CMPE 316
Game Programming

WEEK 1 – INTRODUCTION

Course Info

Goals

- To build games on the top of a game engine
- To improve your software engineering and design skills

Instructors

- Çağatay Ündeğer
- Doğaner Sümerkan

Email

- cagatay@undeger.com
- hdsumerkan@gmail.com

Course Overview

No textbook, Material provided by the instructor.

- Game Engine Architecture – Jason Gregory
- Introduction to Game Design, Prototyping, and Development - Jeremy Gibson Bond
- Learning C# from Developing Games with Unity 5.x - Greg Lukosek

Grading

- Midterm Exam %20
- Final Exam %30
- Quizzes(3-pop) %15
- Homeworks(3) %15
- Project %20
- Bonus for class participation ?

Project

Choose a project in Unity ie. Around weeek 3 or 4?

- Most important aspect of this course
- Highlights your programming/design skills
- You will select the game to develop
  - Firstly, and most importantly, develop something that you think is fun.
  - Base your project on a simple idea that can be readily implemented within the available time.
  - Include contingency planning within your project.
  - At the beginning of the third week you will submit a short outline of your planned game and its features and receive feedback on game challenge, scope, etc.
  - Advice:
    - Game Concept Development: At the end of 3rd week
    - Initial Development and Prototyping: At the end of 6th week
    - Coding: At the worst case; 7th week

Project

Each project will be marked out of 100 and judged against the following categories of assessment

- Game concept and gameplay (10)
- Quality of architectural design (10)
- Extent of game features (15)
- Complexity of game algorithms(25)
- Coding style and code quality (10)
- Github commit history (10)
- Final report (20)

Homework & Project Submission

Github?

Any bits that draw on other sources must be identified

The deadline for every homework will be 12 am

Homeworks submitted at 00:00:01 will be considered late.

Late submissions will have %5 penalty for each hour.

Students are advised to submit their work at least one hour before the deadline

Required Software

Unity 2019.xx

- Android SDK
- UWP

Visual Studio

Course Overview

Focus on programming for games

- Systems integration – graphics, sound, AI, networking, user-interfaces, physics, scripting
- Utilize higher-level toolkits, allowing for more advanced progress while still developing programming skills.

Today

Elements of Computer Games

What is a game engine?

Survey of game engine subsystems

Game Engine Architecture

Install Unity, Git,VS.. (L)

What is a Game?

Interactive experience that provides the player with an increasingly challenging sequence of patterns which he or she learns and eventually masters

This includes lots of things, but the core idea is that the “fun” is experience during the eureka moment

First-person shooters

- Games like Unreal, Half-life, Call of Duty
- Focus on
  - Efficient rendering of large 3d worlds
  - Responsive camera control
  - High-fidelity animations
  - Cool weapons
  - Forgiving physics model
- Rendering technology focuses on optimization for the environment

Third-person games

Includes games like Ratchet and Clank, Gears of War

Focus on

- Puzzle like elements
- Moving environmental objects
- Third person follow camera
- Complex camera collision system

Fighting games

Games like Tekken, Fight Night, and Soul Calibur

Technology focus on

- Fighting animations
- Hit detection
- User input system
- Crowds
- Awesome character animations and shaders
- Physics based cloth and hair

Racing games

Games like Grand Turismo, Mario Kart, and Hydro Thunder

Technology tricks include

- Using simple cards for background objects
- Track is broken down into sectors
- Third-person and first person cameras
- Camera collision

Real-time strategy

Games like Warcraft, Starcraft, Age of Empires

Technology involves

- Low resolution characters
- Height map based terrain
- Complex goal trees
- User interaction can take many forms, but reactivity is really important

MMOG

Games like World of Warcraft, Star Wars Galaxies, EverQuest

Extra technology over 3rd person include

- Server side artifacts for
  - Sign in/out
  - State management
  - Billing
- Client side rendering and state management
- Network layer for state consistency and cheat detection

Player-authored content

Allowing the player to build content as part of the game

- Different from Mods

Good example include

- Little Big Planet series
- Minecraft

Fun is in sharing with others

Simplicity is key

Simulation Games

AİM TO SİMULATE PHYSİCAL ACTİVİTİES SUCH AS FLYİNG AN AİRCRAFT, BOAT, TRAİN VS..

Serious Games

Aimed toward problem-solving rather than entertainment

Used in various areas

- education,
- healthcare,
- marketing
- defence
- simulation

AR&VR&MR Games

Technologically similar in many respects to first-person shooter engines

Many FPS-capable engines such as Unity and Unreal Engine support AR&VR&MR .

Differ from FPS games in a number of significant ways:

- Stereoscopic rendering
  - A VR game needs to render the scene twice, once for each eye
- Very high frame rate.
  - Studies have shown that VR running at below 90 frames per second is likely to induce disorientation, nausea, and other negative user effect
- Navigation issues

Team

Lots of members, many jobs

- Engineers
- Artists
- Game Designers
- Producers
- Publisher
- Other Staff

Engineers

Build software that makes the game and tools works

Lead by a senior engineer

Runtime programmers

Tools programmers

Artists

Content is king

Lead by the art director

Come in many Flavors

- Concept Artists
- 3D modelers
- Texture artists
- Lighting artists
- Animators
- Motion Capture
- Sound Design
- Voice Actors

Game Designers

Responsible for game play

- Story line
- Puzzles
- Levels
- Weapons

Employ writers and sometimes ex-engineers

Elements of Computer Games

3-dimensional computer graphics:

- Most high-end computer games involve the generation of photo-realistic imagery at the rate of 30–60 frames per second. This process is complicated by the following issues
  - Large geometric models: Large-scale models, such as factories, city-scapes, forests and jungles, and crowds of people, can involve vast numbers of geometric elements.
  - Complex geometry: Many natural objects (such as hair, fur, trees, plants, water, and clouds) have very sophisticate geometric structure, and they move and interact in complex manners.
  - Complex lighting: Many natural objects (such as human hair and skin, plants, and water) reflect light in complex and subtle ways.

Elements of Computer Games

Motion and Navigation:

- Nonplayer entities need to be able to plan their movement from one location to another.
  - This can involve finding a shortest route from one location to another, moving in coordination with a number of other nonplayer entities, or generating natural-looking motion for a soccer player in a sports game.

Elements of Computer Games

Artificial intelligence:

- The game software controls the motions and behaviors of nonplayer entitites. Achieving realistic behavior involves an understanding of artificial intelligence.

Elements of Computer Games

Physics:

- The phyical objects of a game interact with one another in accordance with the laws of physics.
- Implicit in this is the capability to efficiently determine when objects collide with one another, and how they should move in response to these collisions

Elements of Computer Games

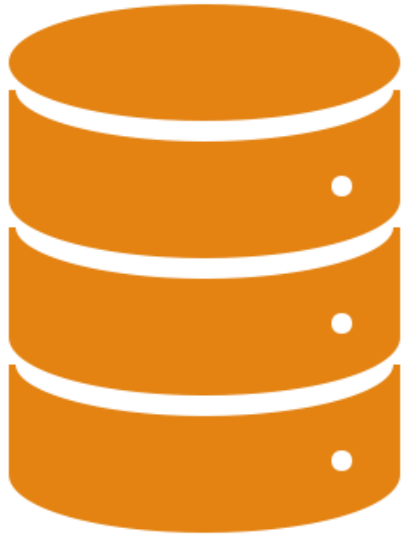
Networking:

- Multiplayer online games use a network to communicate the current game state between players. Due to the latency inherent in network communication, the games states that individual players perceive are momentarily inconsistent. The game software must hide this latency and conceal these inconsistencies from the players.

Elements of Computer Games

Databases:

- The game state, especially for multiplayer online games, is maintained in a database. The database receives requests for updates of the game state from multiple sources, and it must maintain a consistent state throughout.



Elements of Computer Games

Security:

- Since the origin of games, there have been people who have sought ways of circumventing the games. This is particularly an issue in multiplayer games, where one player's cheating behavior degrades the experience for an honest player. For this reason, game software needs to be vigilant to detect and obstruct illicit game interaction.

Game Engines

In the old days, programmers would design games from scratch.

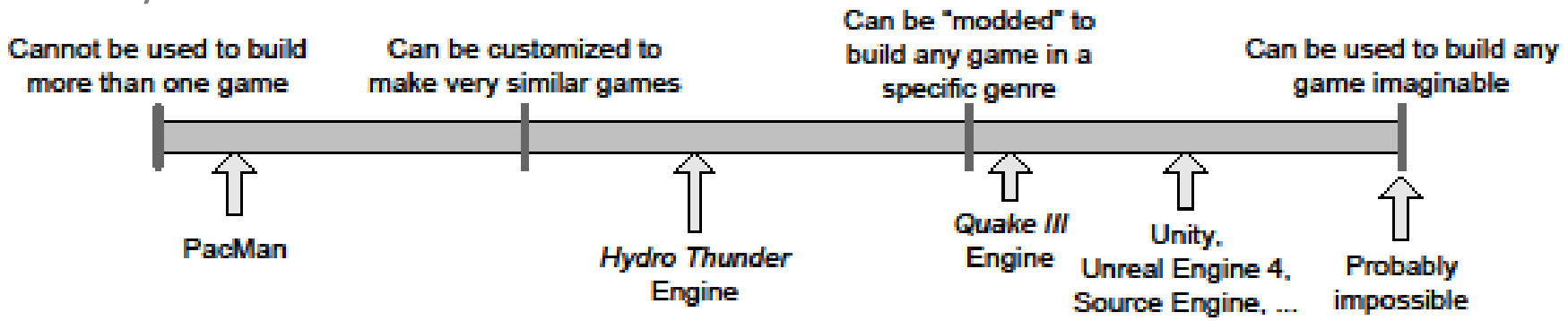
Given the complexities of modern games, game programmers rely on software systems called game engines to provide underlying support (rendering, animation, physics).

