PA199 – Game Engine Architecture

Jiří Chmelík

Semester: Autumn 2023

Ten or twenty years ago it was all fun and games.
Now it's blood, sweat, and code.
--Jonathan Blow, 2004

Outline

- Introduction – What is game Engine?

- Game Engine Architecture

- Game Engine Modules

  - Low-level Engine systems

  - Graphics and Physics

  - Gameplay systems

Outline

► Introduction – What is game Engine?

► Game Engine Architecture

► Game Engine Modules

  ▪ Low-level Engine systems

  ▪ ~~Graphics and Physics~~

  ▪ ~~Gameplay systems~~

What is Game Engine?

"The term "game engine" arose in the mid-1990s in reference to first-person shooter (FPS) games like the insanely popular *Doom* by id Software. *Doom* was architected with a reasonably well-defined separation between its core software components (such as the three-dimensional graphics rendering system, the collision detection system, or the audio system) and the art assets, game worlds, and rules of play that comprised the player's gaming experience."

[Greg2009]

What is Game Engine?

► A game engine is an open, extendable software system that can be used as the foundation for more different games, without major modification.

► A game engine is free from any function, parameter, variable, class or data structure that could be considered as part of an actual game.

► Generic infrastructure for game creation:

  ▪ Enables reuse of code

  ▪ Often facilitates porting code to various hardware platforms

  ▪ Glue together all sub-systems, middleware, libraries, etc.
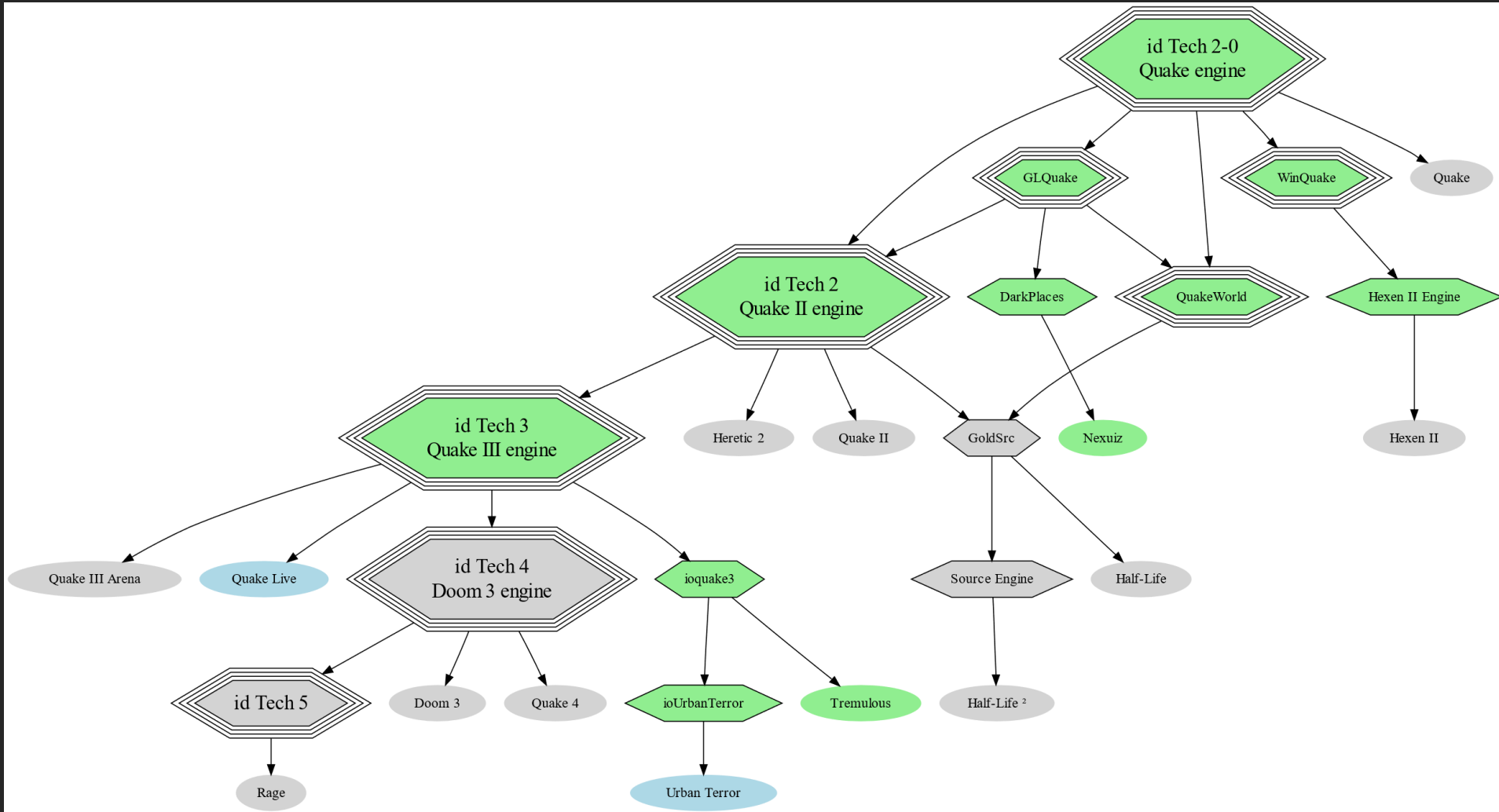
Existing Game Engines

