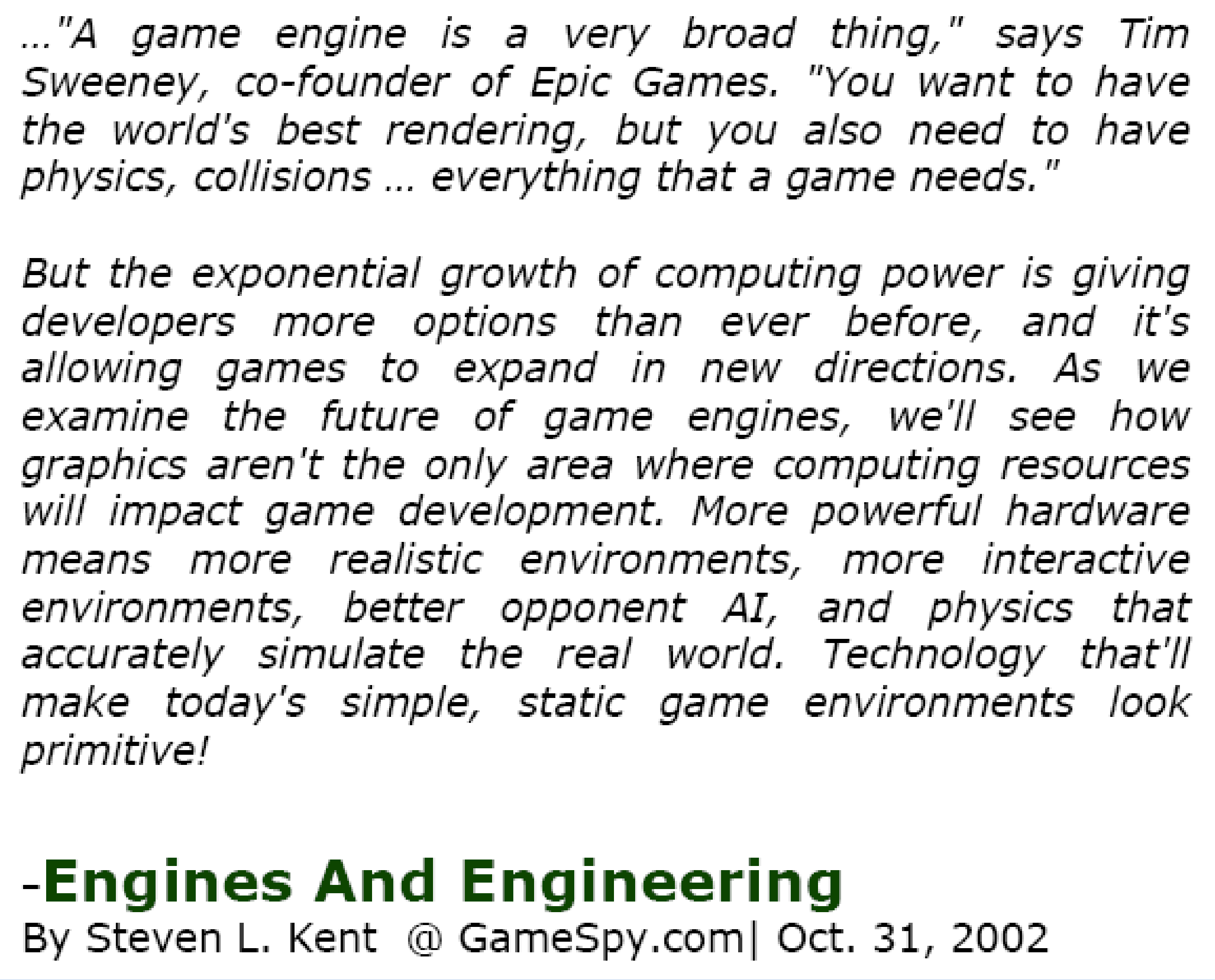
Introduction

An Overview of Game Engines

Overview

- What is a game engine?
- Evolution of game engines
- Game engine features and differences
- Game loop and timelines

What is a Game Engine?



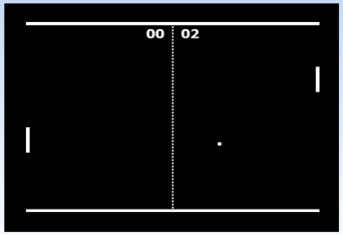
…"A game engine is a very broad thing," says Tim Sweeney, co-founder of Epic Games. "You want to have the world's best rendering, but you also need to have physics, collisions … everything that a game needs."

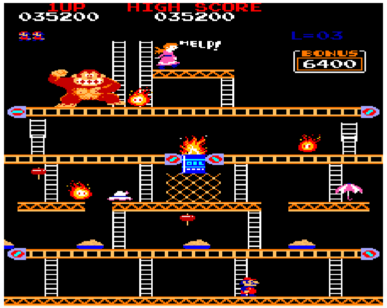
But the exponential growth of computing power is giving developers more options than ever before, and it's allowing games to expand in new directions. As we examine the future of game engines, we'll see how graphics aren't the only area where computing resources will impact game development. More powerful hardware means more realistic environments, more interactive environments, better opponent AI, and physics that accurately simulate the real world. Technology that'll make today's simple, static game environments look primitive!

-**Engines And Engineering**
By Steven L. Kent @ GameSpy.com| Oct. 31, 2002

What is a Game Engine?

- Evolution of video games
  - ➢ Games developed in 70s-80s
    - Relatively cheap to build
      - – Budget ~$1000 - $5000
    - Often designed from scratch by one programmer in a few weeks
  - ➢ Modern Games
    - Enormous implementation complexity and costs if developed from scratch
    - Video games vary widely in their gameplay and settings
      - – Share commonly required modules to support graphics, physics, user interface, etc.





Game Engines

- What is a game engine?
    - ➢ The core software of the game
        - Describes a set of code used to build the game application
        - Software suites that provide the technological underwiring for games
            - – Everything you see on the screen and interact with within the game world is powered by the game engine
        - A framework comprised of a collection of different tools, utilities, and interfaces that hide the low-level details of the various tasks that make up the game
        - Exists to abstract the details of doing common game-related tasks (e.g. rendering, physics, input) so that developers can focus on the details that make their games unique

Game Engines

John Carmack

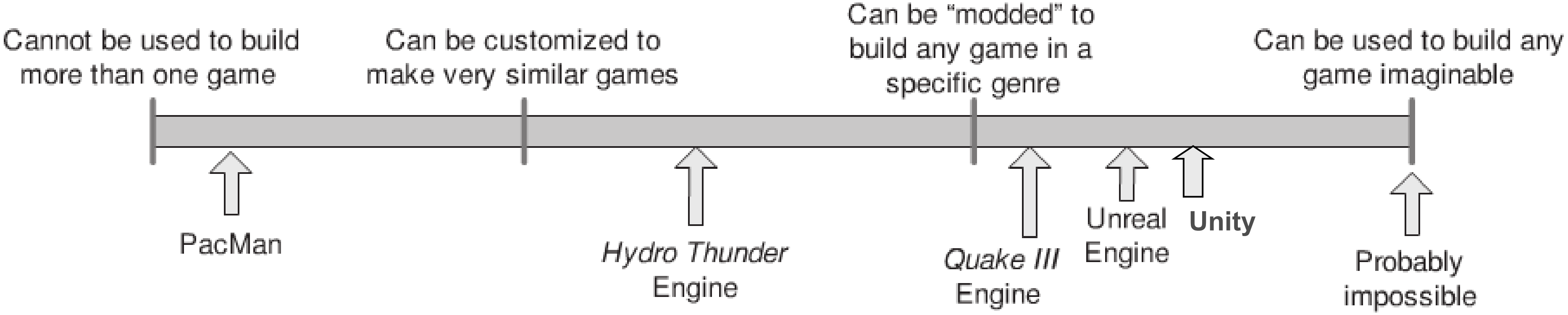
- Game engine
  - ➢ A term that was introduced in the mid-1990s in reference to FPS games like id Software's Doom
    - The game had reasonably well-defined separations between core software components, game assets, rules, etc.
  - ➢ The value of separation became evident
    - When developers began licensing games with well-structured architecture and using them to develop their own games
      - – New art, world layouts, weapons, characters, vehicles, rules, etc. However, only minimal changes to the engine software
    - Success of the game engine concept resulted in highly customisable engines easily extendable using scripting languages

Game Engines

- Game engine
  - ➢ No clear boundary between the game and game engine
    - Hard-coded game logic or game rules make it difficult to reuse that software for other games
  - ➢ Ideally, a game engine should be extensible
    - Can be used as the foundation for many different games without major modification
    - However, many trade-offs involved
      - – Depends on what the engine will be used for
        - » E.g. different genres have different characteristics and focus
      - – Trade-off between generality and optimality
  - ➢ Data-driven architecture is what differentiates a game engine from a program that is a game but not an engine