This frees the game programmer from many of the more mundane tasks in order to focus on the essential elements of the game play.

Some common examples of game engines include Unity 3D, Unreal Engine and CryEngine

What is a game engine

- The term game engine arose in the 1990s
- Doom by id was at the center
  - The core components were separated from the game content
- Quake III and Unreal were designed with the separation in mind
  - Sold licenses to their engine and tools
  - So of you may have done modding using these tools
- Game engine are data-driven architectures that are reusable and therefore do not contain game content – mostly true



What is a game engine

A game engine is a software system designed for the creation and development of video games. (Wikipedia).

The core functionality typically provided by a game engine may include

- a rendering engine ("renderer") for 2D or 3D graphics,
- a physics engine or collision detection (and collision response),
- sound,
- scripting,
- animation,
- artificial intelligence,
- networking,
- streaming,
- memory management,
- threading,
- localization support,
- scene graph,
- video support for cinematics.

Some current engines

Quake family

- Used to create many games
- Has lineage that extends to modern games like *Medal of Honor*
- *Quake* and *Quake II* engines source code are freely available

Unreal Engine

- Very rich tool set
- Large developers network
- Good licensing model – good for small developers

Some current engines

Unity

- Very feature rich
- Uses Javascript or C# for scripting
- Large community support
- Great for cross-platform development

Source Engine

- Games like *Half-life 2* and its sequels, *Team Fortress 2,* and *Portal*
- Very powerful with good graphics capabilities and a good toolset

DICE's Frostbite

- Used to create games like *Battlefield 4*
- FrostEd – asset creation tool

More engines

CryEngine

- Originally developed as a demo for Nvidia
- Used to develop numerous games – starting with *Far Cry*

Sony PhyreEngine

- Uses to create games for the Sony platforms
- Numerous titles have been written with this engine

Microsoft XNA and MonoGame

- Based on C# - easy to use
- Used for Xbox and PC games
- Not longer supported – replaced by MonoGame

2D Engines

Designed for non-programmers to build apps for Android and iPhone

Examples include

- Multimedia Fusion 2
- Game Salad Creator
- Scratch

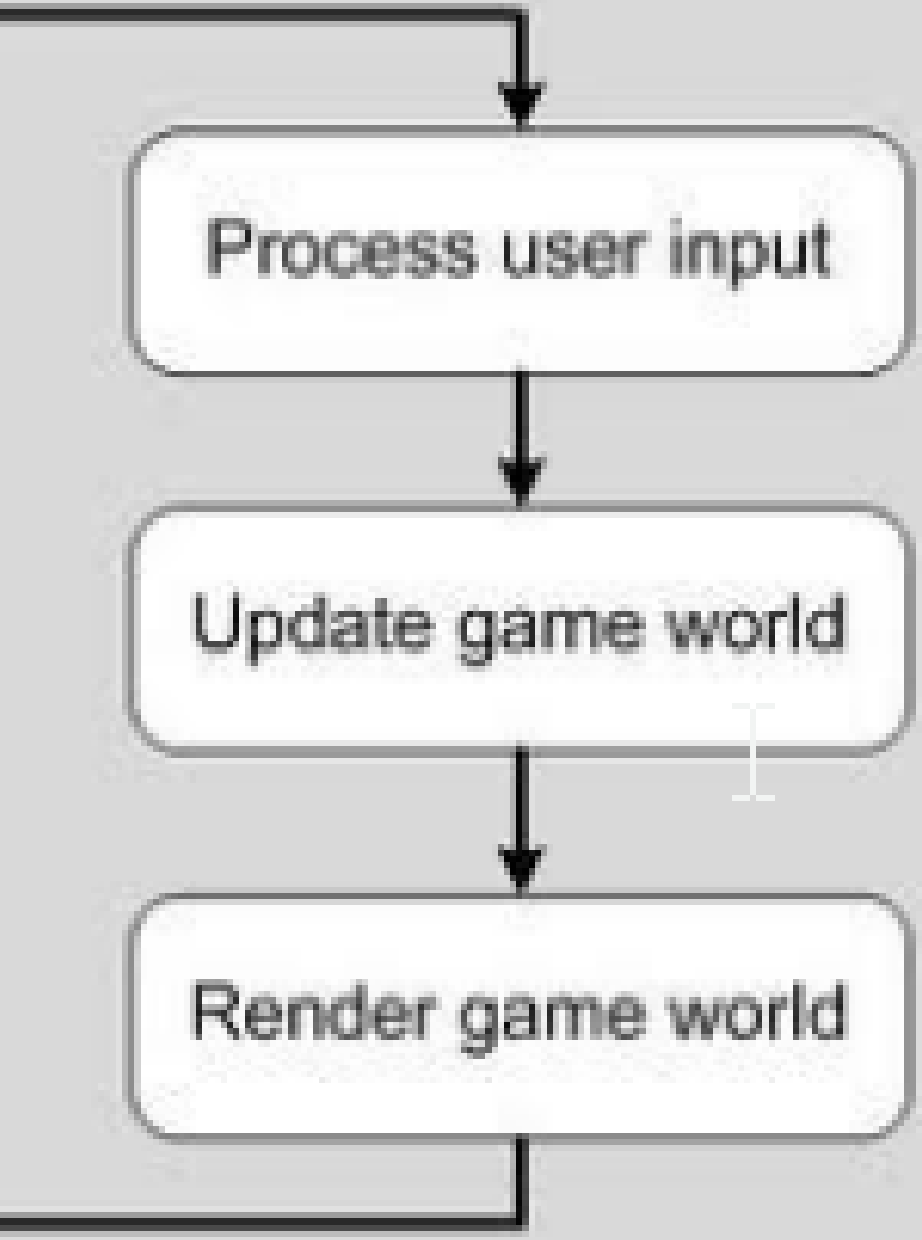
Games for us

At the core of every real-time system is the concept of a deadline.

- In video games is the requirement that the screen be updated at least 24 times per second in order to provide the illusion of motion

Most of these can be described as *soft real-time computer simulations*

- They are simulations of the real world using a mathematical model
- They are interactive so have at least one actor or agent
- They have loose real-time constraints



Game Engines

Since the flow of the program (game) depends on the actions of the player, most of the game engines are event driven:

- Mouse movements.
- Key presses.
- Avatar collision.
- Avatar near a bot.
- Network activity.
- The contents of a environment definition file.

Programming a game engine is therefore a matter of programming what happens when an event occurs.

	Unity	Unreal	Torque	HTML5	Flash	Source Engine	Ogre	Second Life	GameMaker	XNA
Level editor	■	■	■		■	■		■	■	
Scripting	■	■	■	■	■			■	■	
C++		■	■			■	■			
Networking	■	■	■	■	■	■		■	■	■
3D Graphics	■	■	■			■	■	■		■
Shader effects	■	■	■			■	■		■	■
Dynamic shadows	■	■	■			■	■			■
Physics	■	■	■			■		■	■	■
Artificial Intelligence	■	■	■			■			■	
Free non-commercial	■	■	■	■		■	■	■		■
Free for commercial				■			■			■
Mobile Devices	■	■		■	■		■		■	■
Web player	■			■	■				■	■

Game Engines

Cowan, Brent and Bill Kapralos. “A Survey of Frameworks and Game Engines for Serious Game Development.” *2014 IEEE 14th International Conference on Advanced Learning Technologies* (2014): 662-664.

