**Lecture 02**

Layered Architecture of Game Engine
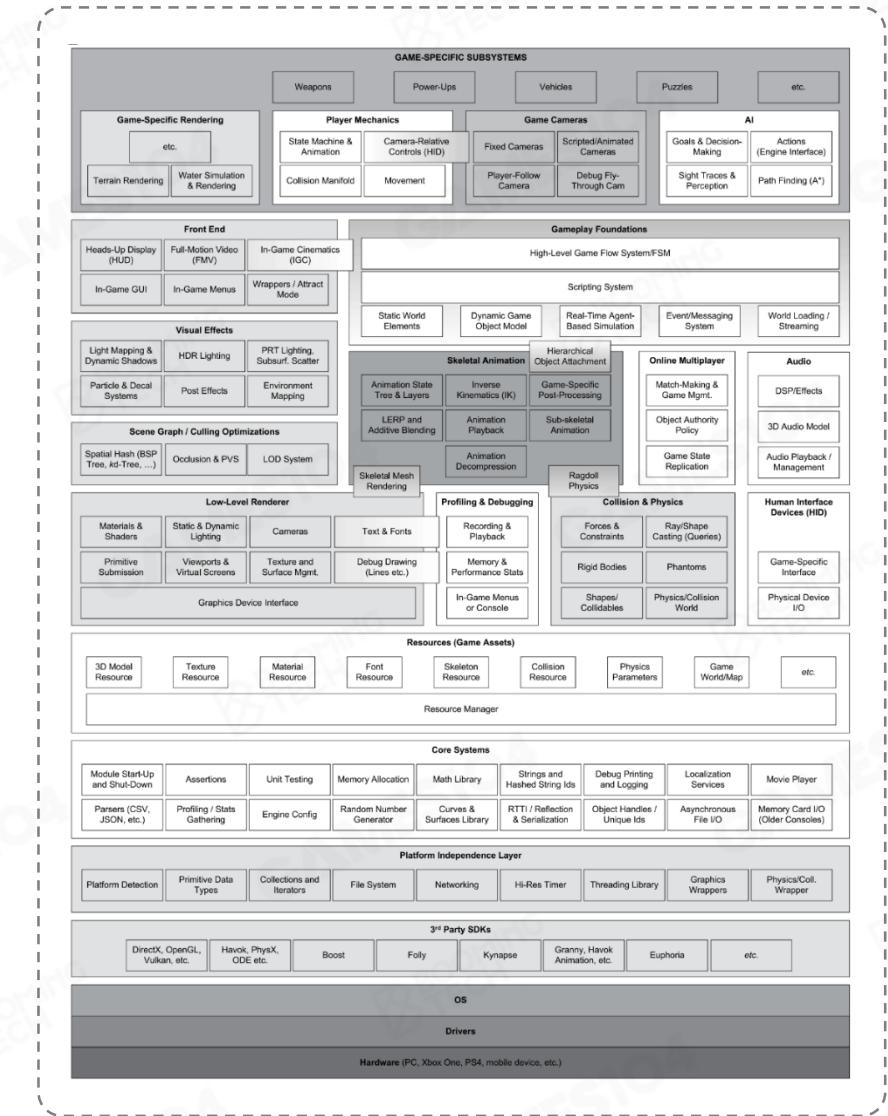
Modern Game Engine - Theory and Practice