Game Engines

- Game engine
  - ➢ Reusability gamut



Cannot be used to build more than one game

Can be customized to make very similar games

Can be "modded" to build any game in a specific genre

Can be used to build any game imaginable

PacMan

Hydro Thunder Engine

Quake III Engine

Unreal Engine

Unity

Probably impossible

Game Engines

- Evolution of game engines
  - ➢ Early game companies used to make their own game engines
  - ➢ Kept in-house and updated as technology improved
  - ➢ Early examples
    - SCUMM (LucasArts) - 1987
    - SCI (Sierra) - 1988
  - ➢ Later on
    - id Tech (Quake series) - 1996
      - – Source engine, used for Half-Life
    - Unreal Engine
      - – v1 (1998), v2 (2002), v3 (2004), v4 (2012), v5 (2022)

Game Engines

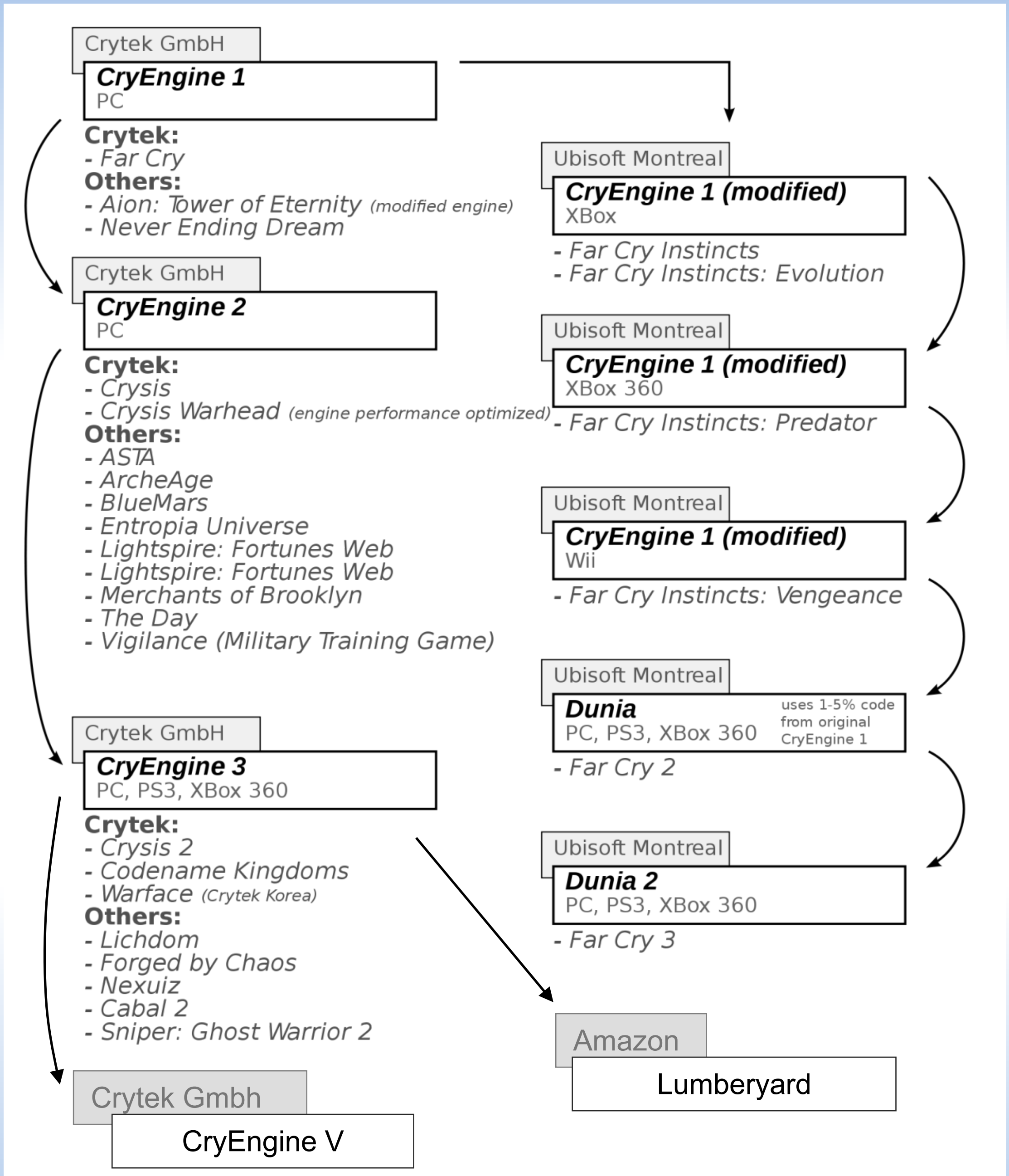
- Evolution of game engines
  - ➢ As the cost of making engines grew
    - More and more companies began to specialise in making game engines or game engine components
  - ➢ These companies are known as middleware providers
  - ➢ Creates a ‘build versus buy’ decision

"I think the trend is going more towards buy than build"

Joanna Alexander
Co-founder of Zombie Interactive,
developer of *Spec Ops*

Game Engines

- Evolution of game engines
  - ➢ Evolution and family tree



Crytek GmbH
CryEngine 1
PC

Crytek:
- *Far Cry*

Others:
- *Aion: Tower of Eternity* (modified engine)
- *Never Ending Dream*

Crytek GmbH
CryEngine 2
PC

Crytek:
- *Crysis*
- *Crysis Warhead* (engine performance optimized)

Others:
- *ASTA*
- *ArcheAge*
- *BlueMars*
- *Entropia Universe*
- *Lightspire: Fortunes Web*
- *Lightspire: Fortunes Web*
- *Merchants of Brooklyn*
- *The Day*
- *Vigilance (Military Training Game)*

Crytek GmbH
CryEngine 3
PC, PS3, XBox 360

Crytek:
- *Crysis 2*
- *Codename Kingdoms*
- *Warface* (Crytek Korea)

Others:
- *Lichdom*
- *Forged by Chaos*
- *Nexuiz*
- *Cabal 2*
- *Sniper: Ghost Warrior 2*

Crytek Gmbh
CryEngine V

Ubisoft Montreal
CryEngine 1 (modified)
XBox
- *Far Cry Instincts*
- *Far Cry Instincts: Evolution*

Ubisoft Montreal
CryEngine 1 (modified)
XBox 360
- *Far Cry Instincts: Predator*

Ubisoft Montreal
CryEngine 1 (modified)
Wii
- *Far Cry Instincts: Vengeance*

Ubisoft Montreal
Dunia
PC, PS3, XBox 360
uses 1-5% code from original CryEngine 1
- *Far Cry 2*