Game Engine Architecture

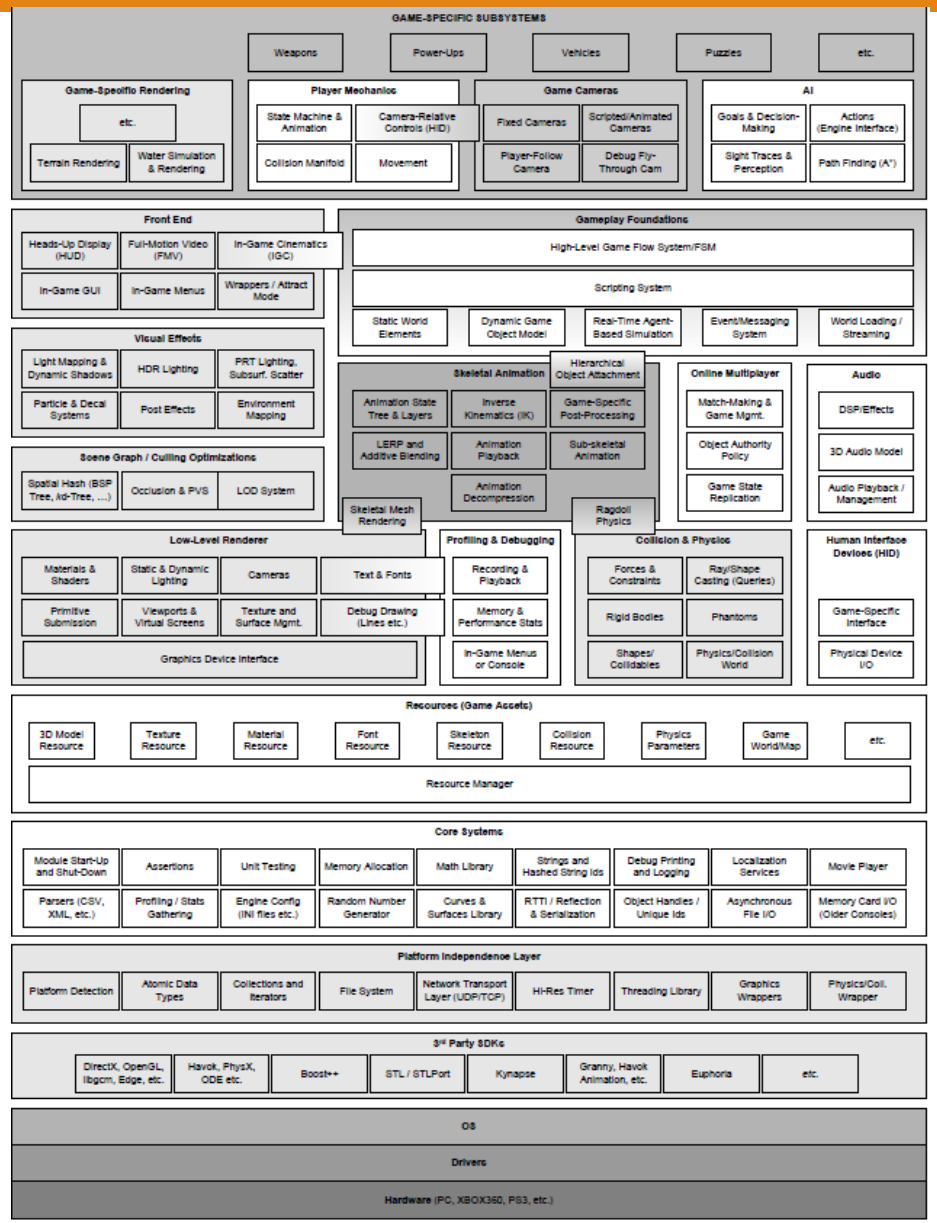
Consists of the tools suite and runtime components

Large

- Spans hardware to high-level application

Designed in layers

- Avoids circular dependencies to maximize reuse and testability


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
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
Scene Graph / Culling Optimizations
Spatial Hash (BSP Tree, kd-Tree, ...)
Occlusion & PVS
LOD System
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 Engine Architecture

Low level components

3rd Party SDKs

Platform independence layer

Core systems

Resources manager

Rendering engine

Profiling/Debugging

Collisions and Physics

Animation

Human Interface Devices

Audio

Gameplay foundation system

Low level components

Hardware

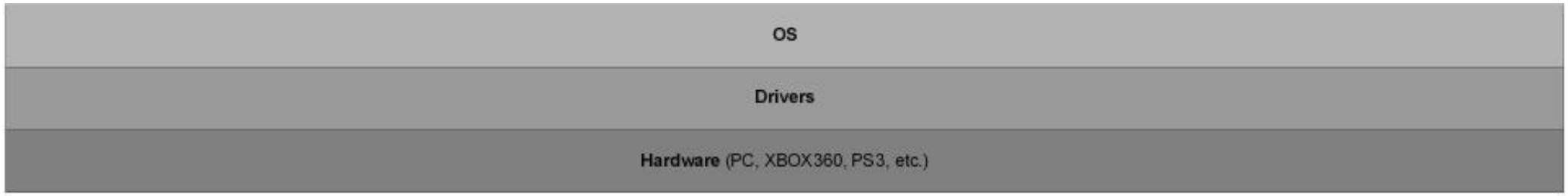
- This is the system that the game is to run on

Device Drivers

- Shield the OS and upper layers from low level device communications details

Operating System

- Handles the execution and interruption of multiple programs on a single machine
- Very thin on a console



3rd Party SDKs

These are libraries and sofware development toolkits (SDKs), usually provided from a third party.

Data Structure and Algorithms

- STL – C++ standard template library data structures, strings, stream-based I/O
- Boost – powerful data structures and algorithms

Graphics

- OpenGL and DirectX

Collisions and Physics

- Havok, PhysX, ODE, Bullet

Character Animation

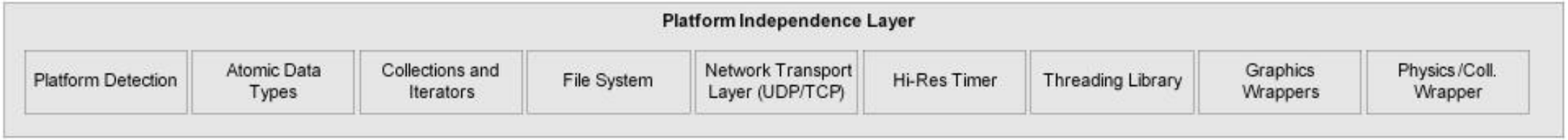
Artificial Intelligence – Kynapse

Platform independence layer

Allows the engine to be developed without the concern of the underlying platform

Provides wrappers to common target specific operations

Include things like primitive types, network, file systems, etc.