- ► History milestones
  - ▪ Quake
    - ▪ 1996, Id Software, John Carmack et al.
    - ▪ Source codes now available
  - ▪ Unreal
    - ▪ 1998, Epic
    - ▪ Designed for FPS games, later extended to general usage.
  - ▪ Source
    - ▪ 2004, Valve
    - ▪ Half-life, Portal, …
  - ▪ Unity
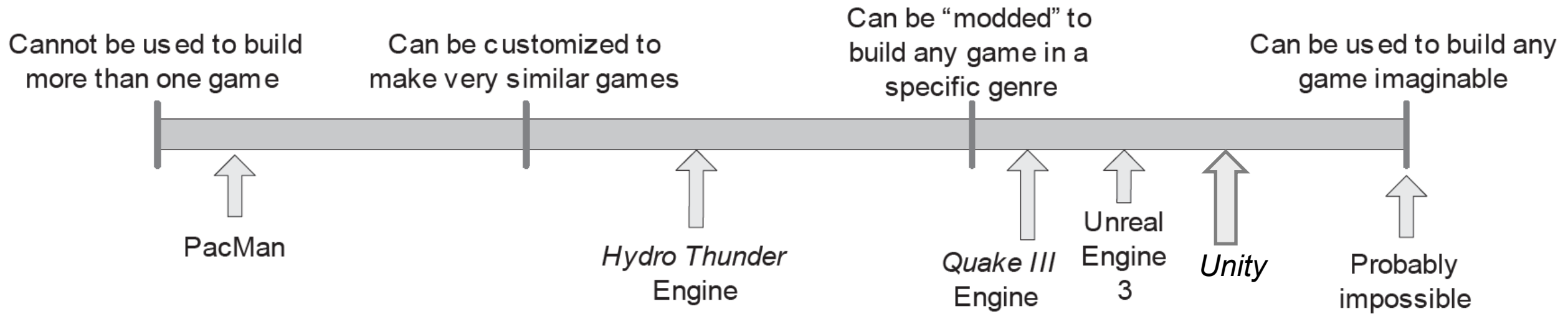   - ▪ 2005, Unity Technologies
    - ▪ many

- ► Proprietary, in-house Engines
  - ► Sage (EA)
    - ► RTS games
  - ► Glacier (IOI)
  - ► Decima (Guerrilla Games)
  - ► Northlight Engine (Remedy Ent.)
  - ► Enforce, Real Virtuality, Enfusion (BI),
  - ► LS3D (2K),

- ► https://en.wikipedia.org/wiki/List_of_game_engines

Existing Game Engines

Quake Engine family, wikipedia

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 3

Unity

Probably impossible

Game Engine Architecture

Two basic parts:

▶ **Runtime components**

▶ Tools and assets pipeline

- Digital Content Creation Tools (DCC, assets)
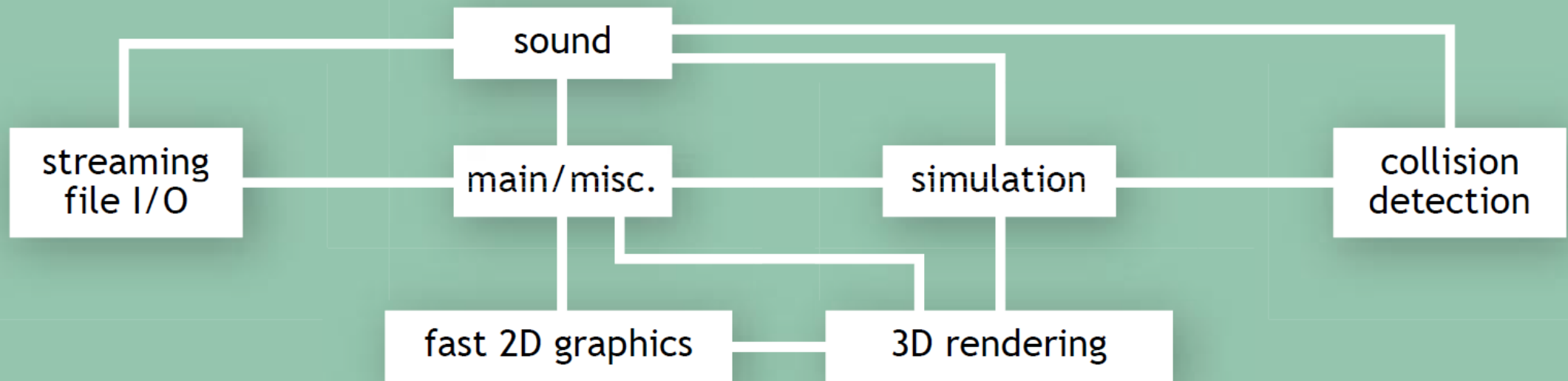
- Asset Conditioning Pipeline