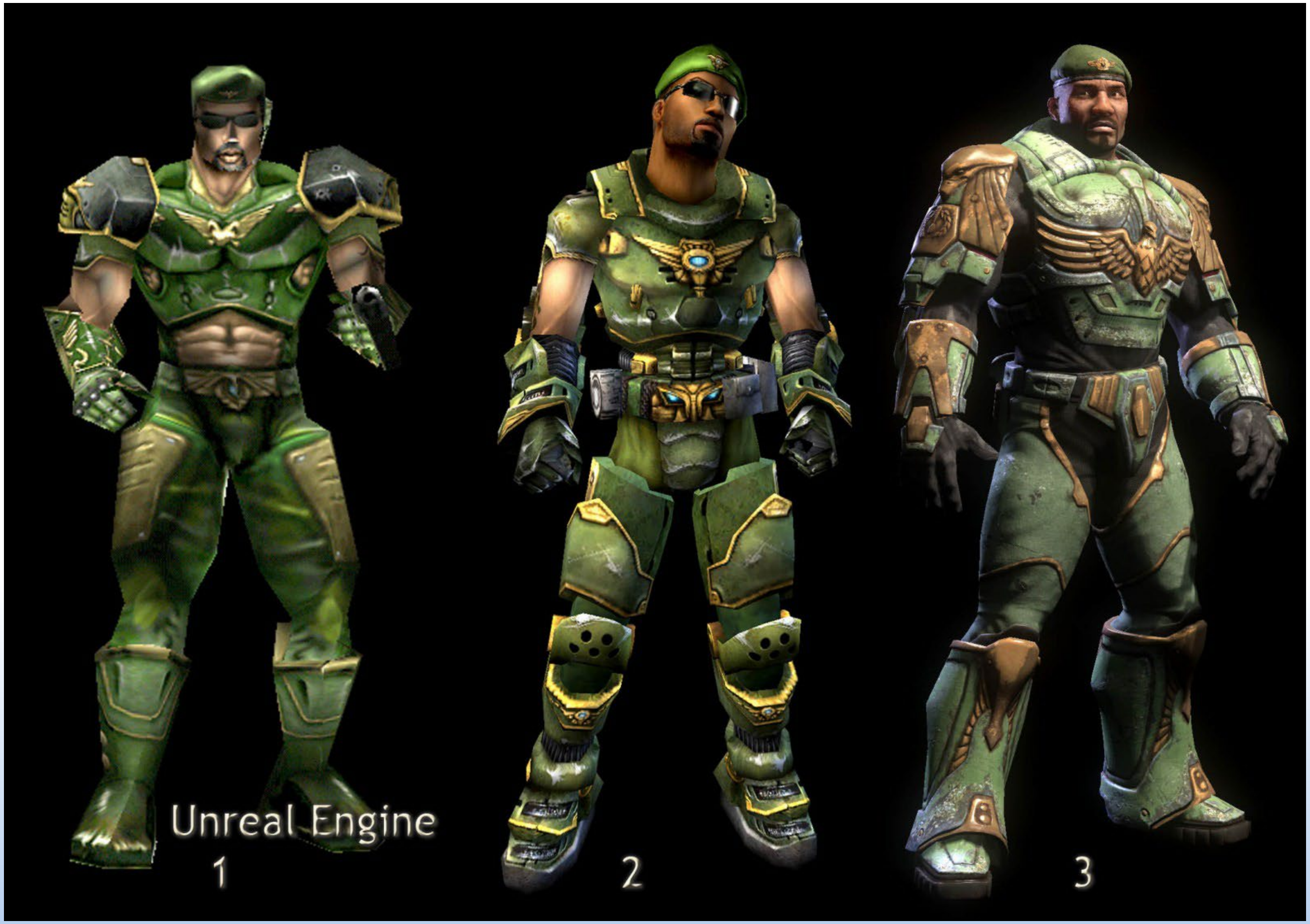
Ubisoft Montreal
Dunia 2
PC, PS3, XBox 360
- *Far Cry 3*