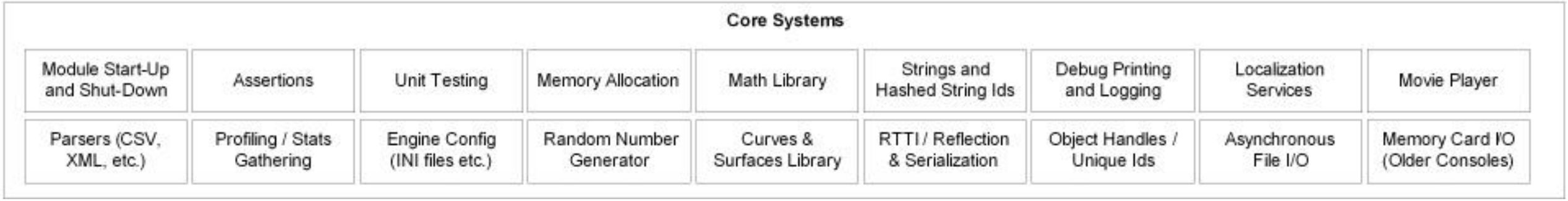
Core systems

Assertions – error checking code

Memory Management

Math library – vector and matrix math, numeric integrators

Custom data structures

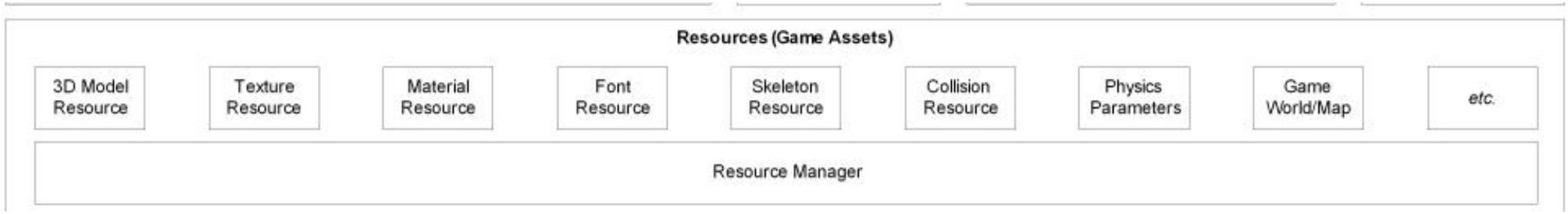


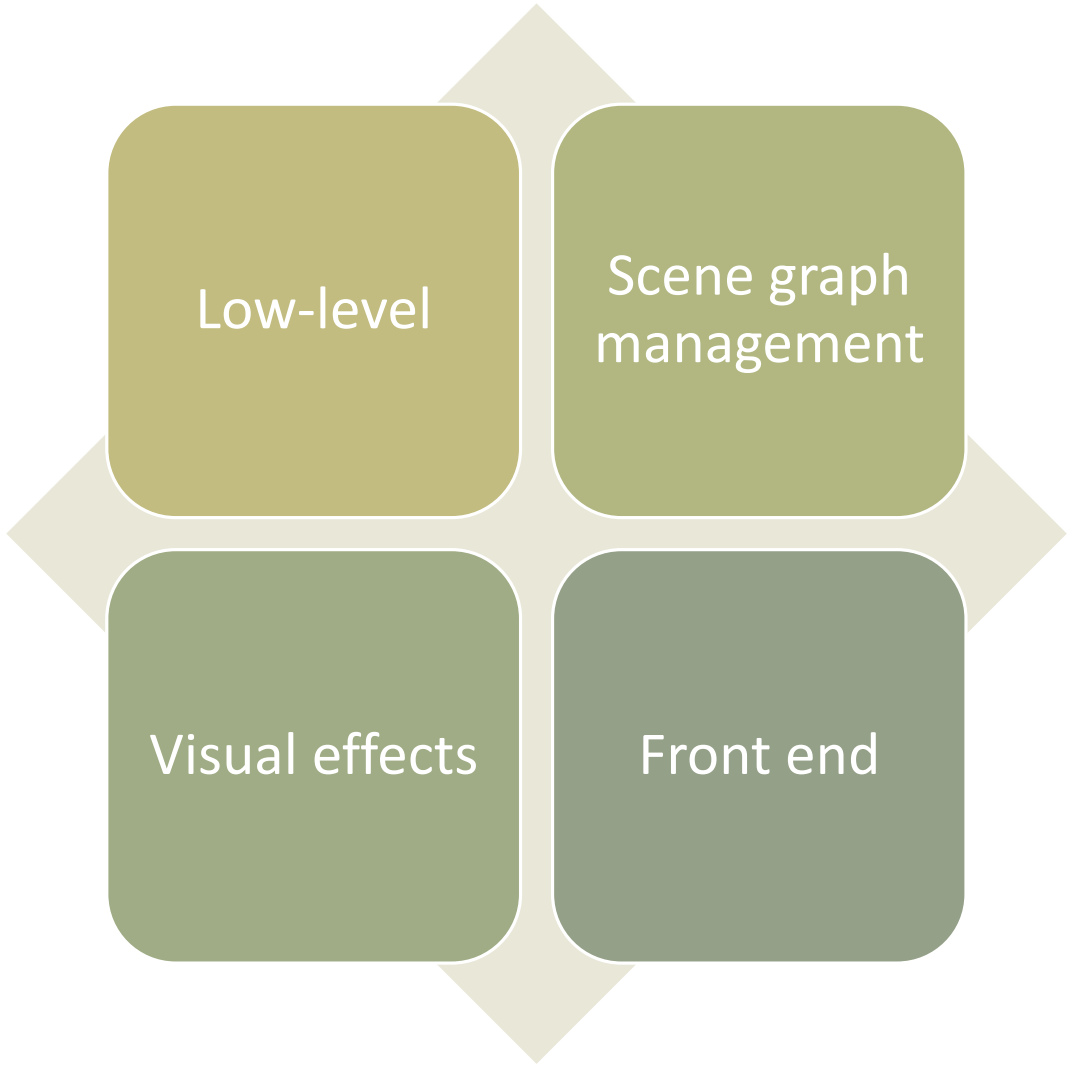
Resource manager

Provides a unified interface for accessing assets

The level of complexity is dictated by need

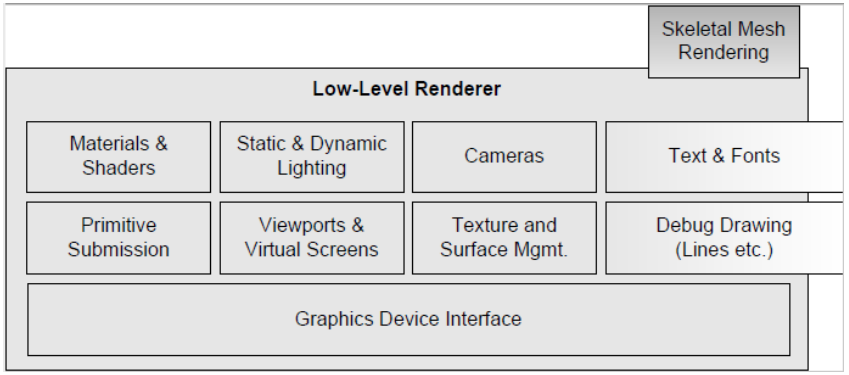
- Often the game programmers must do resource loading directly
- Engines like UT do unpackaging and complex manipulation of assets in the engine





Rendering engine

Low-level Renderer



Focuses on rendering primitives as quickly and richly as possible

- Does not consider visibility

Graphics Device Interface

- Access and enumerate the graphics devices
- Initialize the GD
- Setup buffering

Others

- Representation of the geometric primitives
- Abstraction of the camera interface
- Material system
- Dynamic lighting system
- Text and fonts