- Tools – World editor
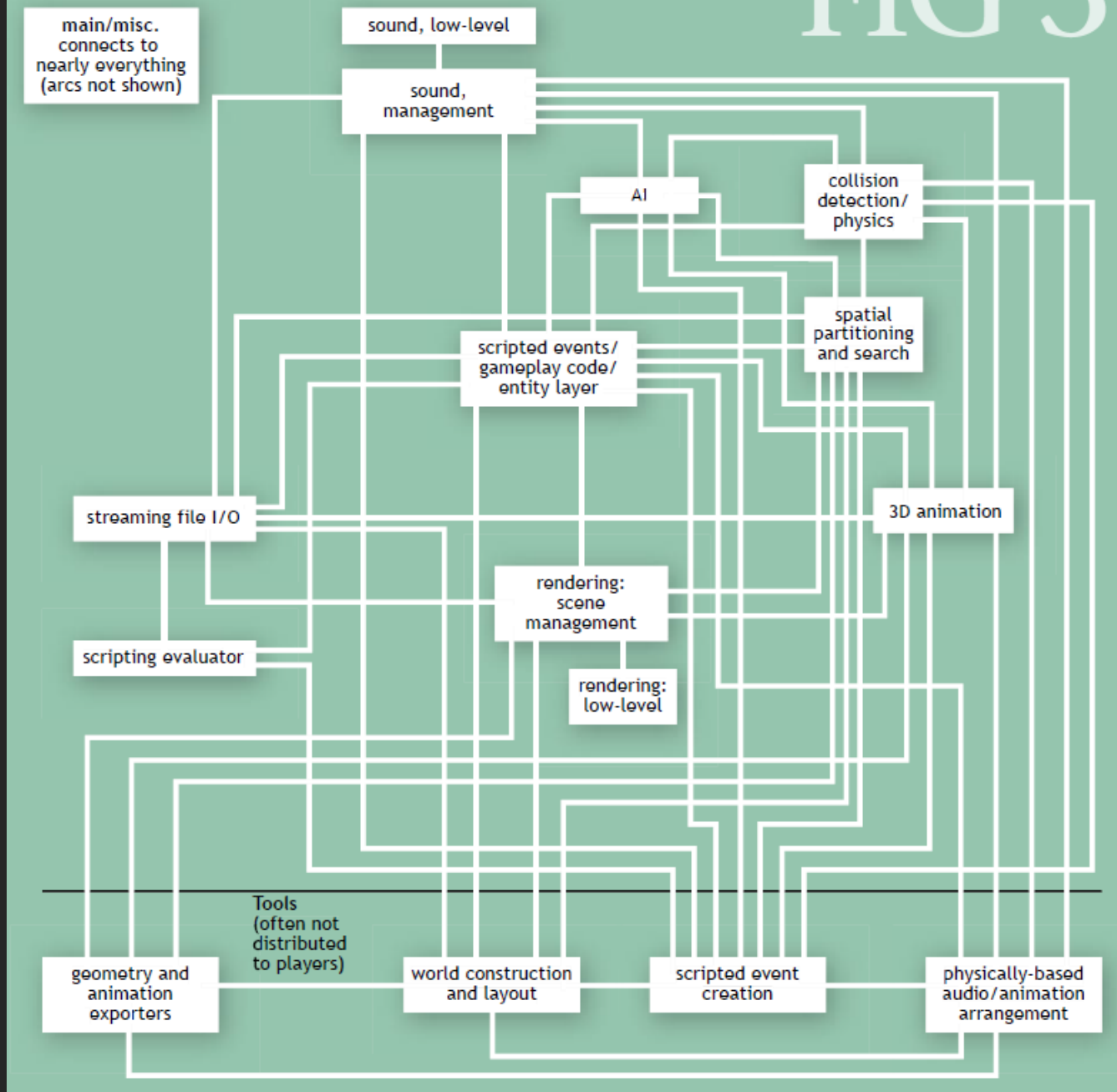
Architecture...