Amazon
Lumberyard

http://en.wikipedia.org/wiki/File:Cryengine_tree.svg

Game Engines

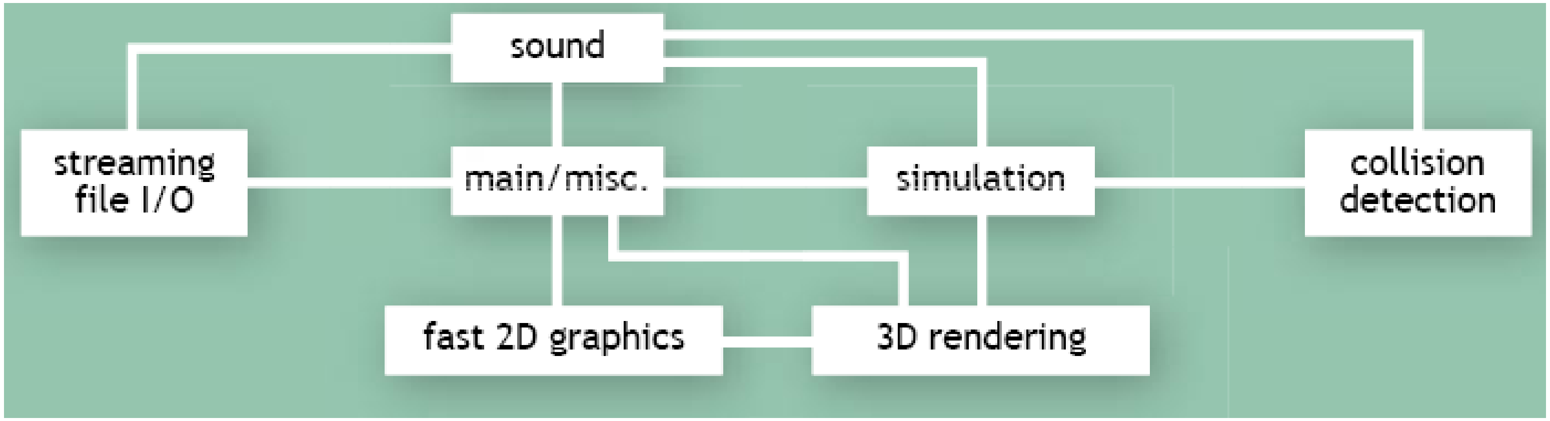
- Evolution of game engines

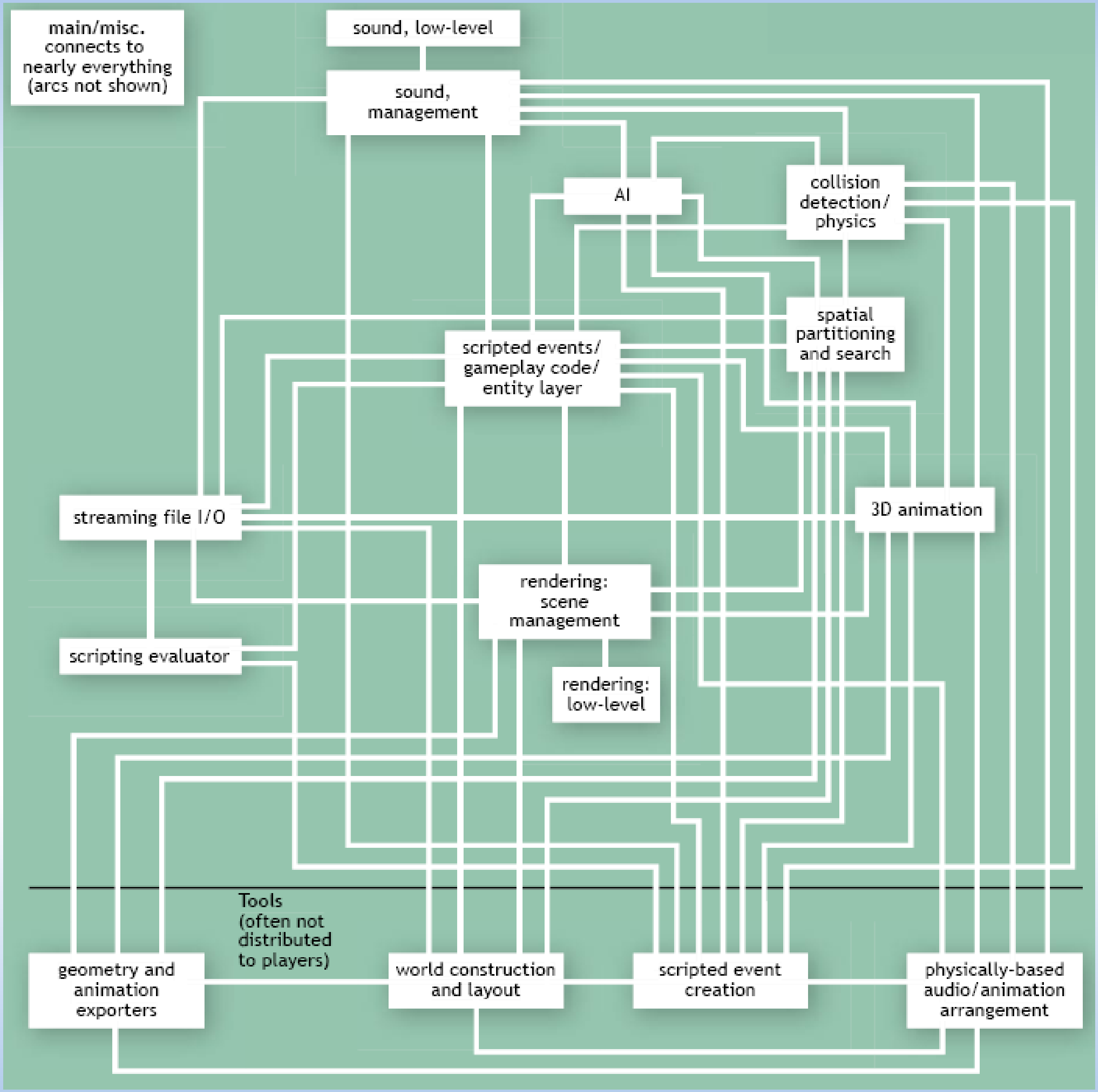


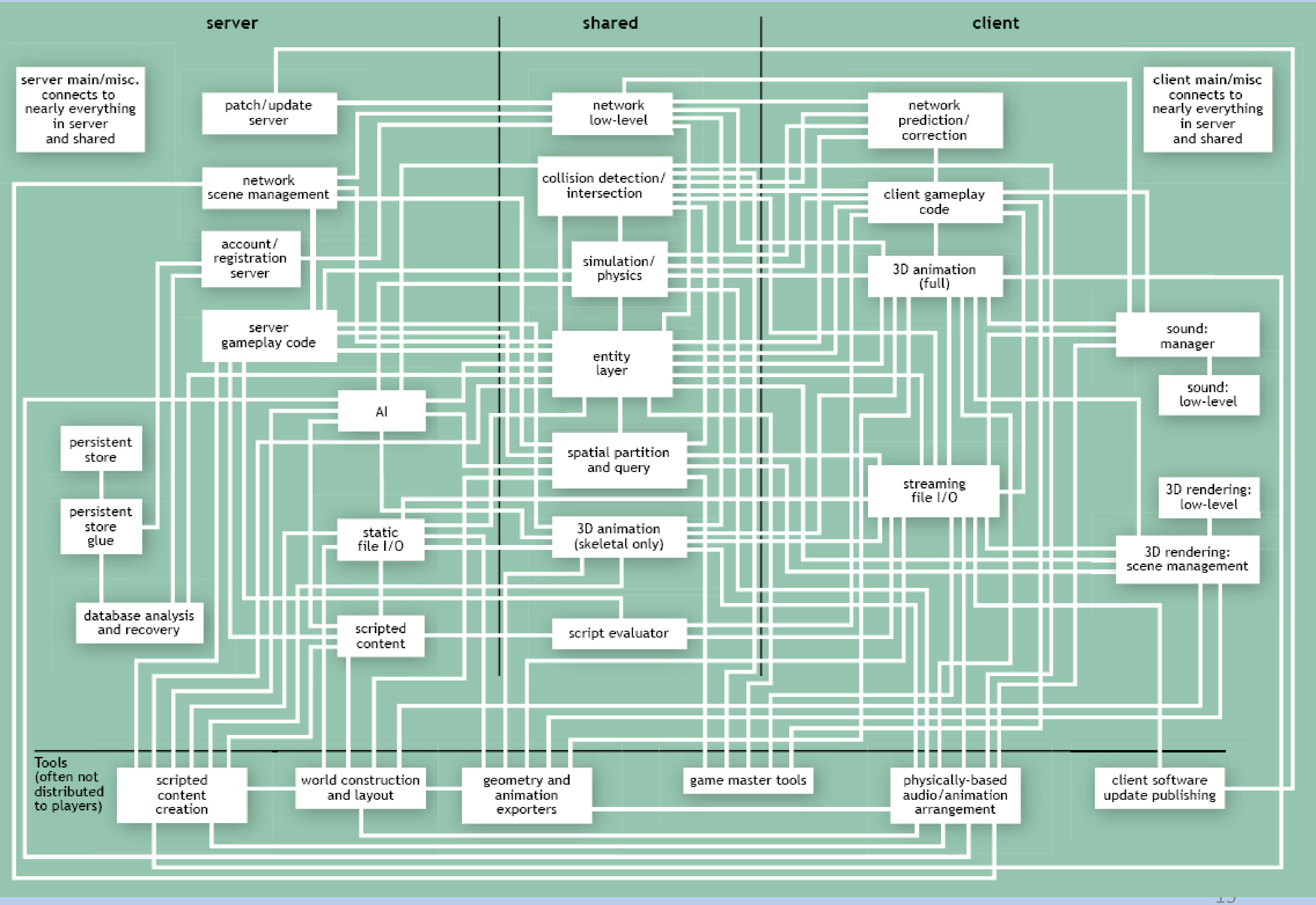
http://en.wikipedia.org/wiki/File:Unreal_Engine_Comparison.jpg

Game Engines

- Evolution of game engines




main/misc. connects to nearly everything (arcs not shown)
sound, low-level
sound, management
AI
collision detection/ physics
spatial partitioning and search
scripted events/ gameplay code/ entity layer
streaming file I/O
3D animation
rendering: scene management
rendering: low-level
scripting evaluator
Tools (often not distributed to players)
geometry and animation exporters
world construction and layout
scripted event creation
physically-based audio/animation arrangement



server

shared

client

server main/misc. connects to nearly everything in server and shared

patch/update server

network low-level

network prediction/ correction

client main/misc connects to nearly everything in server and shared

network scene management

collision detection/ intersection

client gameplay code

account/ registration server

simulation/ physics

3D animation (full)

server gameplay code

entity layer

sound: manager

sound: low-level

AI

persistent store

spatial partition and query

streaming file I/O

persistent store glue

static file I/O

3D animation (skeletal only)

3D rendering: low-level

3D rendering: scene management

database analysis and recovery

scripted content

script evaluator

Tools (often not distributed to players)

scripted content creation

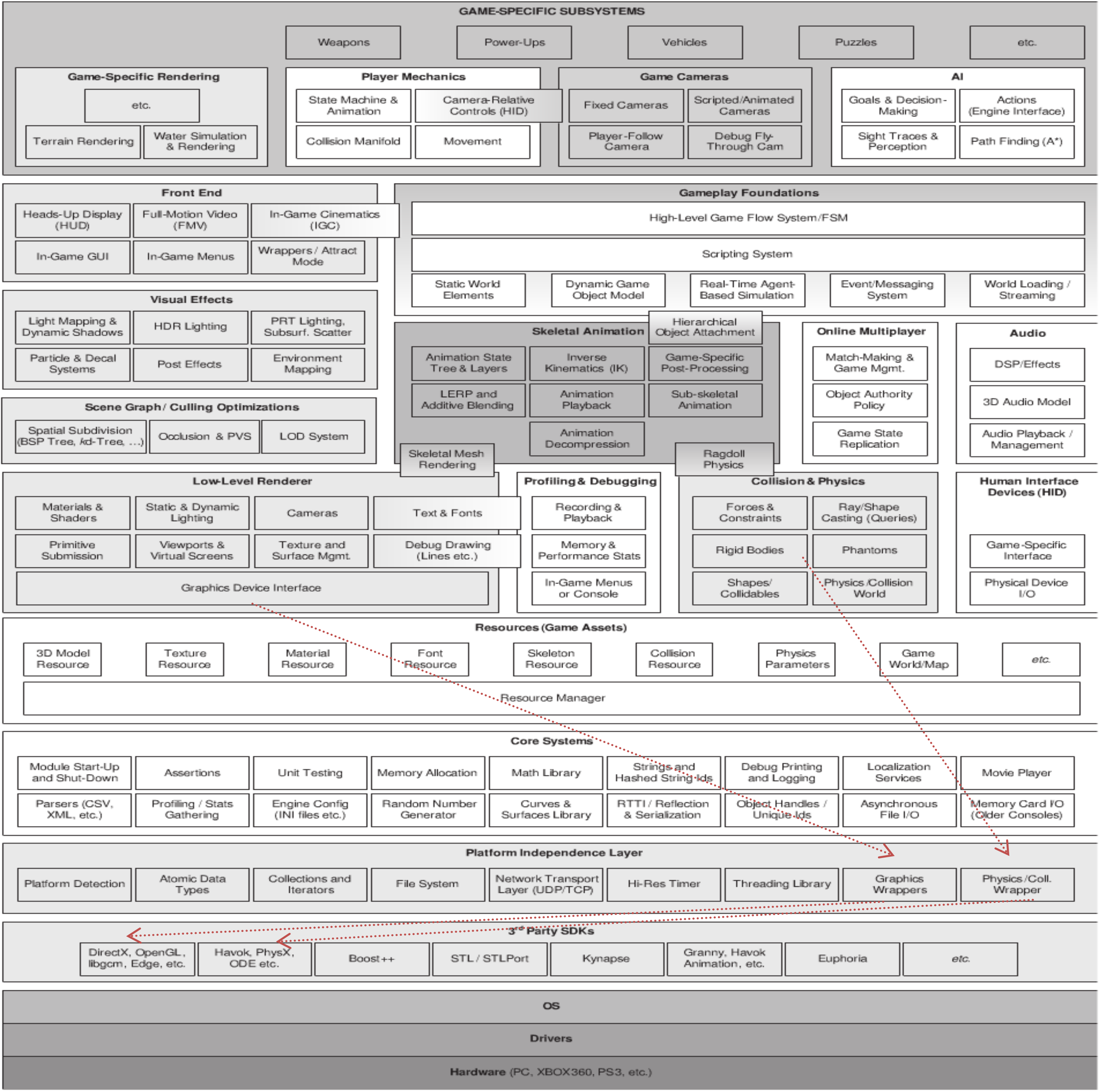
world construction and layout

geometry and animation exporters

game master tools

physically-based audio/animation arrangement

client software update publishing



GAME-SPECIFIC SUBSYSTEMS

Weapons | Power-Ups | Vehicles | Puzzles | etc.