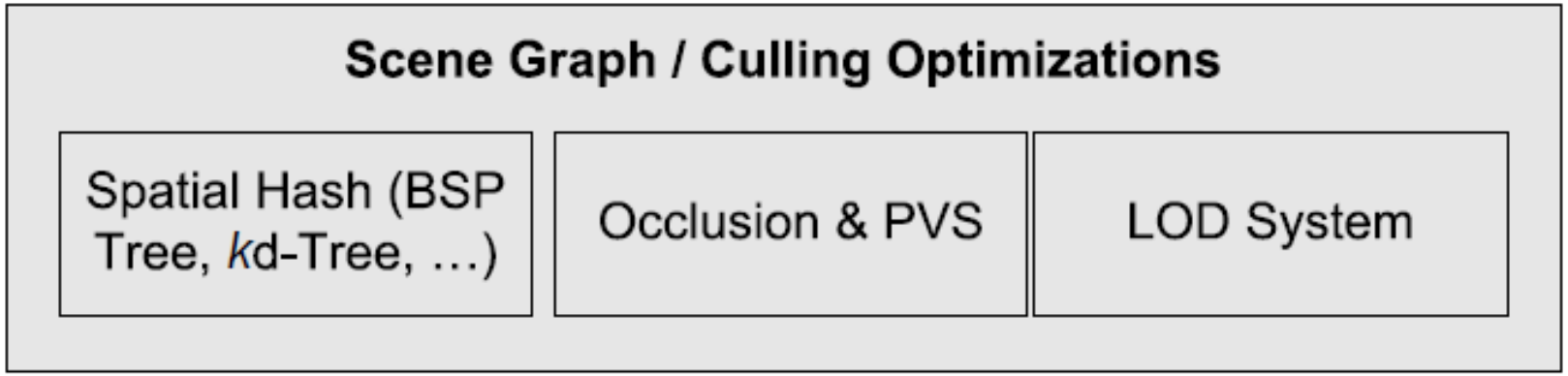
Scene graph

Limits the number of primitives submitted for rendering

Uses frustum culling – remove things outside of the visible screen

Spatial subdivision

- BSP, quadtree, octree, kd-tree



Visual effects

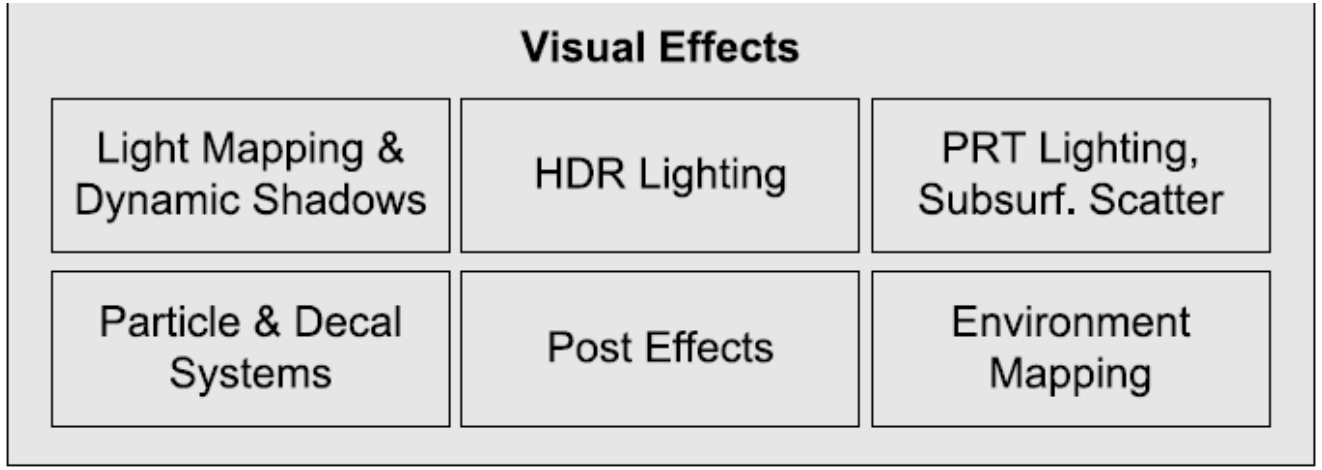
Particle systems

Decal systems

Light mapping

Dynamic shadows

Full screen post effects



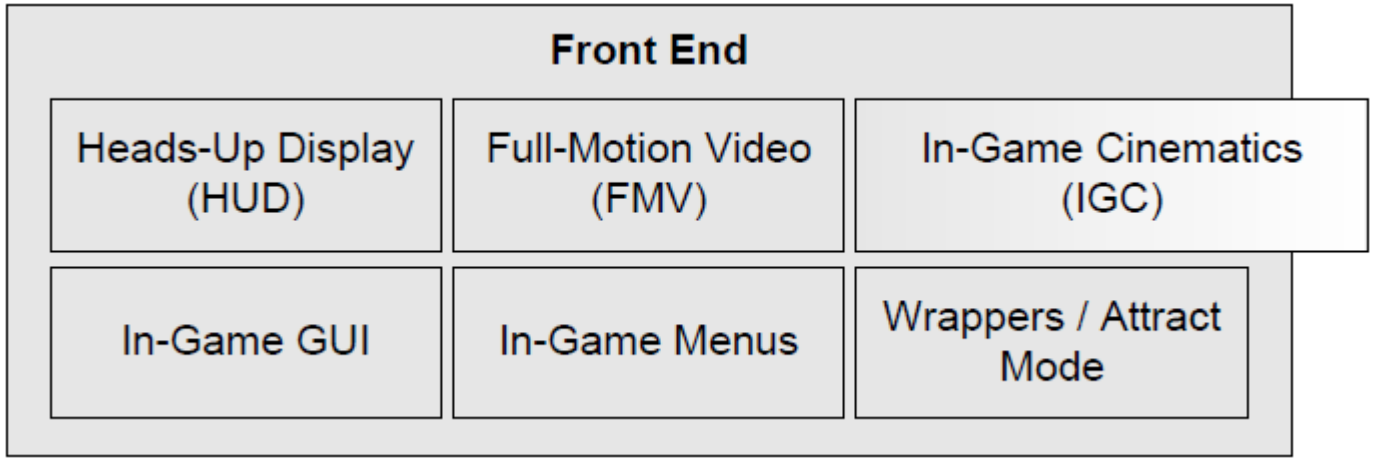
Front end

HUD

Menus

GUI for character manipulation

Full-motion video for cut scenes



Profiling/Debugging

Code timing

Display stats on the screen

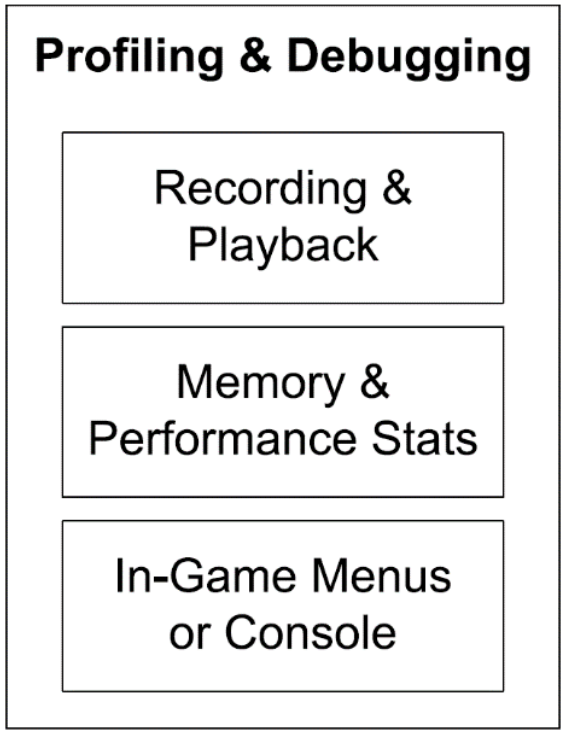