A 2D Game Circa 1994
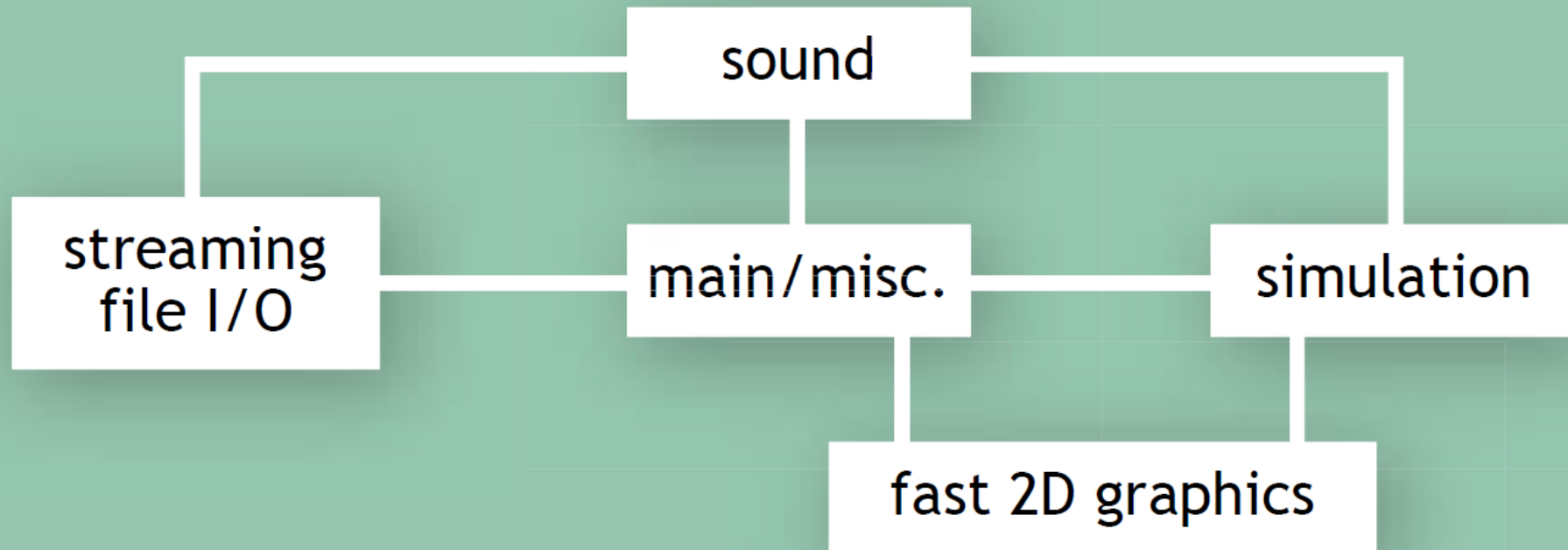
Architecture…

A 3D Game Circa 1996

[Blow2004]

Architecture...

[Blow2004]

A 3D Single-Player Game Circa 2004 — FIG 3

A...