Sea of Codes

Where to begin?
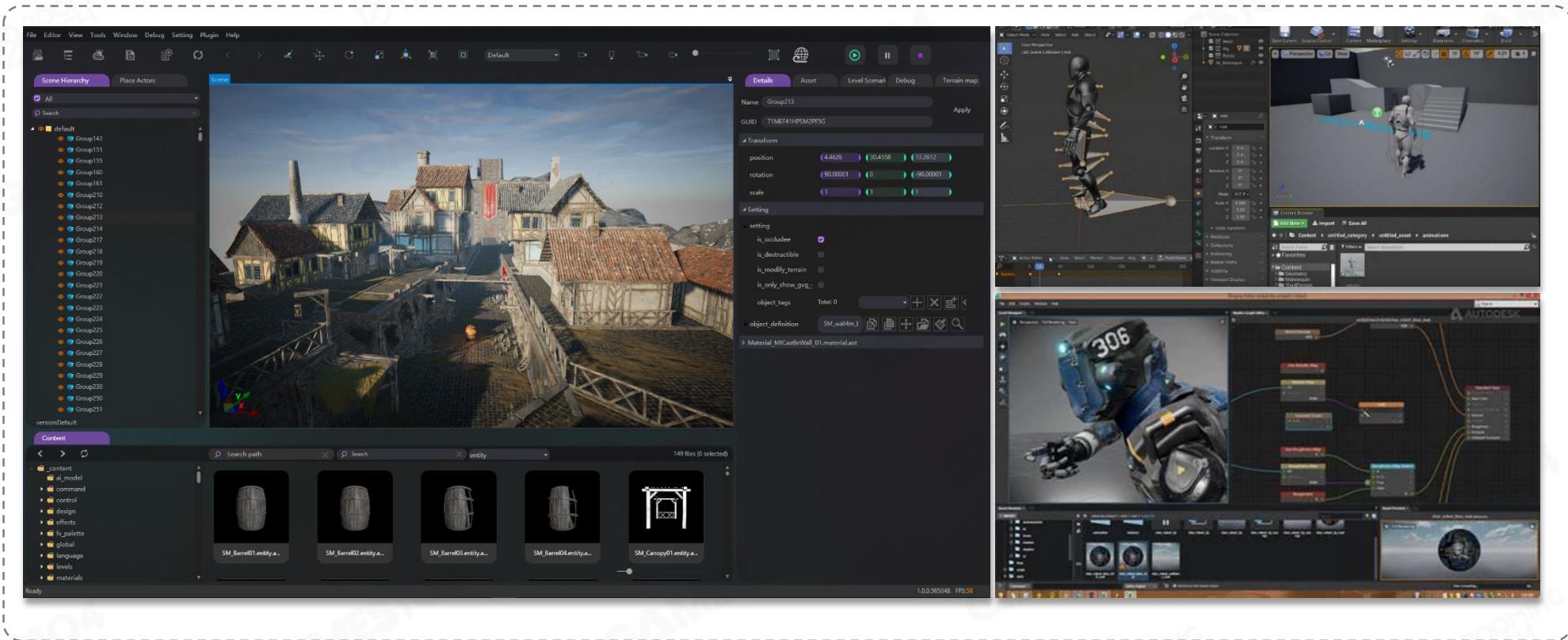
A Glance of Game Engine Layers

Chain of Editors



Tool Layer
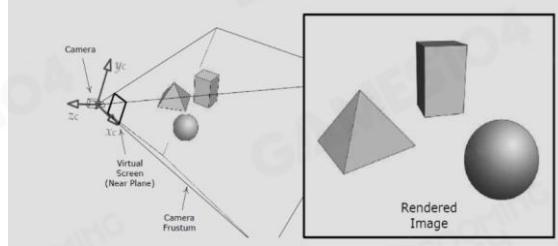
Make It Visible, Movable and Playable
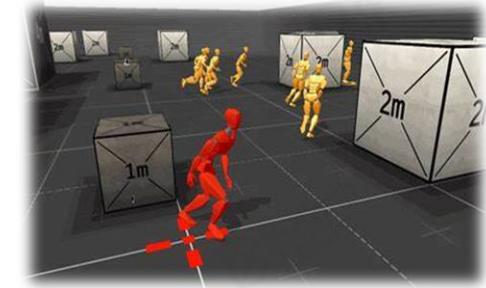
Rendering