Dumping performance stats

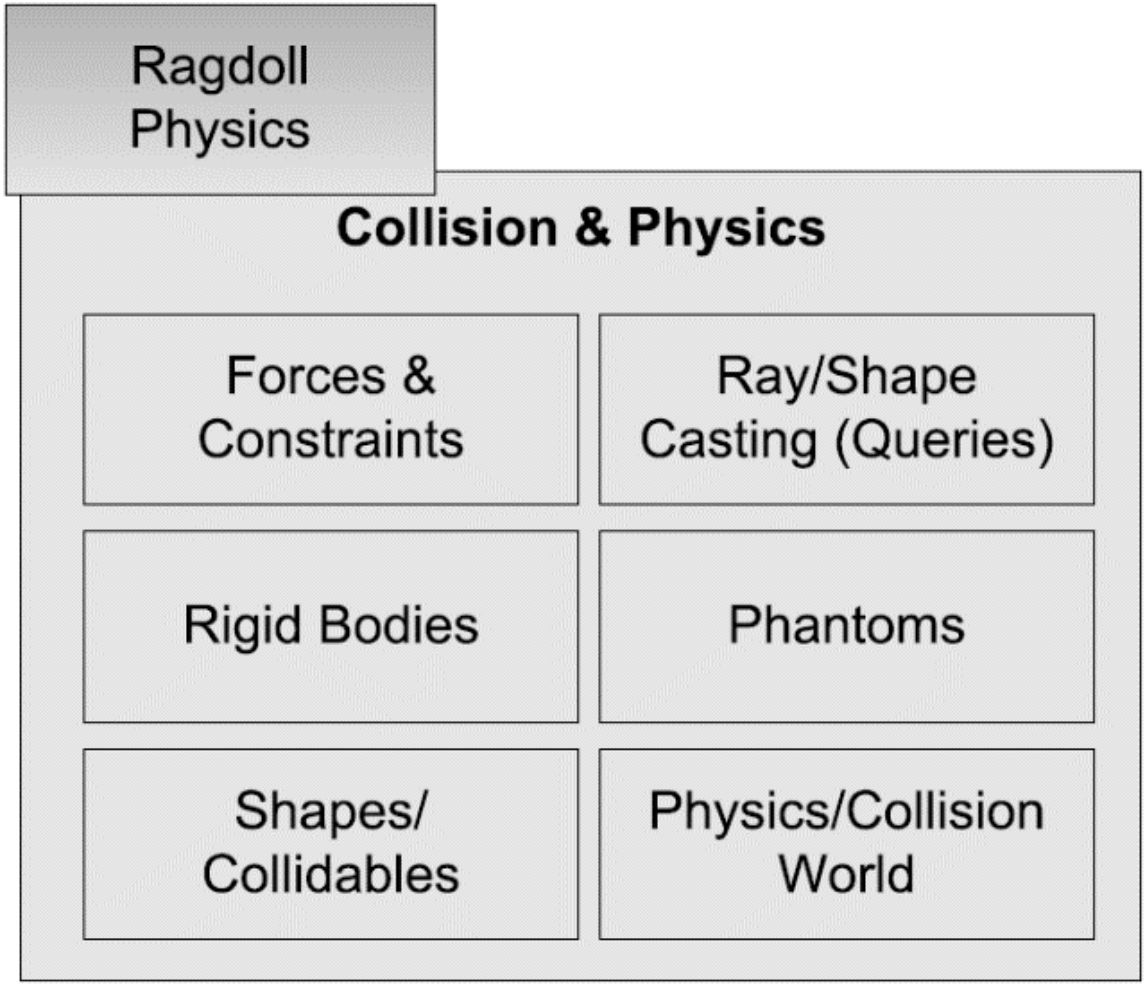
Determining memory usage

Dumping memory usage

Record and playback game events

Print statement output control





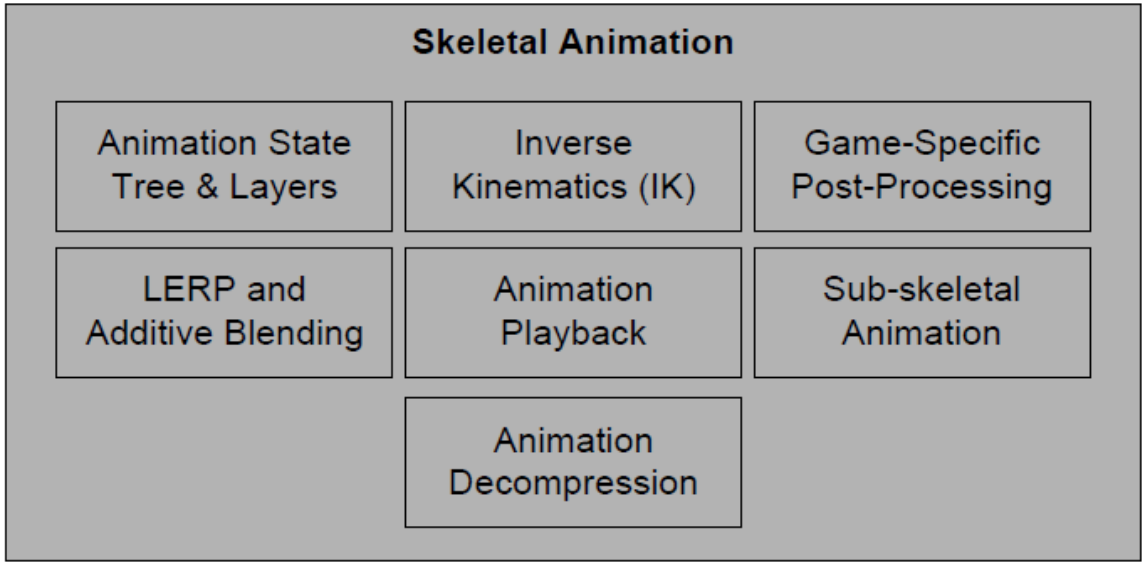
Collisions and Physics

Usually rigid body dynamics

Physics engine creation is its own unique undertaking

Many companies use available libraries

- Havok
- PhysX
- ODE
- Bullet

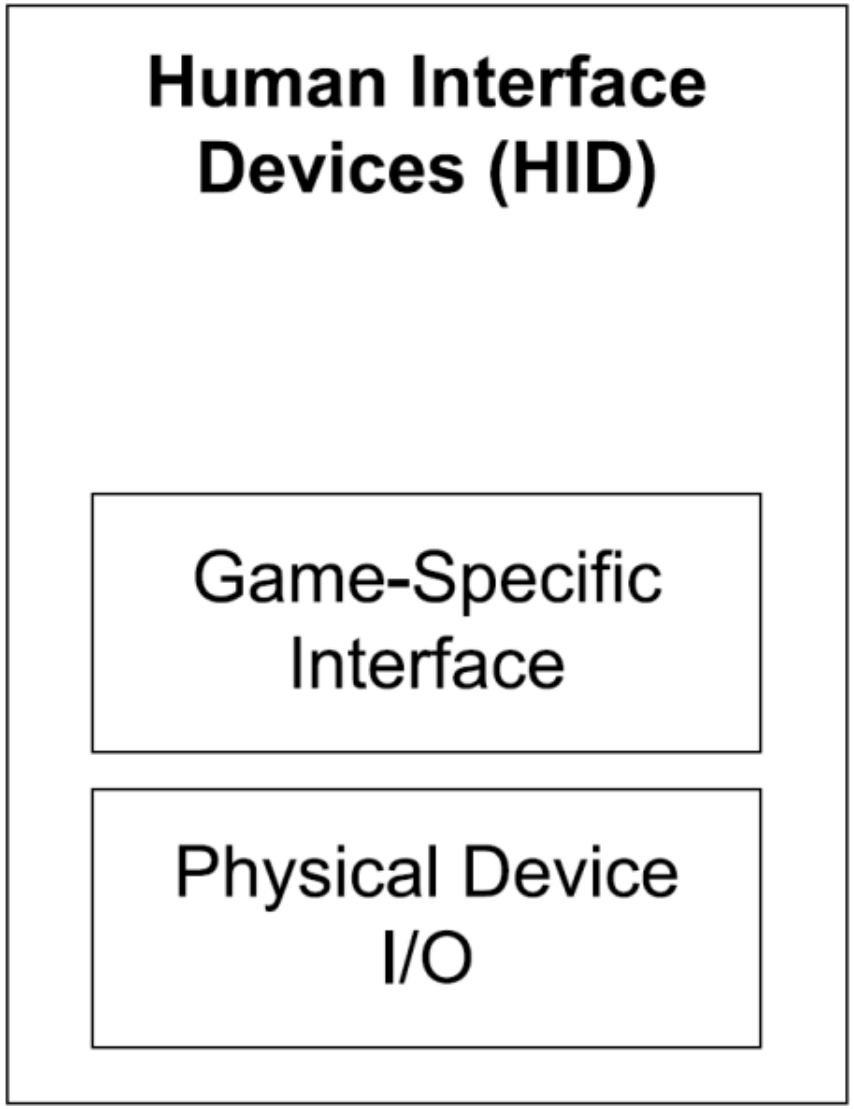


Animation

Five types of animation are used

- Sprite/texture animation
- Rigid body hierarchy animation
- Skeletal animation
- Vertex animation
- Morphing