Game-Specific Rendering: etc.; Terrain Rendering; Water Simulation & Rendering

Player Mechanics: State Machine & Animation; Camera-Relative Controls (HID); Collision Manifold; Movement

Game Cameras: Fixed Cameras; Scripted/Animated Cameras; Player-Follow Camera; Debug Fly-Through Cam

AI: Goals & Decision-Making; Actions (Engine Interface); Sight Traces & Perception; Path Finding (A*)

Front End: Heads-Up Display (HUD); Full-Motion Video (FMV); In-Game Cinematics (IGC); In-Game GUI; In-Game Menus; Wrappers / Attract Mode

Gameplay Foundations: High-Level Game Flow System/FSM; Scripting System; Static World Elements; Dynamic Game Object Model; Real-Time Agent-Based Simulation; Event/Messaging System; World Loading / Streaming

Visual Effects: Light Mapping & Dynamic Shadows; HDR Lighting; PRT Lighting, Subsurf. Scatter; Particle & Decal Systems; Post Effects; Environment Mapping

Scene Graph / Culling Optimizations: Spatial Subdivision (BSP Tree, kd-Tree, …); Occlusion & PVS; LOD System

Skeletal Animation: Hierarchical Object Attachment; Animation State Tree & Layers; Inverse Kinematics (IK); Game-Specific Post-Processing; LERP and Additive Blending; Animation Playback; Sub-skeletal Animation; Animation Decompression; Skeletal Mesh Rendering; Ragdoll Physics

Online Multiplayer: Match-Making & Game Mgmt.; Object Authority Policy; Game State Replication

Audio: DSP/Effects; 3D Audio Model; Audio Playback / Management

Low-Level Renderer: Materials & Shaders; Static & Dynamic Lighting; Cameras; Text & Fonts; Primitive Submission; Viewports & Virtual Screens; Texture and Surface Mgmt.; Debug Drawing (Lines etc.); Graphics Device Interface

Profiling & Debugging: Recording & Playback; Memory & Performance Stats; In-Game Menus or Console

Collision & Physics: Forces & Constraints; Ray/Shape Casting (Queries); Rigid Bodies; Phantoms; Shapes/ Collidables; Physics/Collision World

Human Interface Devices (HID): Game-Specific Interface; Physical Device I/O

Resources (Game Assets): 3D Model Resource; Texture Resource; Material Resource; Font Resource; Skeleton Resource; Collision Resource; Physics Parameters; Game World/Map; etc.; Resource Manager

Core Systems: Module Start-Up and Shut-Down; Assertions; Unit Testing; Memory Allocation; Math Library; Strings and Hashed String Ids; Debug Printing and Logging; Localization Services; Movie Player; Parsers (CSV, XML, etc.); Profiling / Stats Gathering; Engine Config (INI files etc.); Random Number Generator; Curves & Surfaces Library; RTTI / Reflection & Serialization; Object Handles / Unique Ids; Asynchronous File I/O; Memory Card I/O (Older Consoles)

Platform Independence Layer: Platform Detection; Atomic Data Types; Collections and Iterators; File System; Network Transport Layer (UDP/TCP); Hi-Res Timer; Threading Library; Graphics Wrappers; Physics/Coll. Wrapper

3rd Party SDKs: DirectX, OpenGL, libgcm, Edge, etc.; Havok, PhysX, ODE etc.; Boost++; STL / STLPort; Kynapse; Granny, Havok Animation, etc.; Euphoria; etc.

OS

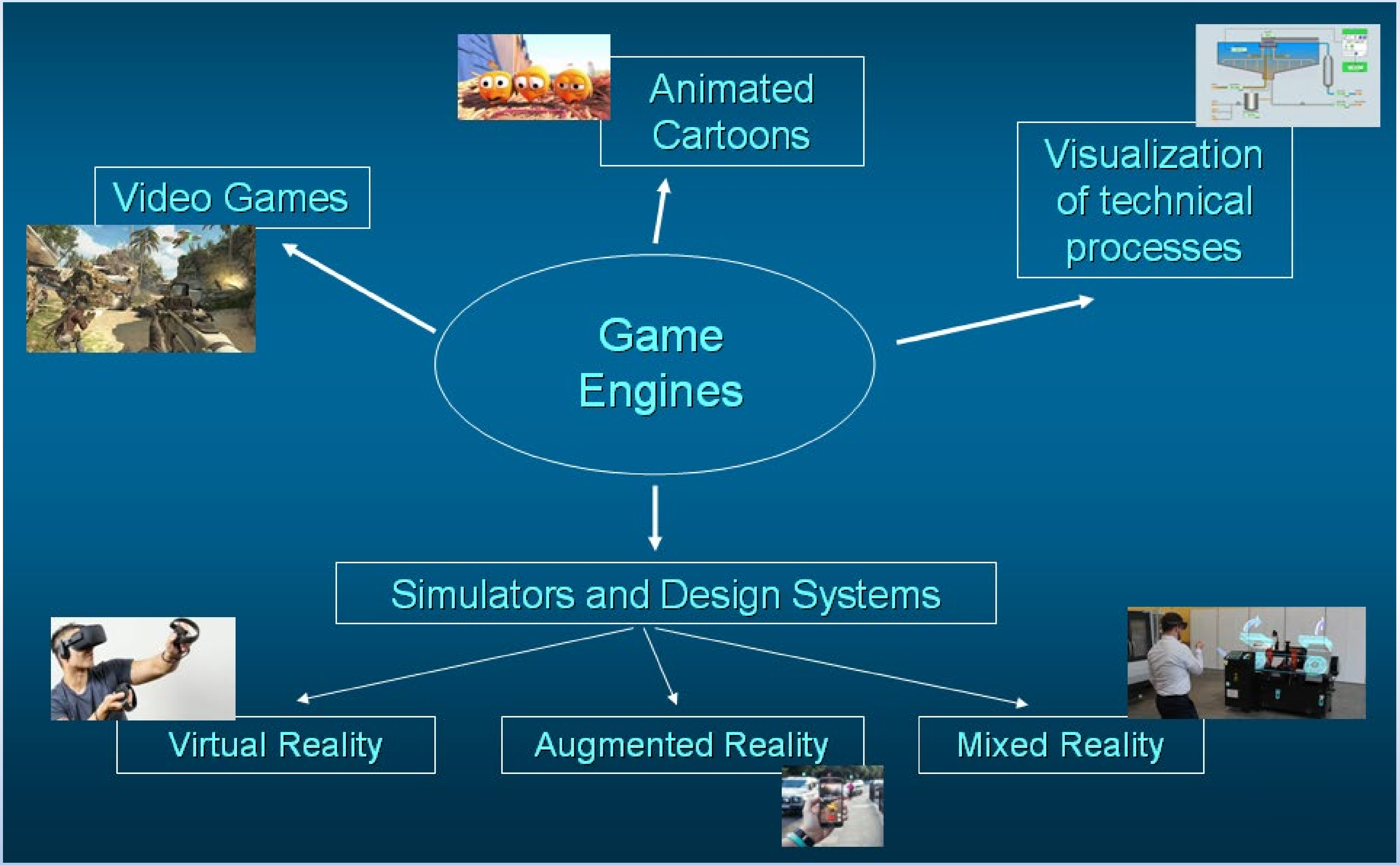
Drivers

Hardware (PC, XBOX360, PS3, etc.)

Game Engines

- Key components of a game engine might include
  - Graphics
  - Input system
  - Physics and collision
  - Artificial intelligence
  - Audio system
  - Networking
  - Other utilities

Game Engine Applications



Game Engines

Video Games

Animated Cartoons

Visualization of technical processes

Simulators and Design Systems

Virtual Reality

Augmented Reality

Mixed Reality

Game Engines

- Choosing a game engine
  - ➢ Each engine has its advantages and disadvantages
  - ➢ Depends on the developer’s needs in terms of budget, schedule, and the game itself
  - ➢ Several criteria to consider
    - Quality and performance
      - – Technologies used for rendering, animation, simulation of physics, AI, etc
    - Ease of use
      - – Each game engine has its own editor, user community forums usually have vigorous discussions about the plusses and minuses