Camera, HUD and Input

Animation

Physics

Script, FSM and AI
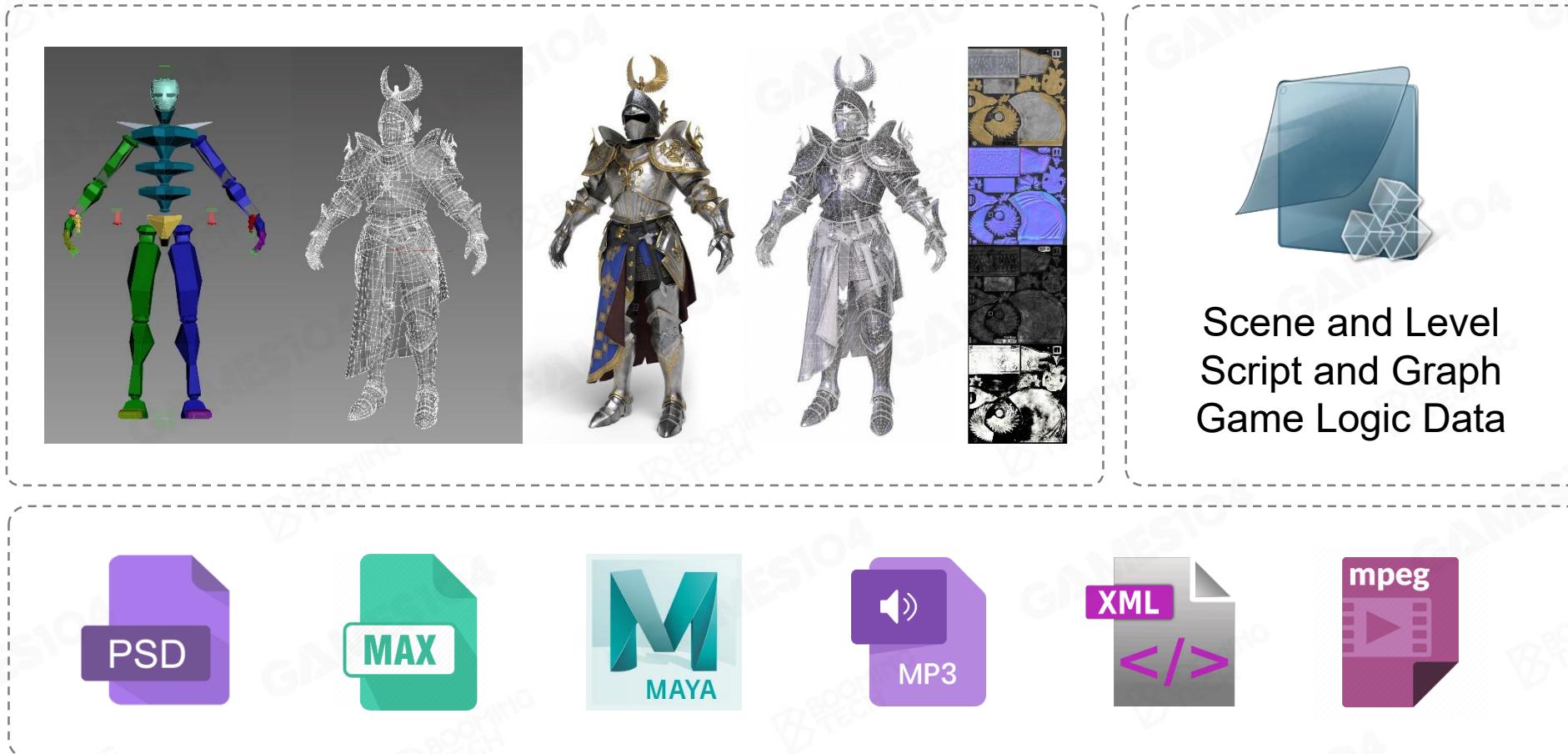
Tool Layer

Function Layer

Data and Files

Scene and Level
Script and Graph
Game Logic Data