Skeletal animations still the most popular



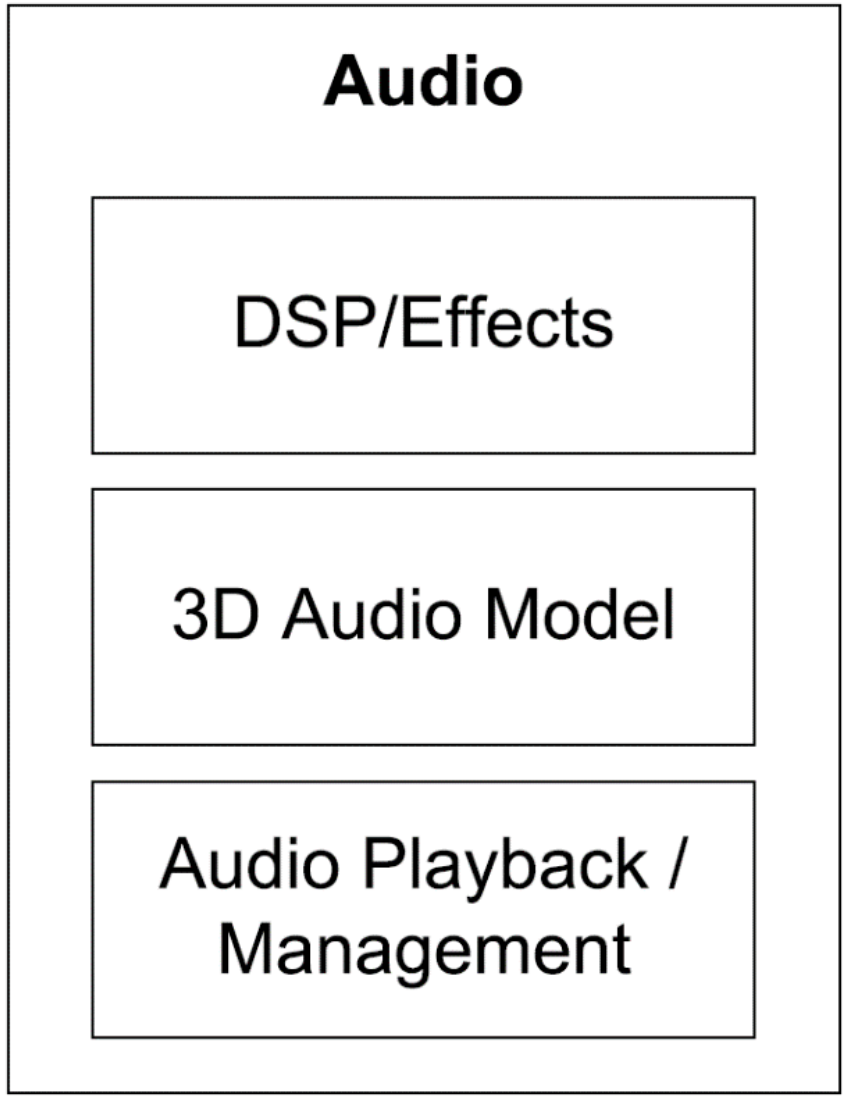
HID

Keyboard and mouse abstractions

Joypads

Specialized controllers

Massages raw data into useful information



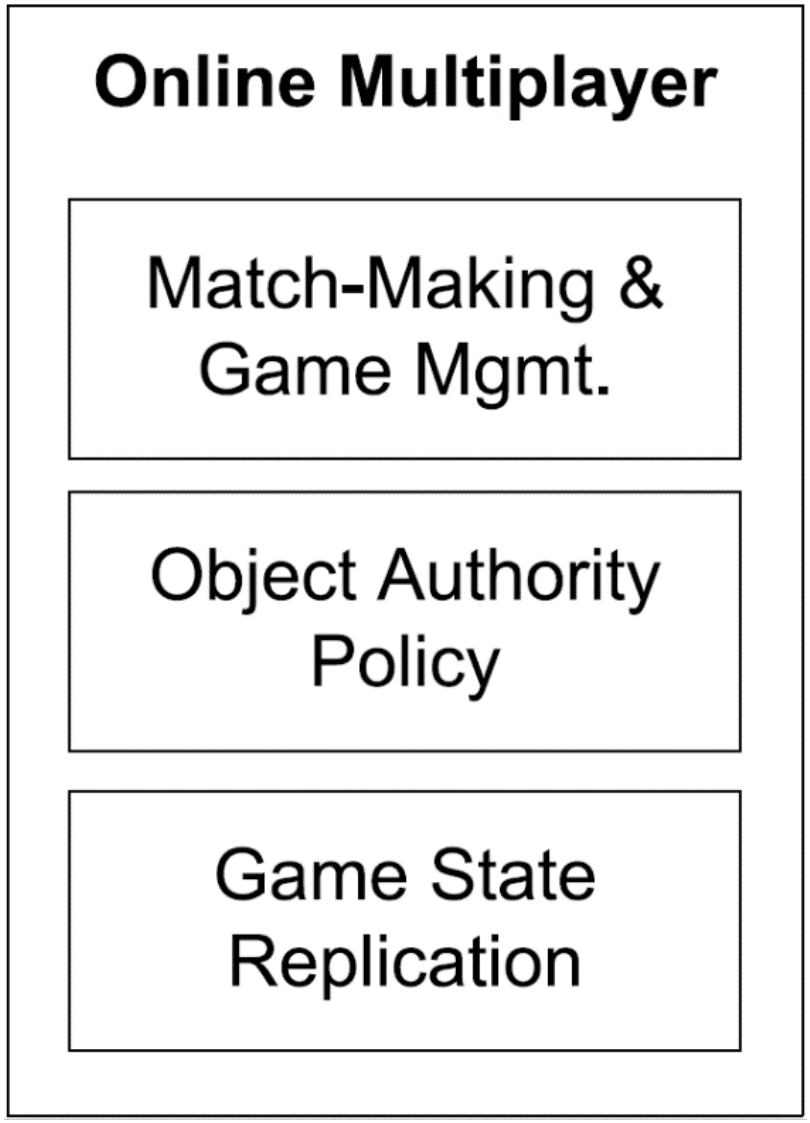
Audio

Often overlooked until the end

Varies in sophistication based on need

Many games use existing tools

- XACT
- Scream



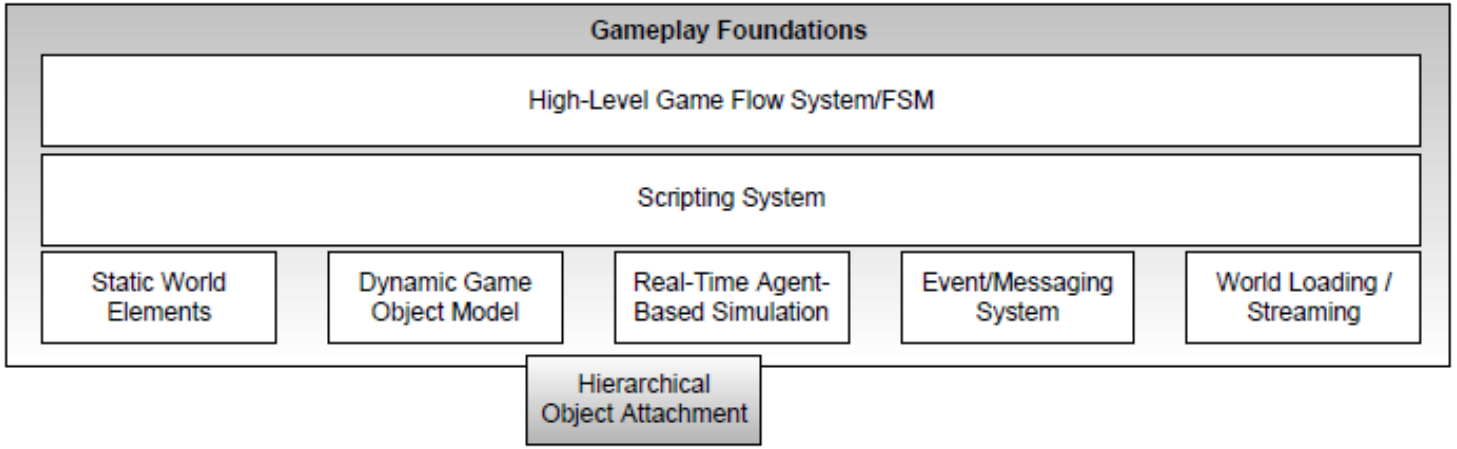
Multiplayer/networking

Four main flavors

- Single screen – multiple players on the same screen
- Split-screen multiplayer – multiple perspectives on the same screen
- Networked multiplayer – multiple computers networked together
- Massive multiplayer online games – run in a central server

Difficult to convert single to multiplayer, easy to do the opposite

Gameplay foundation system



Most everything that makes the game a game

World loading

Game object model

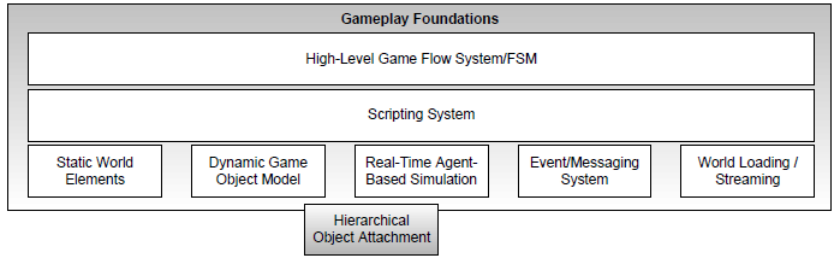
Static world elements

Real-time agent simulations

Gameplay foundation system

Game Worlds and Object Models:

- This constitutes the basic entities that make up the game's world. Here are some examples:
  - static background objects (buildings, terrain, roads)
  - (potentially) dynamic background objects (rocks, chairs, doors and windows)
  - player characters (PCs) and non-player characters (NPCs)
  - weapons and projectiles
  - vehicles
  - graphics elements (camera and lights)



Event system

Objects need to communicate with one another

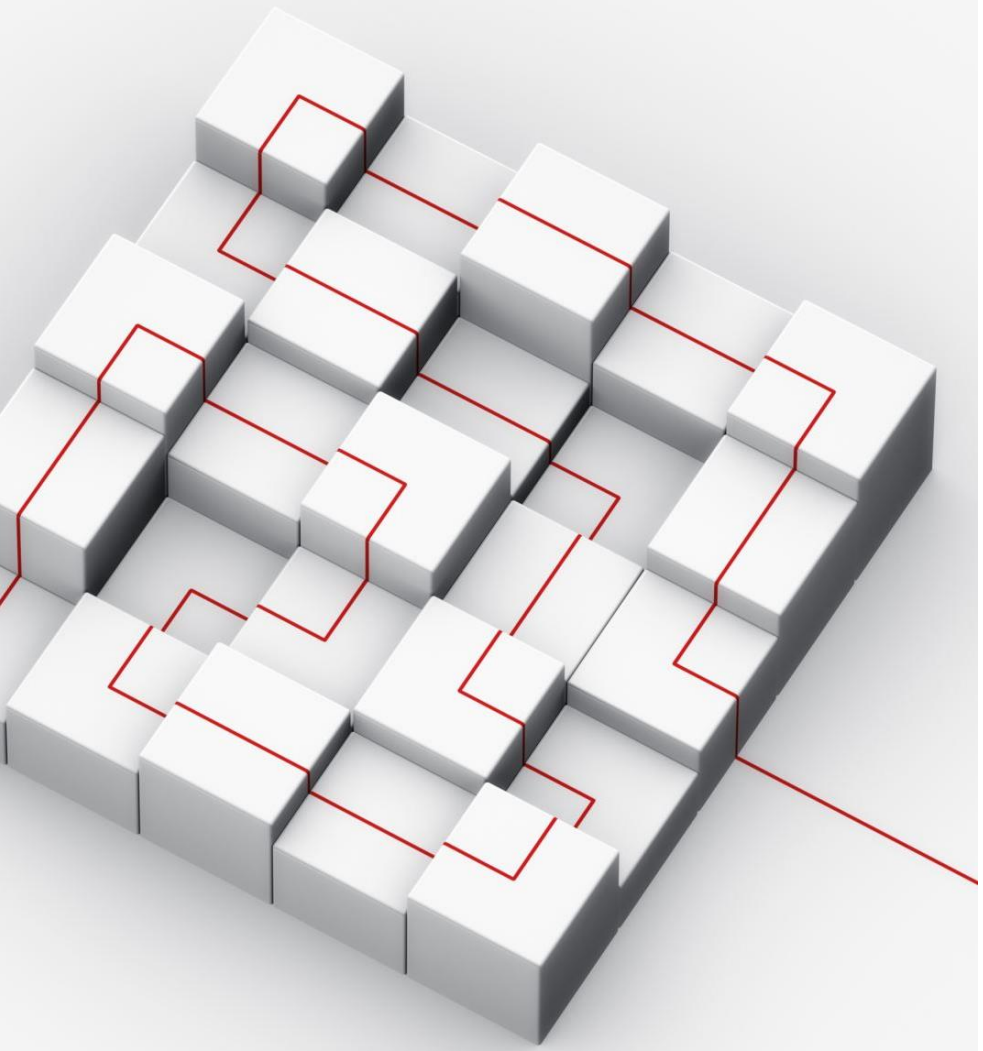
Easiest to handle this through a common system