Game Engines

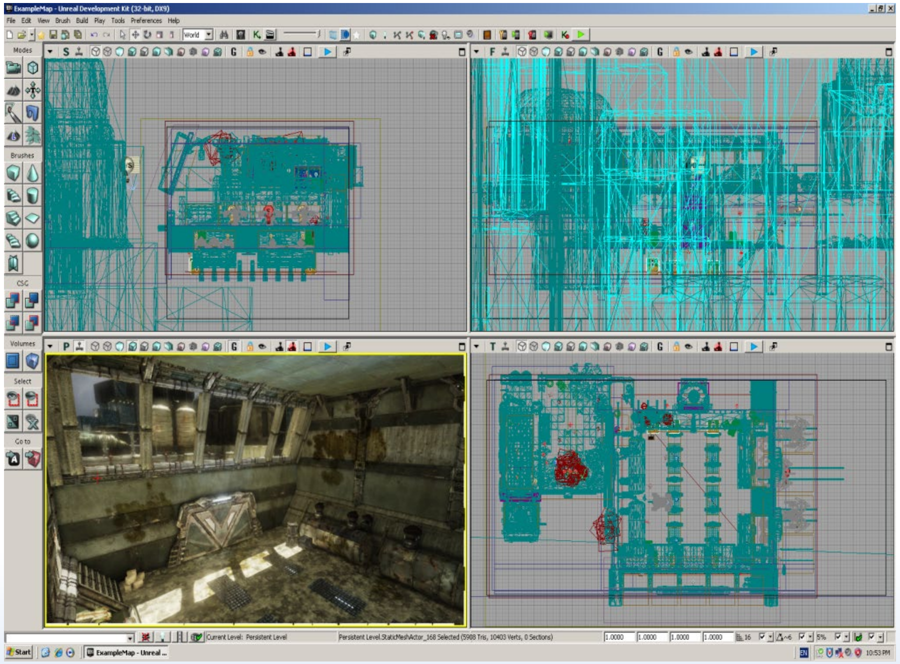
- Cross-platform capabilities
  - Depends on what platform the developer is targeting, the engine must support the platform(s)
- Look and feel
  - Games developed using a particular engine tend to resemble one another
- Support
  - Some engines receive ongoing support from creators and have active user communities
- Documentation
  - Engines without proper documentation are difficult to use
  - Takes a long time to figure out how to use it, and might not be able to maximise its capabilities

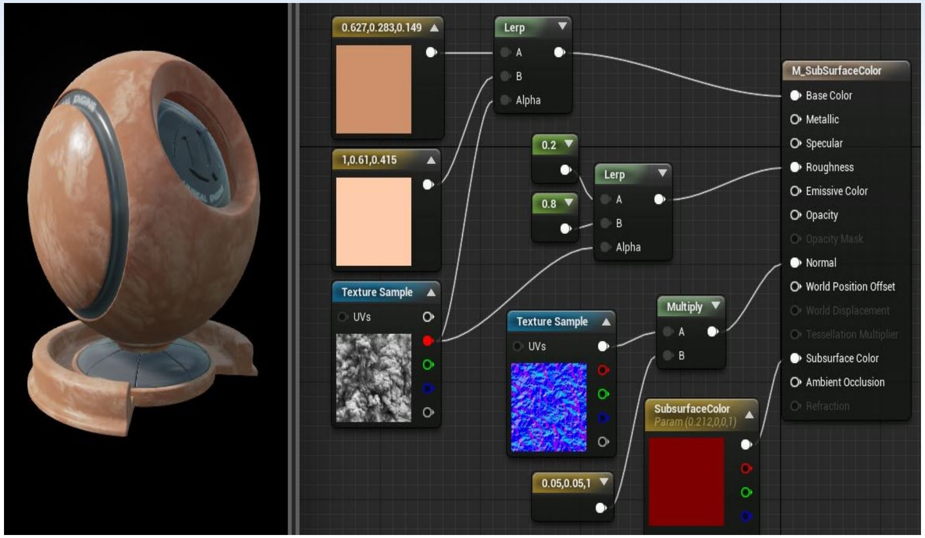
Game Engines

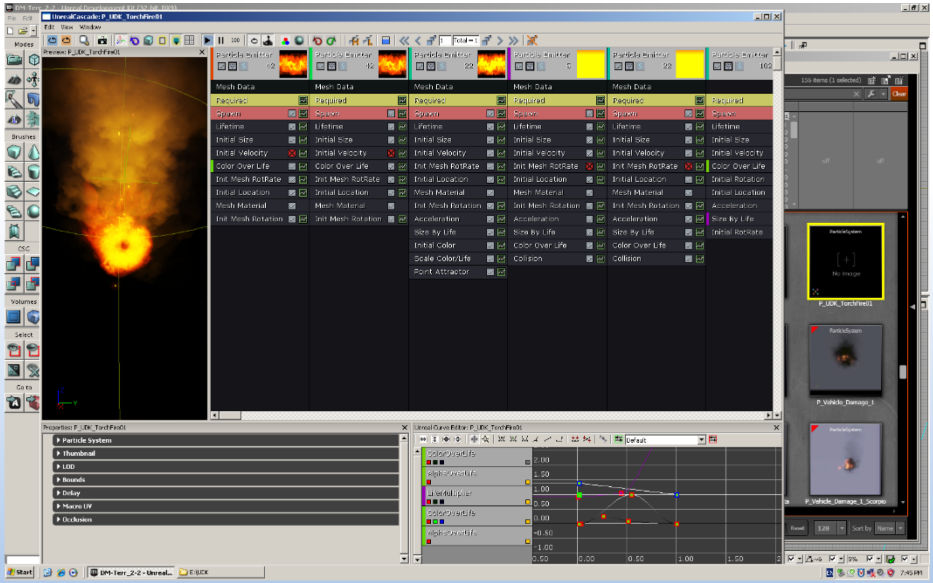
- ➢ Availability
  - Whether to select an engine still in development or one that is already established
  - Choosing a new engine is risky especially if on a tight schedule and it might not be stable, but often has more cutting-edge technology
- ➢ Extendibility
  - Ease for the developer's team to modify and add features
- ➢ License fees
  - Good engines are expensive, but may be cheaper than building the technology yourself

Game Engines

- Game engine tools


ExampleMap - Unreal Development Kit (32-bit, DX9)
File Edit View Brush Build Play Tools Preferences Help
Modes
Brushes
CSG
Volumes
Select
Go to
Current Level: Persistent Level
Start
ExampleMap - Unreal ...