Tool Layer

Function Layer

Resource Layer

PSD

MAX

MAYA

MP3

XML </>

mpeg
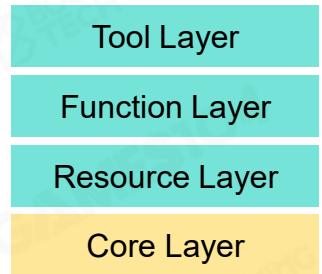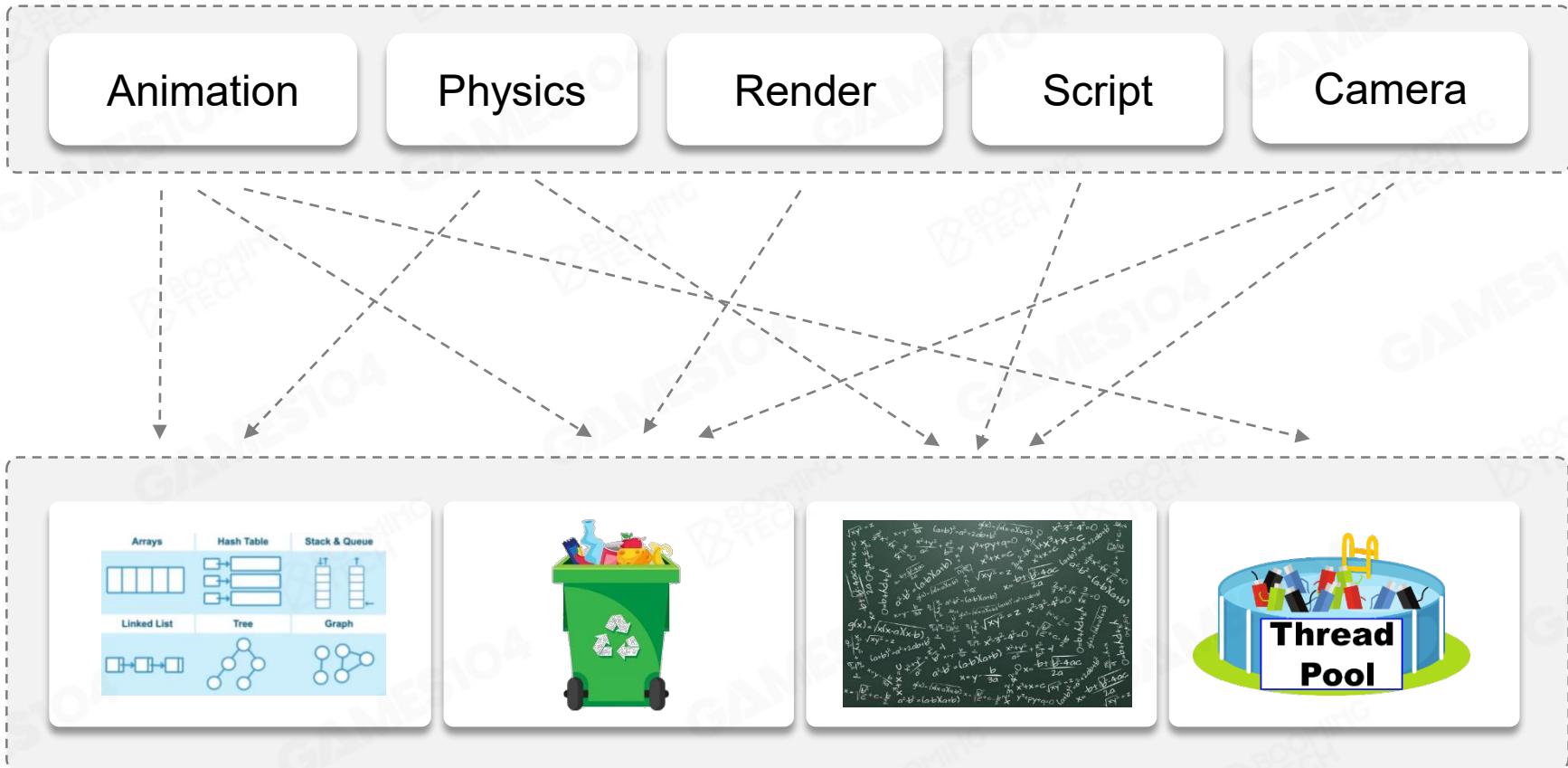
Swiss Knife of Game Engine

