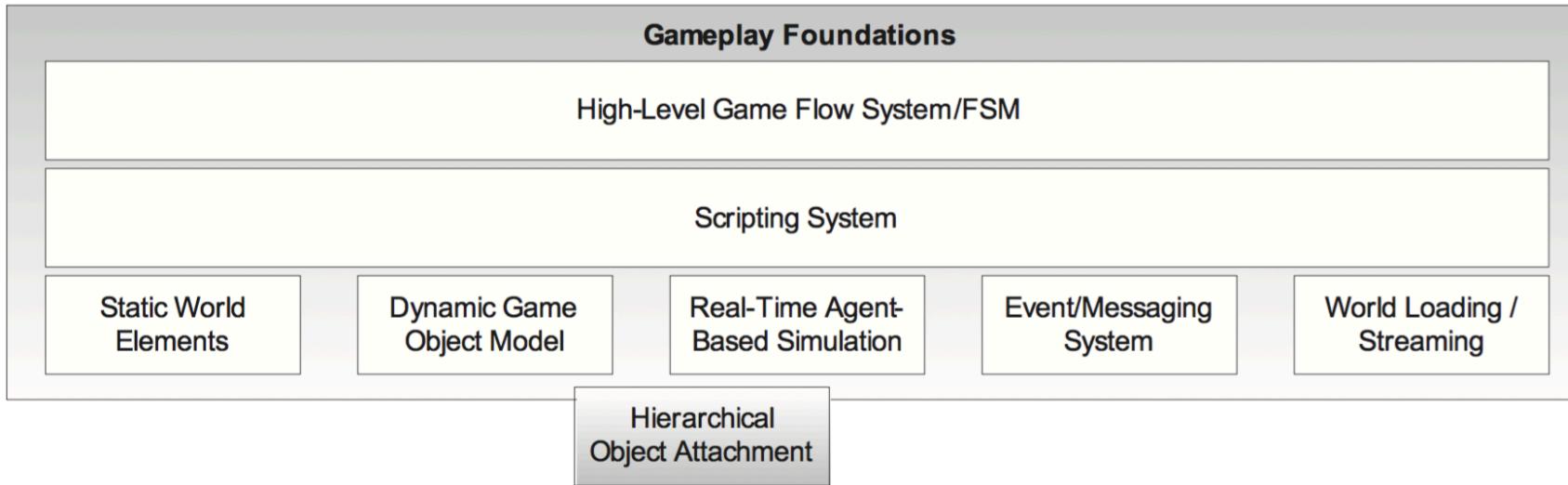
- Abstraction between
 - Low level controller
 - PS4, Xbox controllers
 - Keyboard, mouse
 - High level game actions
 - Jump, fire, move
- Player I/O
 - Includes haptic feedback

Networking

- To allow multiple players to play together within a share world
 - Networked multiplayer
 - Massively multiplayer
- Transfer data between multiple machines
 - How to share data?
 - What data?
 - How to enforce consistency?
- Has a profound impact on the design of a game engine
 - On world modeling, rendering, animation, input, objects