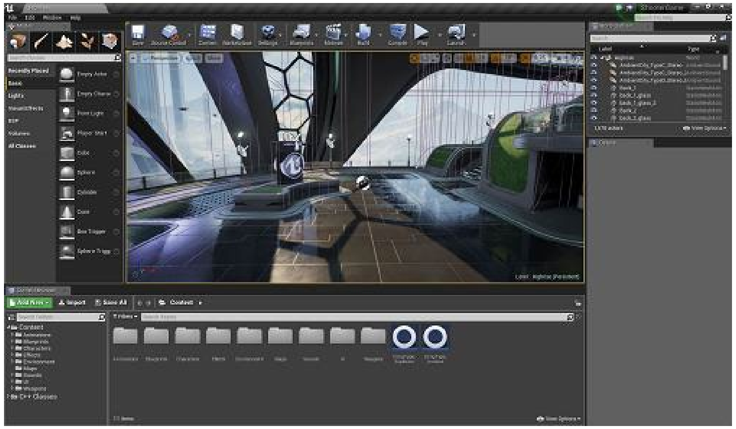




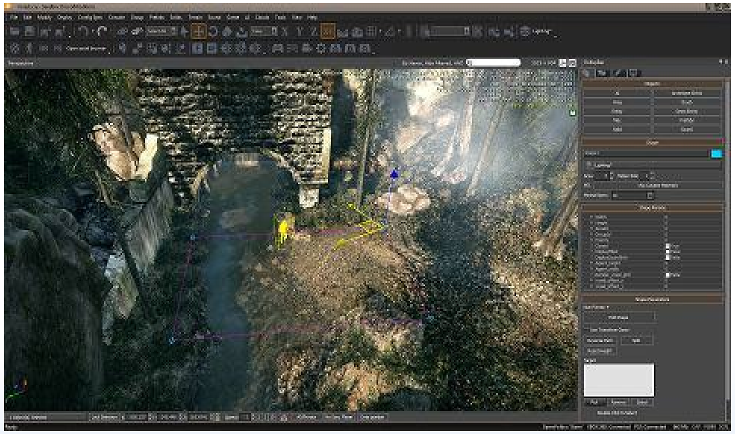
Game Engines

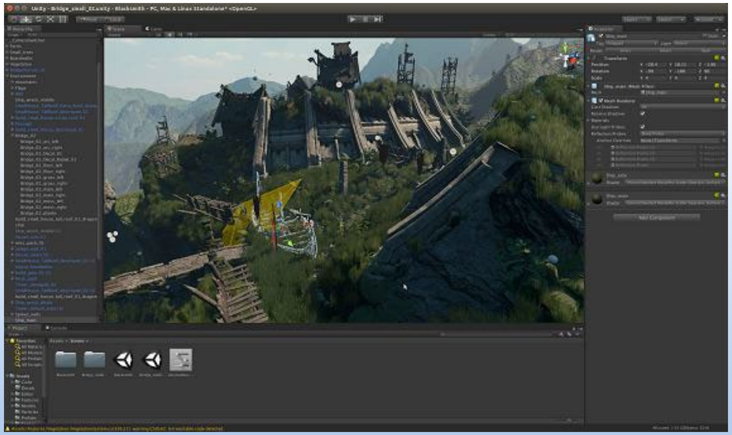
- Popular commercial game engines

Unreal Engine



CryEngine

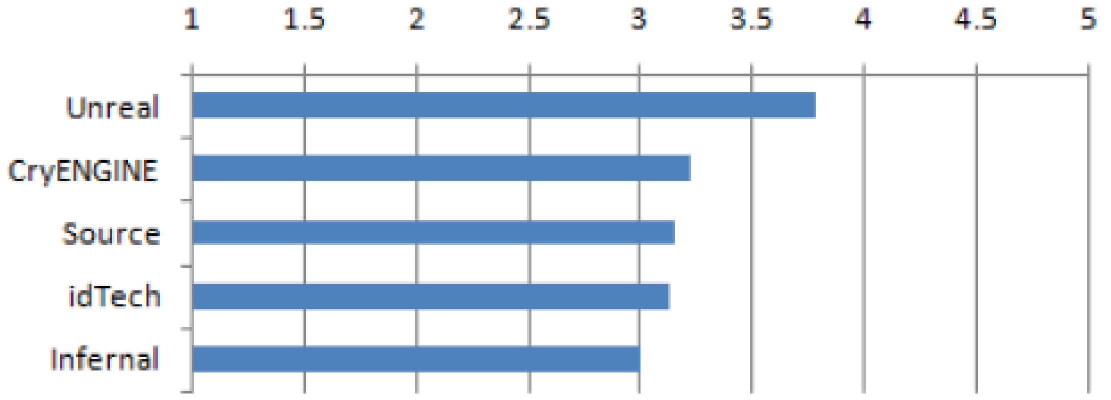




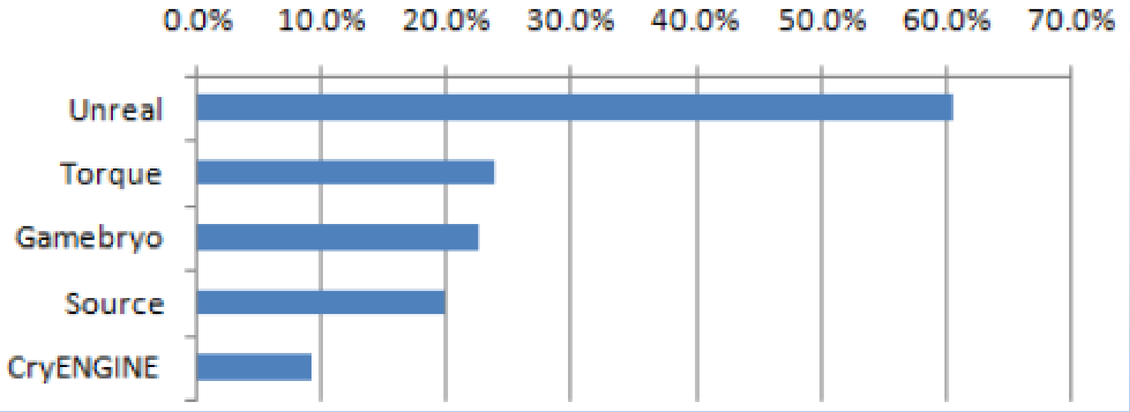
Unity

Game Engine Survey (2009)

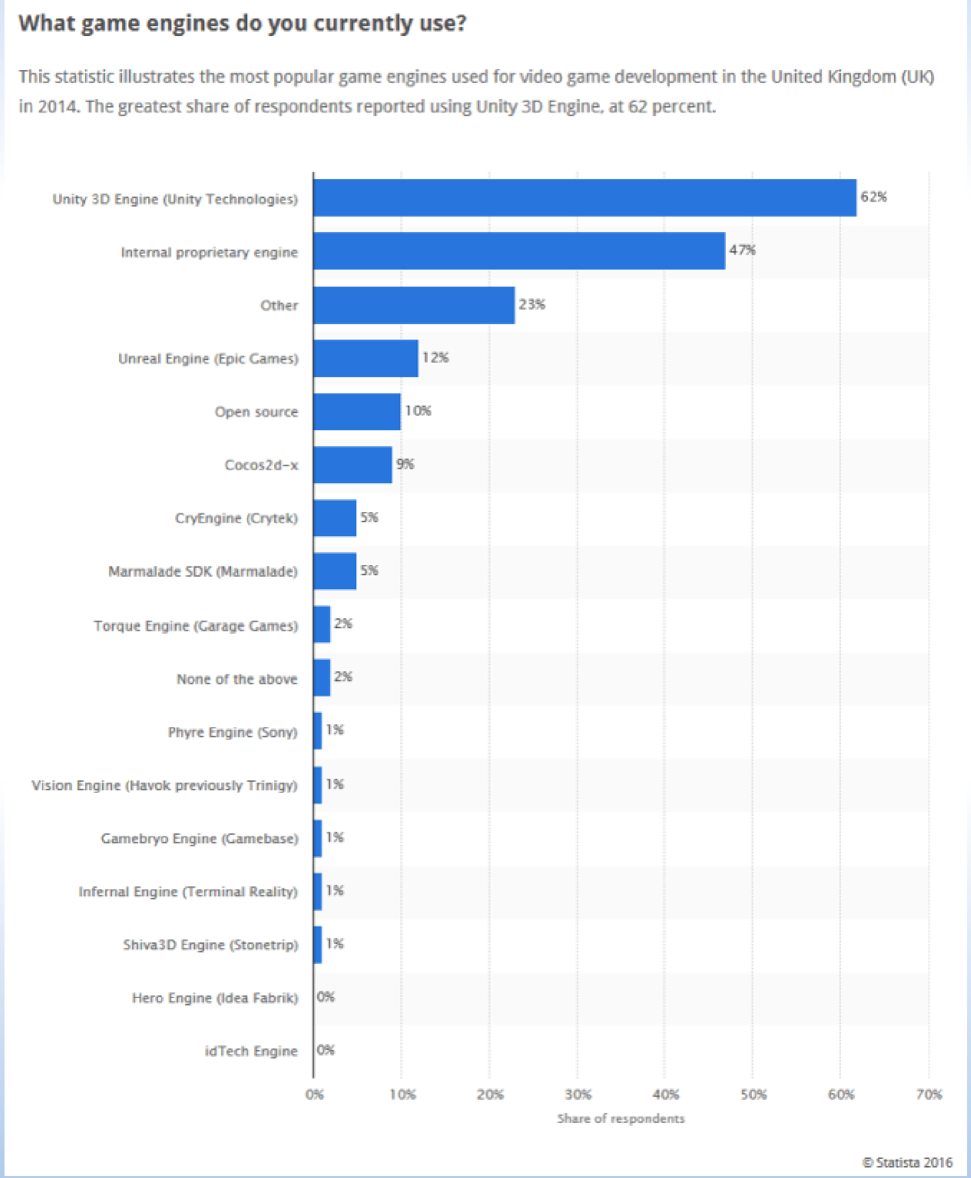
- Perceived usefulness



- Actually used



Game Engine Survey (2016)