Objects send messages that are routed to the event handler

Scripting system

Allows for the creation on new game logic without recompiling

Speeds software development considerably



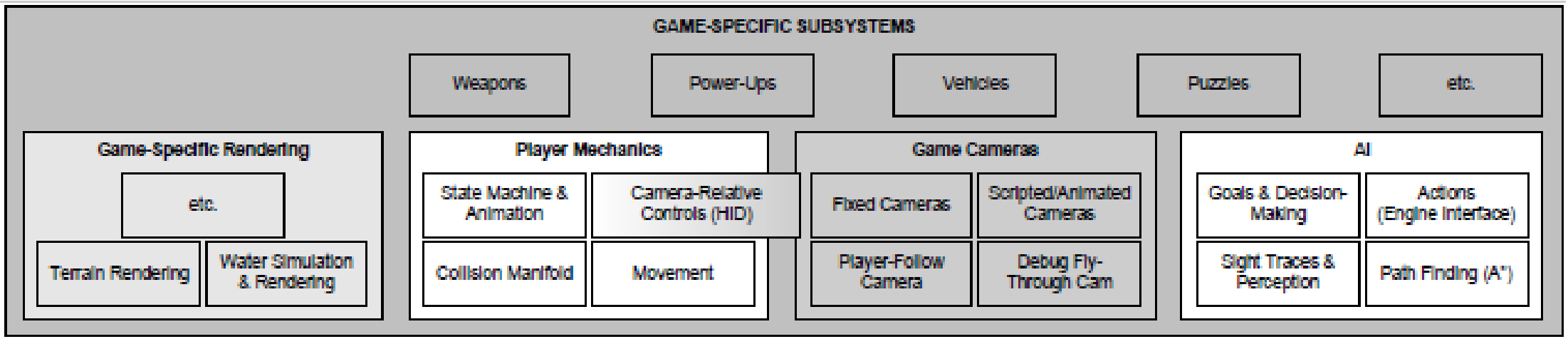
Artificial intelligence foundations

Provides AI building blocks

- Path planning
- Nav mesh generation
- Object avoidance

Autodesk has a middleware called *Gameware* that provides many of these features

Game-specific subsystems



All of the specific stuff needed for a game

This layer could be considered outside of the game engine itself

Digital content creation

Game engines deal with data in many forms

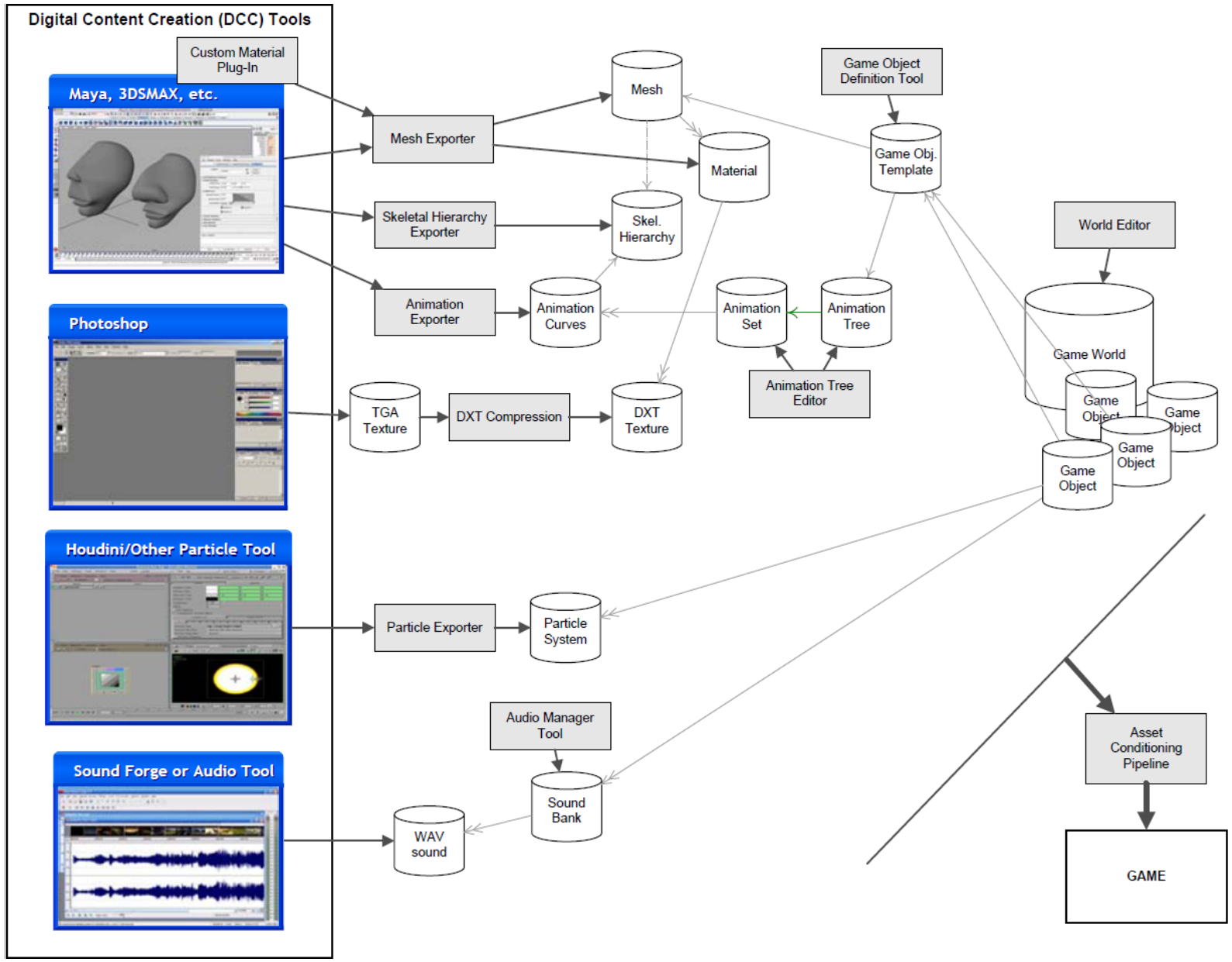
The data has to be created somehow

- 3D Meshes
- Textures
- Sound
- Animations

Often created using outside tools

- Maya/3ds Max
- Photoshop
- SoundForge

DCC tools need to be easy to use and very reliable



Tools and the asset pipeline

Assets conditioning pipeline

DCC tools produce a variety of file formats that are not optimized for game

Game engines usually store the data in an easy to read and platform specific format

The ACP does this translation

Conditioning assets

3D Model/Mesh data	Skeletal animation data	Audio data	Particle system data
• 3D Models – Must be properly tessellated • Brush Geometry – A collection of convex hulls with multiple planes	• Must be compressed and converted to properly work	• Should convert multiple formats to a single format for the target system	• Usually need a custom editing tool within the engine

Game world editor

Usually integrated into the engine

Essential to allow game designers to work with the engine

Resource database

Need a way to store and manage the vast amounts of data

Some companies use relational databases

- MySQL or Oracle

Some companies use version control software

- Subversion, Perforce, or GIT

Still others use custom software

- Naughty Dog uses a custom GUI called Builder

Game engine: main goals

Provide for underlying technologies

- Graphics Rendering
- Physics engine
- Sound
- Scripting
- Animation
- Artificial Intelligence
- Networking

Simplify development process

- Run on multiple platforms

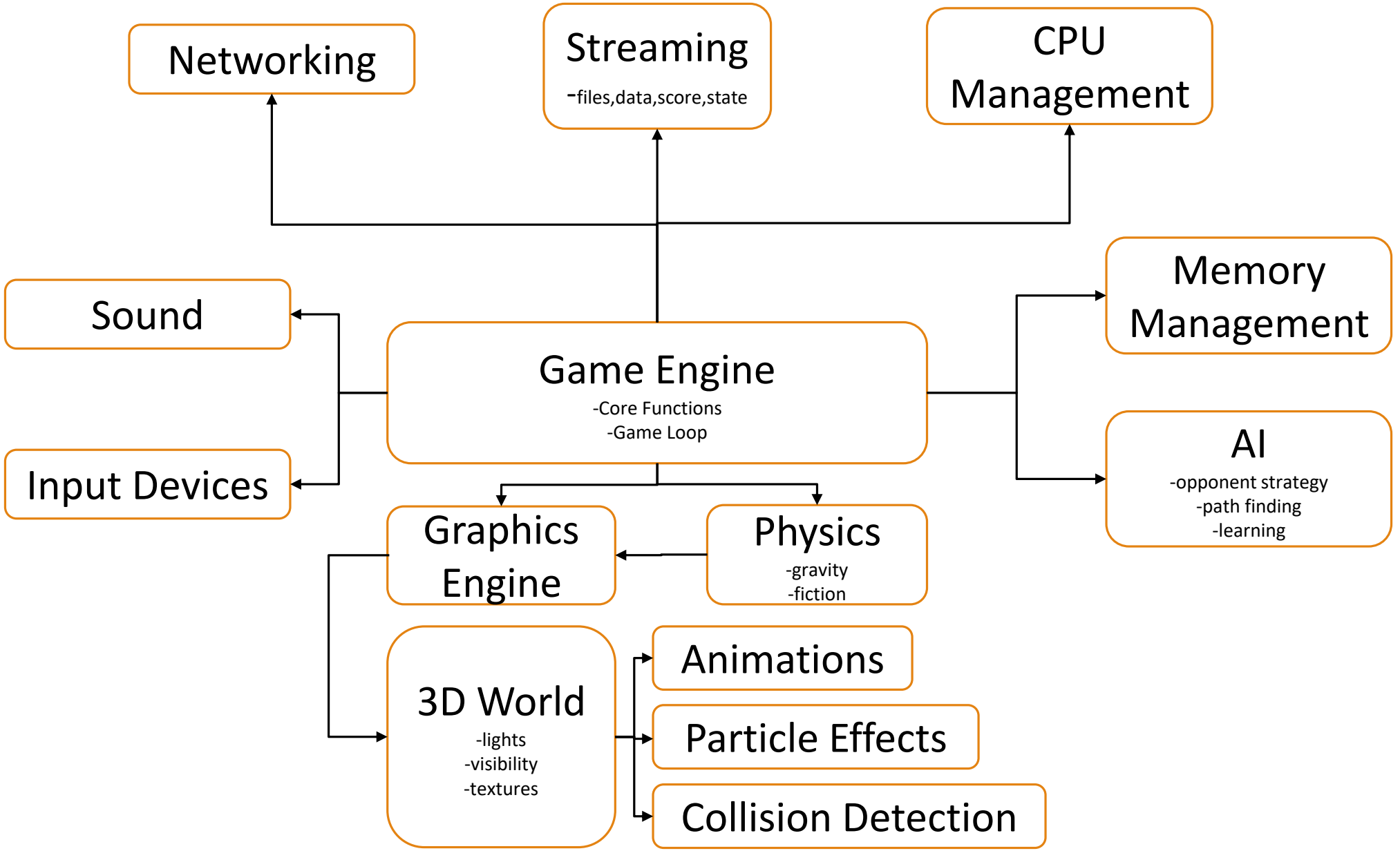
Pros & Cons

Advantages in using a game engine

- Less development time required
- Less testing and debugging
- Many features directly available
- Better focus on the game design
- Other libraries/toolkits linked with the game engine (physics, AI...)

Disadvantages in using a game engine

- No or less control over the implementation of features
- Adding features not yet in the game engine might be cumbersome
- Dependent on other licensing scheme for release
- Other libraries/toolkits linked with the game engine (physics, AI...)


Networking
Streaming
-files,data,score,state
CPU Management
Sound
Input Devices
Game Engine
-Core Functions
-Game Loop
Memory Management
AI
-opponent strategy
-path finding
-learning
Graphics Engine
Physics
-gravity
-fiction
3D World
-lights
-visibility
-textures
Animations
Particle Effects
Collision Detection

Next Week

Game Loops and Realtime Simulation

Introduction to Unity3D

Unity Basic Concepts

- Project, Scenes, Packages, Prefabs,Game Objects, Components, Assets, Scripts..

Overview of the Unity IDE(L)

- Scene View, Game View, Inspector, Hierarchy, Project