A 3D MMG Circa 2004

FIG 4

server   shared   client

[Blow2004]
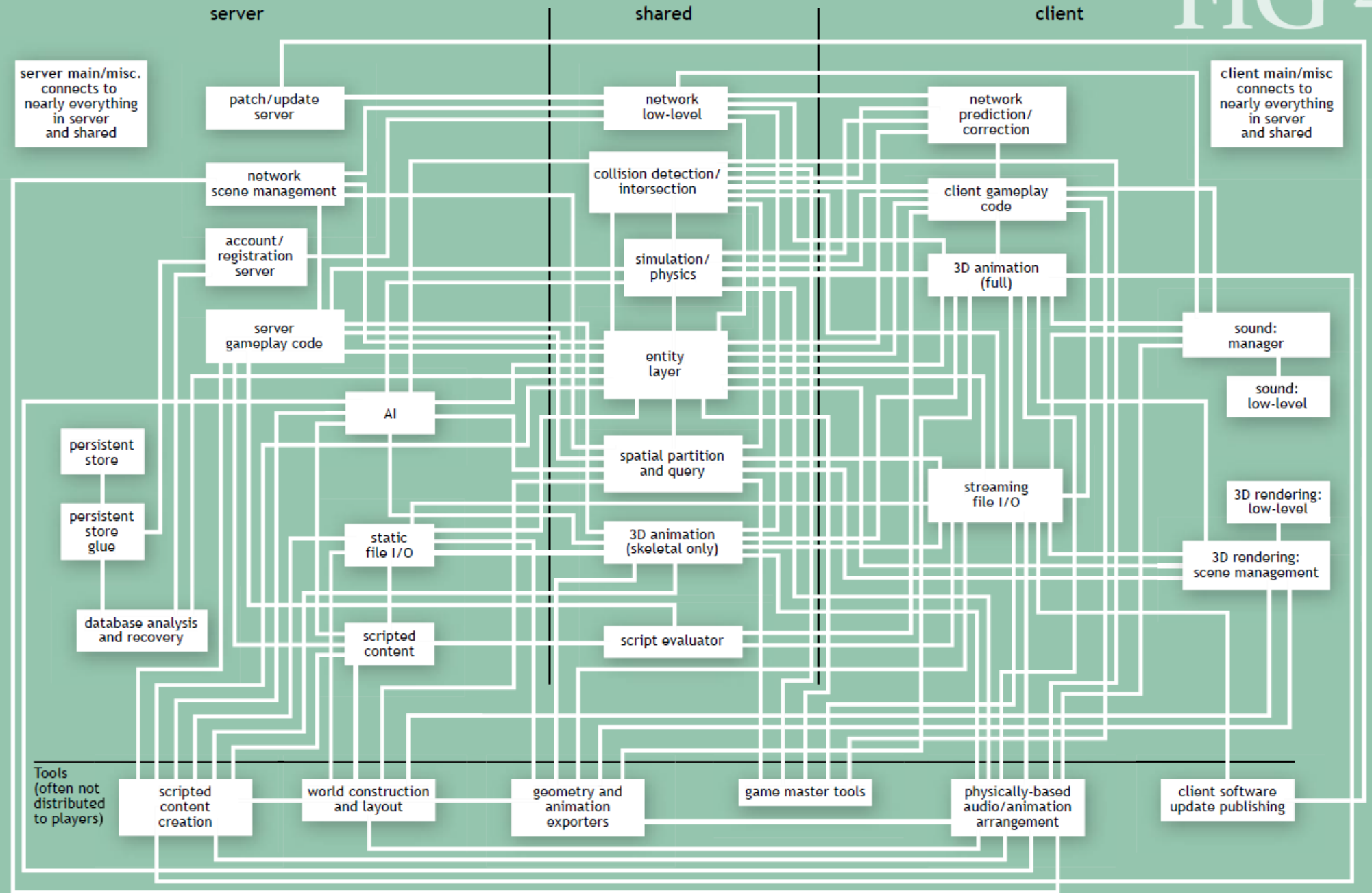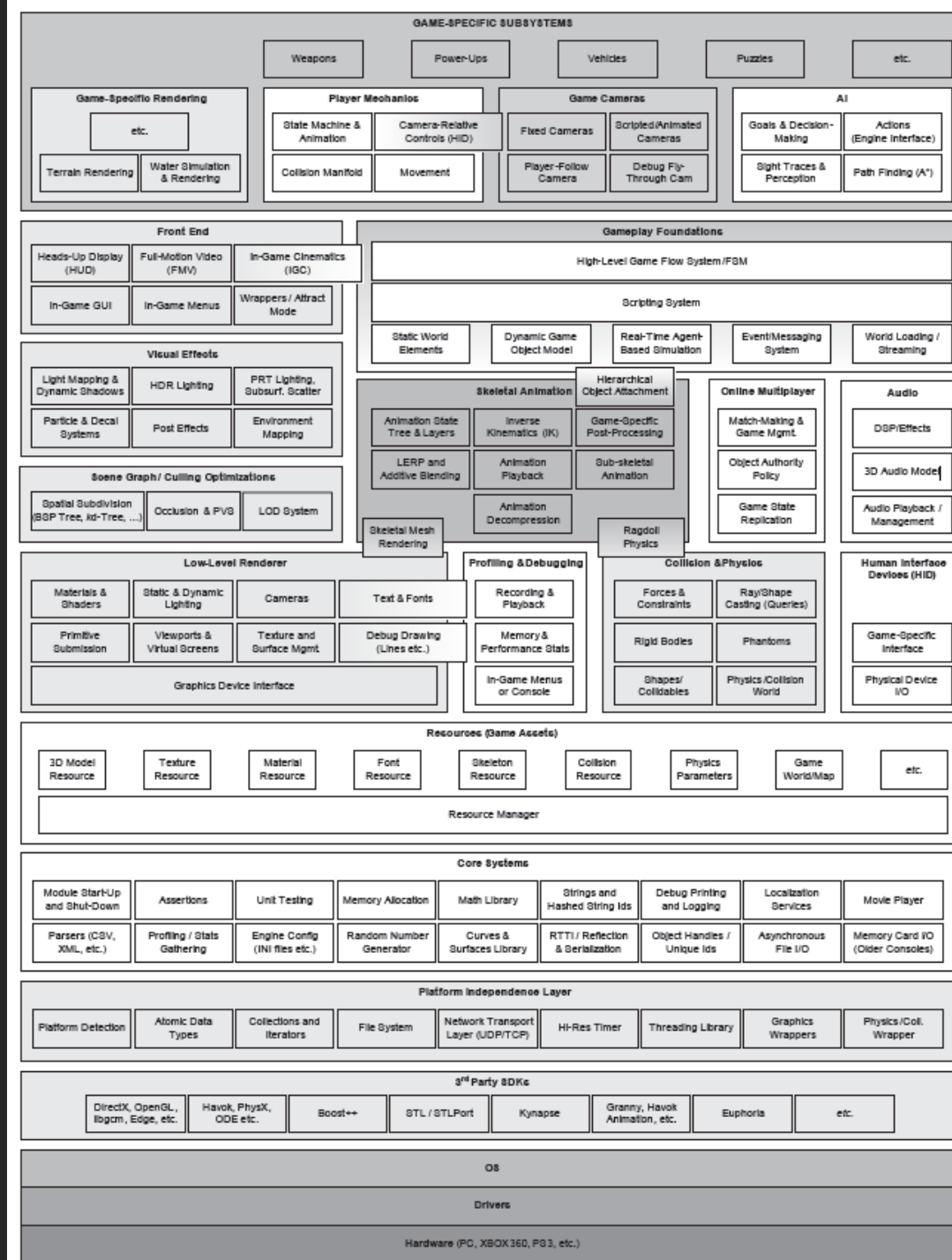
Architecture...