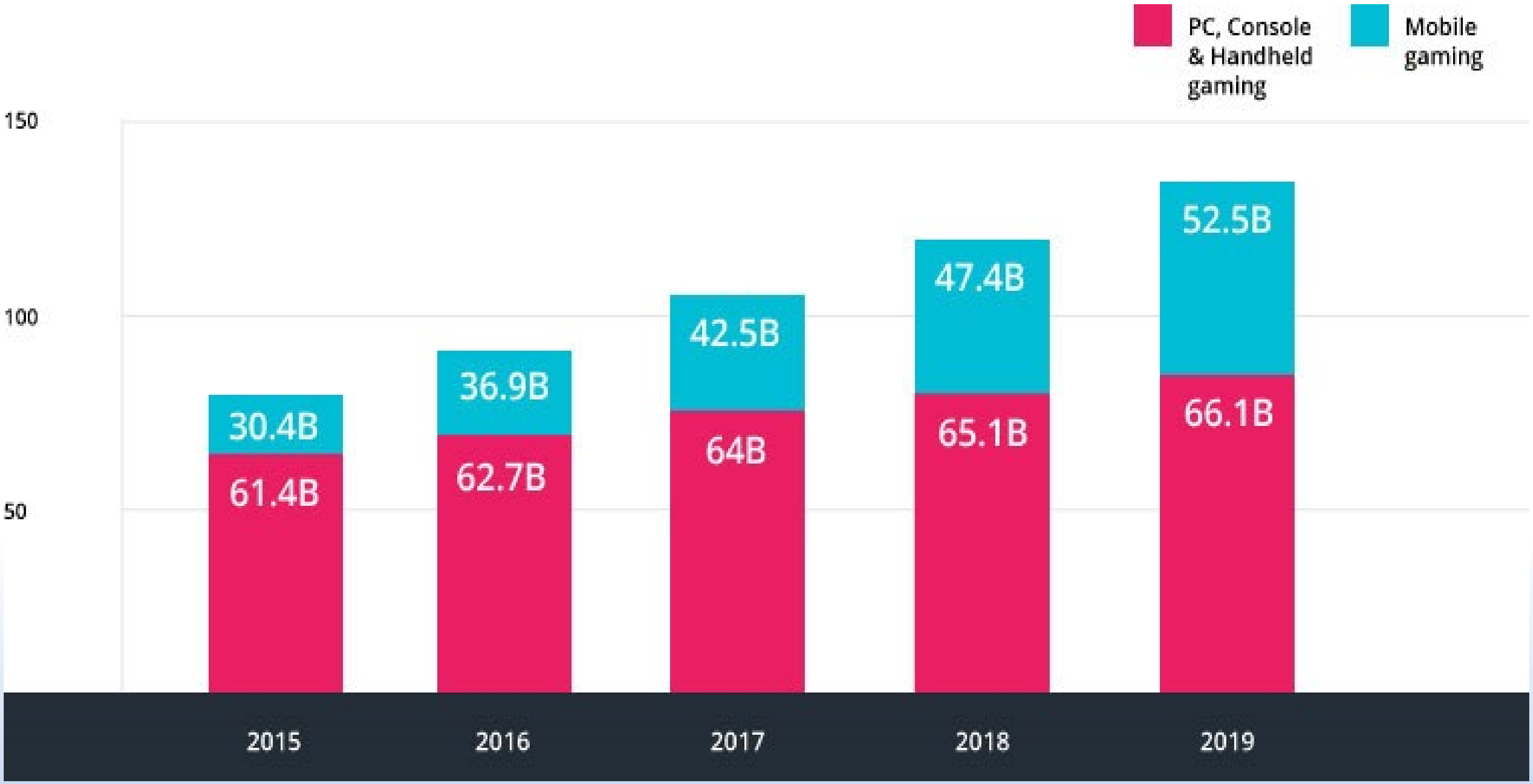
What game engines do you currently use?

This statistic illustrates the most popular game engines used for video game development in the United Kingdom (UK) in 2014. The greatest share of respondents reported using Unity 3D Engine, at 62 percent.

© Statista 2016

Mobile Gaming

- Video game market revenue



The Rendering Loop

- The rendering loop
  - For the GUI of normal applications
    - Contents on screen are mostly static
    - Only re-draw sections of screen that change via rectangle invalidation
    - Some older games use similar approaches
      - (CPUs were much slower then)
  - In real-time 3D graphics
    - Moving camera in 3D scene
    - Entire screen contents continually change
    - Use a loop to present user with a series of still images in rapid succession

http://www.stibbe.net/History/Games_Speech/Pong.htm

The Rendering Loop

- The rendering loop
  - ➢ Simple example

```
while(!iQuit)
{
    updateScene();          // update all dynamic elements
                            // in the scene
    renderScene();          // render frame into back buffer
    swapBuffers();          // swap the back buffer with
                            // the front buffer
}
```

The Game Loop

- Interacting subsystems
  - Rendering, input/output systems, animation, collision detection, physics simulations, networking, audio, etc.
  - Most require periodic servicing
    - The servicing rate for each subsystem may vary
- The game loop
  - Master loop that updates everything
  - Needs some form of time management

General Game Structure

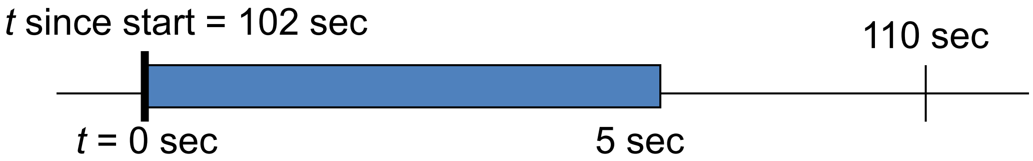
- Central logic for games include
  - ➢ Initialise graphics, input and sound controllers
  - ➢ Load resources
  - ➢ Start game loop. At every step (generates a frame):
    - Update subsystems
      - – E.g., input, camera, AI, collision detection, etc.
    - Check for terminating condition – if met, stop looping
    - Render the scene, generate sounds and game controller feedback
  - ➢ Finalise graphics, input, and sound
  - ➢ Free resources

Time

- In a simple game loop that doesn't handle time
  - ➢ The game just runs
    - On slower hardware the game runs slow, on faster hardware it will run fast
    - Slow hardware
      - – Game will run slow
      - – Worst case: game has some heavy chunks, timing becomes variable
    - Fast hardware
      - – Wasted clock cycles? Smoother experience, but
        - » Mobile devices, save battery life
        - » Running other applications at same time

Time

- Abstract timeline
  - Continuous, one-dimensional axis whose origin (*t = 0*) can lie at any arbitrary location relative to other timelines



- Real time
  - Can be obtained from CPU’s high-resolution timer register (all modern CPUs have this)
    - Different systems provide different mechanisms for accessing the high-resolution timer
    - Origin defined by when the CPU was last turned on or reset
    - Time units are in CPU cycles, can easily convert to seconds

Time

- Game time
    - Useful to have different timers
        - To pause the game, stop updating game timeline
        - An example of a simple but naïve pause that stops the game loop

```
bool pause = true;
...
while(!iQuit)     // game loop
{
  ...
  while(pause)    // if pause == true, run idle loop
  {
     sleep(1);    // sleep for 1 second
     isResume();  // check for a resume event
  }
  ...
}
```

Time

- Game time (cont.)
  - Useful to have different timers (cont.)
    - Various effects of scaling/warping one timeline relative to another
      - E.g. slow motion, fast forward, reverse time
    - Debugging
      - If camera runs on different clock, can pause game clock but still navigate around in the game world
      - Single-stepping game clock, by advancing game clock by one target frame interval
      - Note that in both cases game loop still running

Time

- Frame rate
  - ➢ How rapidly the sequence of still 3D frames is presented to the viewer, typically measured in frames-per-second
- Frame time
  - ➢ Also known as time delta/delta time (*Δt* or *dt*)
  - ➢ Amount of time that elapses between frames
  - ➢ Can measure by simply subtracting time from previous frame with time at current frame
    - Then made available to all engine subsystems
    - Note that using current frame time for upcoming frame isn’t necessarily accurate

Time

- Frame time (cont.)
  - ➢ Frame rate spikes
    - Certain frames take longer than others
  - ➢ Running average
    - Allows game to adapt to varying frame rates, softening effects of spikes
    - The longer the averaging interval
      - – The less responsive the game will be to varying frame rates, but spikes will have less of an impact
  - ➢ Governing the frame rate
    - Rather than guessing, try to guarantee every frame duration
    - Put main thread to sleep until target frame time has elapsed

Time

- Frame time (cont.)
  - ➢ Governing the frame rate (cont.)
    - Advantages
      - – Can avoid tearing artefacts due to video buffer updates not matching monitor refresh rate
      - – Recording and play back features more reliable
    - Fine if frame rate reasonably close to target frame rate, otherwise game quality can degrade significantly
    - Still a good idea to design all engine systems to allow for arbitrary durations
      - – During development leave engine in variable frame rate
      - – Switch to frame rate governing during later stages when frame rate more consistent

Time

- Multiple update frequencies
  - ➢ No law saying all parts of a game must update at the same frequency
    - Example, MotoGP combined 3 different update rates
      - – Main game logic, fixed time step 60 fps
      - – Physics update, fixed time step 120 fps
      - – Network update, fixed time step between 4 – 30 fps depending on number of players (more players less updates)

References

- Among others, material sourced from
  - ➢ Jonathan Blow, “Game development: harder than you think”, ACM Queue, Vol 1, Issue 10, Feb 2004
  - ➢ Jeff Ward, “What is a game engine?”
    - http://www.gamecareerguide.com/features/529/what_is_a_game.php
  - ➢ Steven L. Kent, “Engines and engineering: what to expect in the future of PC games”
    - http://archive.gamespy.com/futureofgaming/engines/
  - ➢ Mark DeLoura, “The engine survey”
    - http://www.gamasutra.com/blogs/MarkDeLoura/20090316/903/The_Engine_Survey_Technology_Results.php
  - ➢ Jason Gregory, Game Engine Architecture, A.K. Peters