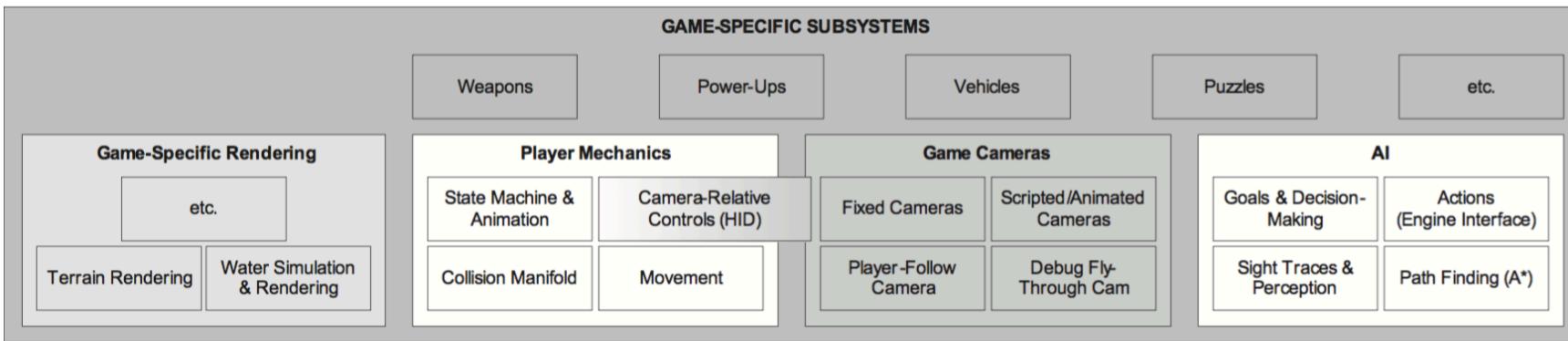


Gameplay / Game State



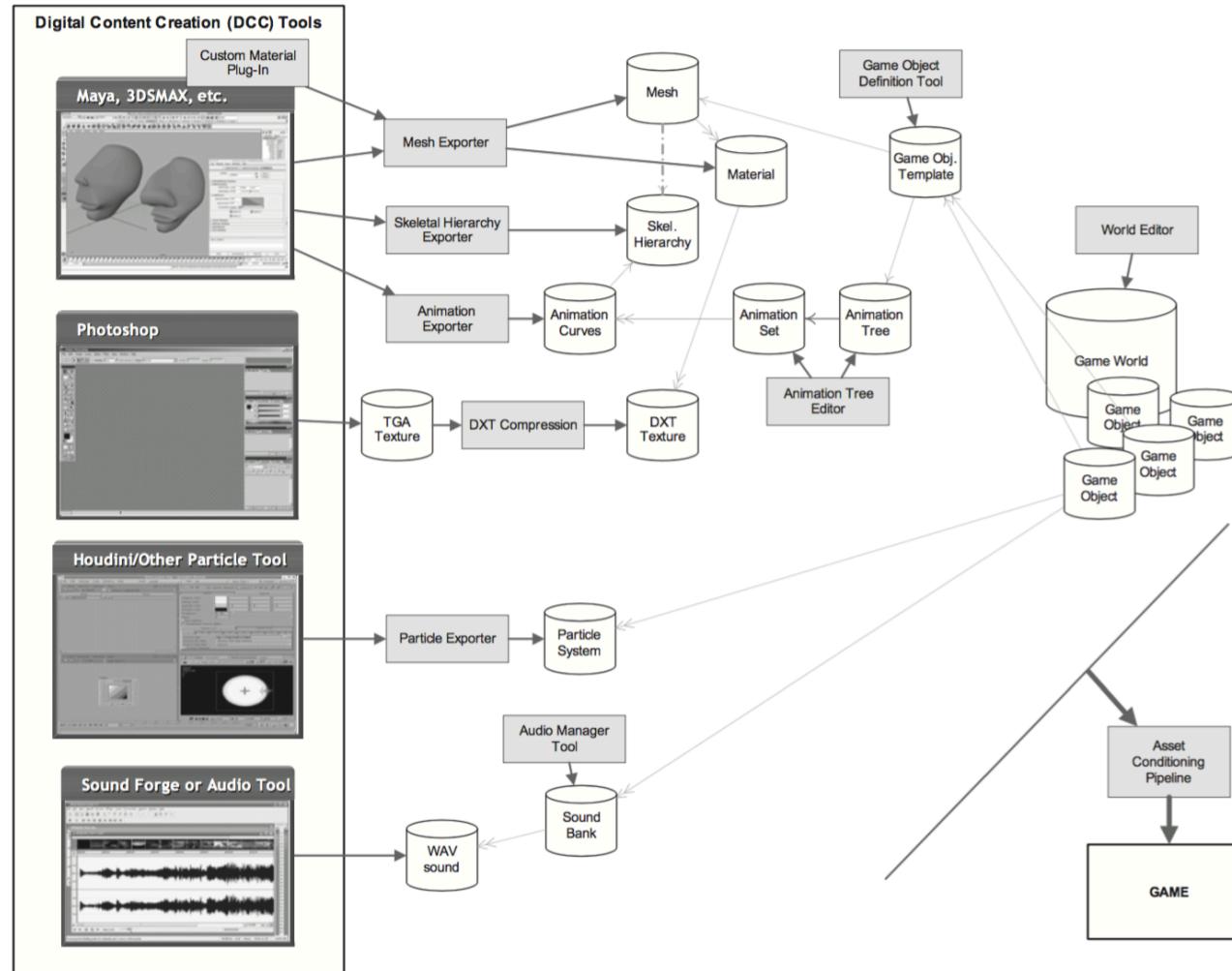
- A game-object model
 - The “state” of the game
 - How is the simulation driven (onTick)
 - What are the core game objects?
 - Actors, created / destroyed
- Pathfinding (navmesh), scripting (blueprint), events
- Levels and game modes
- Promotes “data driven design”

Game Specific



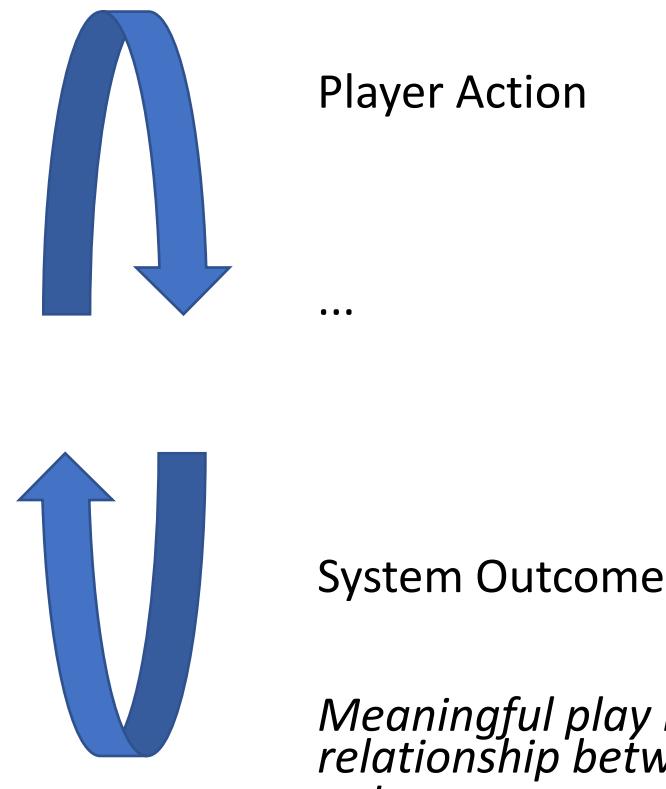
- The line between engine / game
 - Core mechanics
 - Game objects
 - AI Behaviours
 - Implementation of rules
 - ...

Asset Pipeline



How do we put it all together?

- User interface
 - Configuration and selection
 - Help
 - HID / HUD / UI
- Game Logic
 - Loading
 - Script
 - Physics Engine
 - Artificial Intelligence
 - Events
 - Collisions
 - Network communication
- Outputs
 - Graphics renderer
 - Sound and music



References

- Game Engine Architecture, Jason Gregory 2014, chapter 1