► Ideal scenario – reality is elsewhere

► Each node = a lot of code

[Greg2009]

Architecture - Game Engine Modules

- ► Core
- ► Graphics
- ► Animation
- ► Physics
- ► Sound
- ► Scripting
- ► Artificial Intelligence
- ► Networking
- ► User Interface
- ► Many more

- ► Low-level Engine systems
  - ▪ Core / Engine Support Systems
  - ▪ Resources and File Systems
  - ▪ Game Loops and Times
  - ▪ Human Interface Devices
  - ▪ Tools for Debugging and Development
- ► Graphics and Physics systems
- ► Gameplay systems
- ► Middleware

Outline

- Introduction – What is game Engine?

- Game Engine Architecture

- Game Engine Modules

  - Low-level Engine systems

  - Graphics and Physics

  - Gameplay systems

Outline

► Introduction – What is game Engine?

► Game Engine Architecture

► Game Engine Modules

- Low-level Engine systems

  - Core / Engine Support Systems

  - Resources and File Systems

  - Game Loops and Times

  - Human Interface Devices

  - Tools for Debugging and Development

Core / Engine Support Systems

► Starting up, shuting down subsystems

- In defined order

- Could be solved e.g. by Singleton pattern

  - Ogre (Rendering engine):

    ```
    OgreRoot.h

    class _OgreExport Root : public Singleton<Root>

    // Singletons

    LogManager* mLogManager;

    SceneManager* mCurrentSceneManager;

    MaterialManager* mMaterialManager;

    MeshManager* mMeshManager;

    SkeletonManager* mSkeletonManager;

    …
    ```

Core / Engine Support Systems

- ► Memory Management
  - ▪ RAM (along CPU, GPU times) are main resources in "runtime budget"
  - ▪ Efficient data storage
  - ▪ Standard vs. custom made data structures
    - • Continuous LOD, UE5 Nanite

- ► Localization system
  - ▪ Not just strings
- ► Engine configuration
  - ▪ Usually config files. Ogre example:
    - • `plugins.cfg` – list of optional engine plug-ins are enabled and where to find them on disk.
    - • `resources.cfg` – paths to game assets folders.
    - • `ogre.cfg` - options specifying renderer (DirectX or OpenGL), preferred video mode, screen size, etc.
  - ▪ How to load them, activate them (in-game console).

Resources and File Systems

▶ Wide variety of assets in use:

  ▪ texture (various formats),

  ▪ 3D meshes for graphics, for collisions,

  ▪ animation clips, audio clips,

  ▪ level design, etc.

▶ Each particular asset should be loaded in memory just once

  ▪ If five meshes share the same texture…

▶ Offline Asset manager (recourse manager, media manager)

Resources and File Systems

- File system
  - Wraps OS native file system API → multiplatform support
  - Filenames and paths
  - Synchronous (loading screen), asynchronous I/O operations (streaming)
    - „Genshin Impact" example
      - Cross-platform MO (not „massive")
      - Auto-updater, DLCs
      - Huge and detailed world – cannot fit into memory
        - Teleport mechanics – synchronous loading
        - Exploring mechanics – streaming – visible LOD poping effect
        - Different sizes on different platform
      - Unity

Resources and File Systems

- Asset Manager
  - Off-line (non run-time) part
    - Example – 2D artist PoW vs. level designer PoW.
    - Version control system for source assets (PSD, blender files), e.g. Perforce
    - Tools to transform assets to engine-ready form
    - Packing assets
    - Resource database tool

  - Runtime asset managment
    - Lifetime – data loading / unloading
    - Redundancy – single copy
    - Memory managment

Outline

- ▶ Introduction – What is game Engine?

- ▶ Game Engine Architecture

- ▶ Game Engine Modules

  - ▪ Low-level Engine systems

    - • Core / Engine Support Systems

    - • Resources and File Systems

    - • Game Loops and Time

    - • Human Interface Devices

    - • Tools for Debugging and Development

Game Loop

```
While (true)
{
   processInput();
   update();
   render ();
}
```

"program spends 90% of its time in 10% of the code"

Rendering Loop

```
While (!quit)
{
    updateCamera();

    updateScene();

    renderScene();

    SwapBuffers();
}
```

► Target – at least 60 FPS (about 16 milliseconds per frame).

Game Loop

► Composition of all subsystems

- Rendering loop
- Simulation loop
- I/O handling
- Audio
- Networking
- AI
- Etc.

► Various subsystems uses various frequency

- Graphics – 60Hz
- Physics simulations – 50Hz at Unity, 1000Hz for haptics
- AI – few Hz
- OS messages, callbacks – not fix frequency

► Some have to be in sync, some not

Time in Game

- ▶ Real time

- ▶ Game time, time scale, pause, …

- ▶ Animation timeline

- ▶ CPU time budget

- ▶ GPU time budget

- ▶ `Update` Δ-time, `FixedUpdate` time

- ▶ Network time (hit/miss problem)

- ▶ display's refresh rate, multithreading,

- ▶ etc.

- ▶ Time precision vs. Magnitude
  - ▪ Example: Time since game was started
    - • in seconds, stored as float value
    - • MMORPG, server running for days, weeks, …

Outline

- ► Introduction – What is game Engine?

- ► Game Engine Architecture

- ► Game Engine Modules

  - ▪ Low-level Engine systems

    - • Core / Engine Support Systems

    - • Resources and File Systems

    - • Game Loops and Time

    - • Human Interface Devices

    - • Tools for Debugging and Development

Human Interface Devices (HID)

- Pleathora of devices; input, output taxonomy …
- Key term "mapping"
  - Translates raw input (state of device, analog signals) into events (button down, button up)
  - Cross-platform support
  - Gestures (repeated tapping) recognition, combos (sequences), chords
  - Translates input events into game actions (On higher level of game engine)

- Chords
  - Multiple keys / buttons pressed in "the same" time
  - Detection (human imperfection, few frames buffer)
    - Collision with single-button actions
      - Wait before performing action
      - Start action, cancel it if chord is detected

Outline

► Introduction – What is game Engine?

► Game Engine Architecture

► Game Engine Modules

  ▪ Low-level Engine systems

    • Core / Engine Support Systems

    • Resources and File Systems

    • Game Loops and Time

    • Human Interface Devices

    • Tools for Debugging and Development

Logging and Tracing

- Old-school `print...()` functions
  - PC – console application
  - On game consoles, mobile platform – through engine – console window
- Verbosity level
  - `void VerboseDebugPrint(int verbosity, string message, ...)`
- Channnels, filters
  - Log, warning, error
  - Rendering, simulation, animation, file system, …
  - Output to file
- Crash report
  - Via exception handler
  - Current level, World-space location of the player
  - Animation/action state of the player
  - Current state of other subsystems

Debug Drawing Facilities

- Debug ray, debug 2D, 3D shapes
  - Simple to use in code, do not have to be super-fast
  - Not included in released version
- "One might say that a picture is worth 1,000 minutes of debugging." [Greg2009]

- Usually provide a simple way for taking screenshots

Debug Drawing Facilities

► *Uncharted: Drake's Fortune,* Naughty Dog [Greg2009]

Menus, console
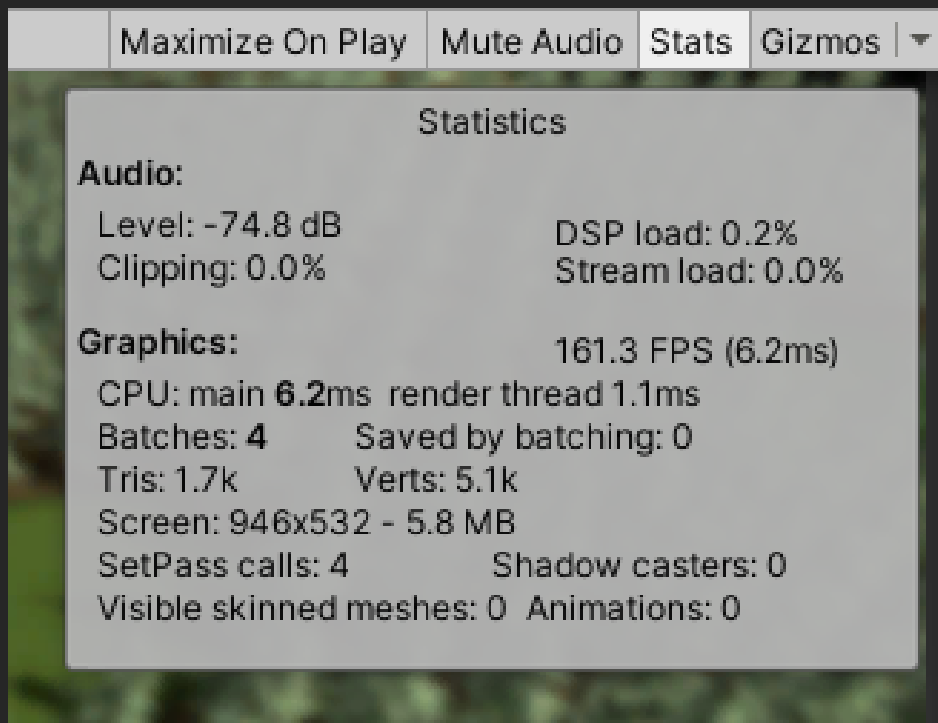
- ▶ In-game Menus
  - ▪ Turning on/off, configuring engine subsystems – in runtime
  - ▪ Should not be accessible in released games
  - ▪ Video: Uncharted 4: Debug Menu
  - ▪ Video: Unreal Engine 5 tech demo
    - • Optimized for console controls

- ▶ In-game Console
  - ▪ Video: Counter-strike with steering wheel

In-Game Profiling

▶ Simple overlay – in game, in editor

▪ Performance in editor ≠ performance in build game

Unity

Unreal Engine

In-Game Profiling

- ▶ Profiler tool
  - ▪ Timeline, recording
  - ▪ Hierarchy

- ▶ 3rd Party Tools
  - ▪ NVIDIA Nsight Graphics
  - ▪ RenderDoc
  - ▪ Xcode tools
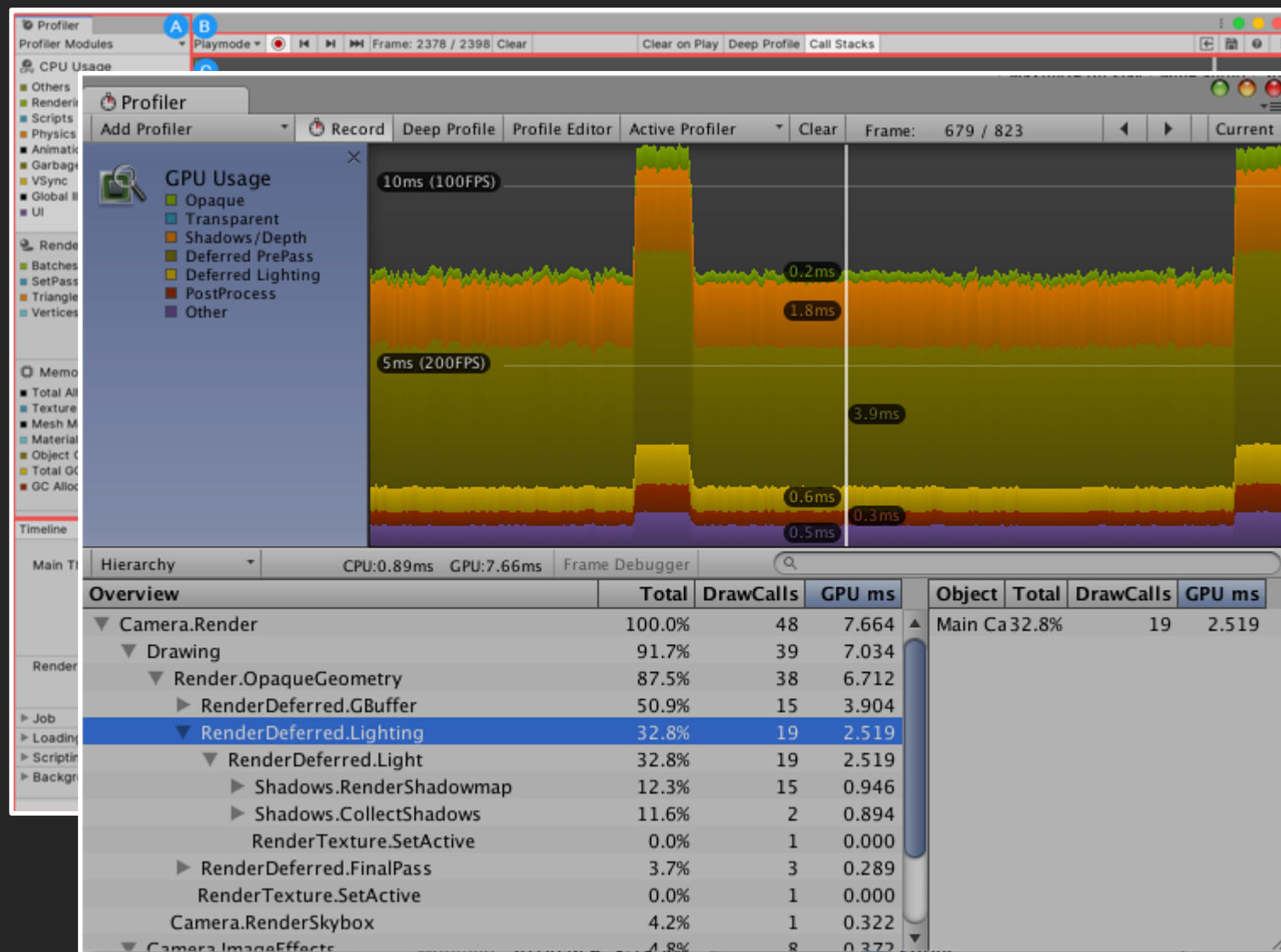
Middleware

- ► 3rd party software „layer" providing functionality of some engine sub-system
  - Graphical subsystem
    - trueSky - Cloud, Atmosphere and Weather Tool Kit
    - Ogre3D – Graphical engine itself, could be used as middleware
    - Enlighten – used in previous versions of Unity for Global Illumination
  - Physics, Animation
    - Havok – 3D physics Engine,
    - Euphoria – motion synthesis, „inteligent ragdoll"

Middleware

- ► Sound
  - ▪ Wwise - audio engine and authoring tools
    - • Cyberpunk, Hitman III, …
  - ▪ FMOD
    - • Creaks, Tomb Raider, KCD, Witcher 2, …
  - ▪ OpenAL
    - • Similar to OpenGL

References

- ► Game Programming Patterns – Robert Nystrom, 2009-2014, available online.

- ► [Greg2009] – Jason Gregory: Game Engine Architecture, 2009

- ► BinSubaih et al. - A Survey of 'Game' Portability, 2007, available online.

- ► [Blow2004] - Game Development: Harder Than You Think: Ten or twenty years ago it was all fun and games. Now it's blood, sweat, and code, 2004, available online.