| Animation | Physics | Render | Script | Camera |

Tool Layer

Function Layer

Resource Layer

Core Layer

Launch on Different Platforms

Operation Systems

Platform File Systems

Graphics API

Platform SDK

…

Consoles

Input Devices

EPIC GAMES | STEAM

Publishing Platforms

Tool Layer

Function Layer
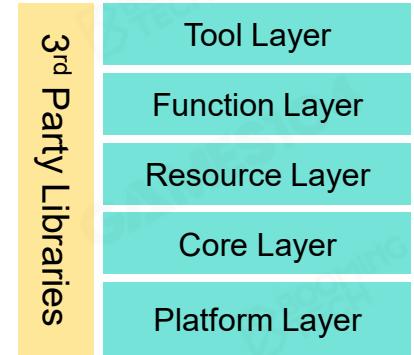
Resource Layer

Core Layer

Platform Layer

Middleware and 3ʳᵈ Party Libraries

3ʳᵈ Party Libraries	Tool Layer
	Function Layer
	Resource Layer
	Core Layer
	Platform Layer

Explore Game Engine Layers

Think I'm ready to roll!

Practice is the Best Way to Learn

Simple Animated Character Challenge

- Create, animate and render a character
- Playable on selected hardware platforms

I want to build an animation system
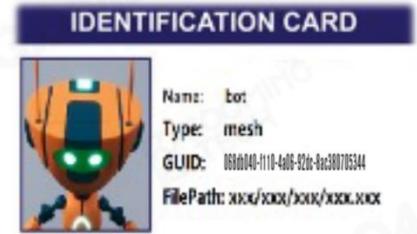
Resource - How to Access My Data

```xml
<character name="robot" GUID="068db040-f110-4a06-92dc-8ac380705344">
  <geometry>
    <mesh file_path="robot.mesh.ast"/>
    <texture>
      <abledo_texture file_path="robot_ambient.texture.ast"/>
      <roughness_texture file_path="robot_roughness.texture.ast"/>
    </texture>
    <material file_path="robot.material.ast"/>
  </geometry>
  <animation>
    <animation_clip name="stand">
      <clip_file path="robot_stand.animation.ast"/>
    </animation_clip>
    <animation_clip name="walk">
      <clip_file path="robot_walk.animation.ast"/>
    </animation_clip>
  </animation>
</character>
```
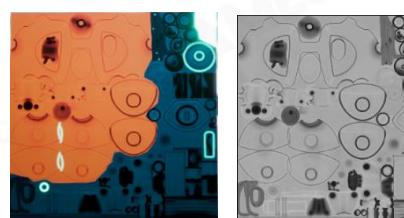
Offline Resource Importing
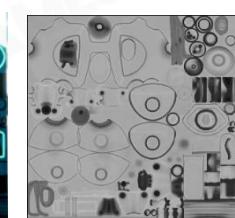
- Unify file access by defining a meta asset file format (ie.ast)
- Assets are faster to access by importing preprocess
- Build a composite asset file to refer to all resources
- GUID is an extra protection of reference

IDENTIFICATION CARD

Name: bot
Type: mesh
GUID: 068db040-f110-4a06-92dc-8ac380705344
FilePath: xxx/xxx/xxx/xxx.xxx
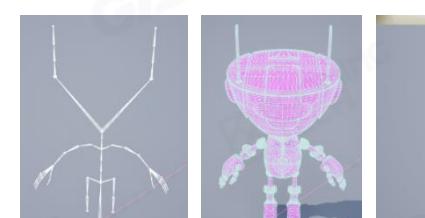
GUID

texture.ast   character.ast

mesh.ast   animation.ast

XML
</>

PSD

MAX

MAYA

3rd Party Libraries

| Tool Layer |
| Function Layer |
| Resource Layer |
| Core Layer |
| Platform Layer |
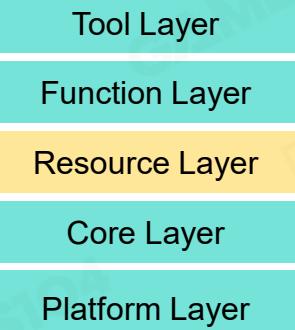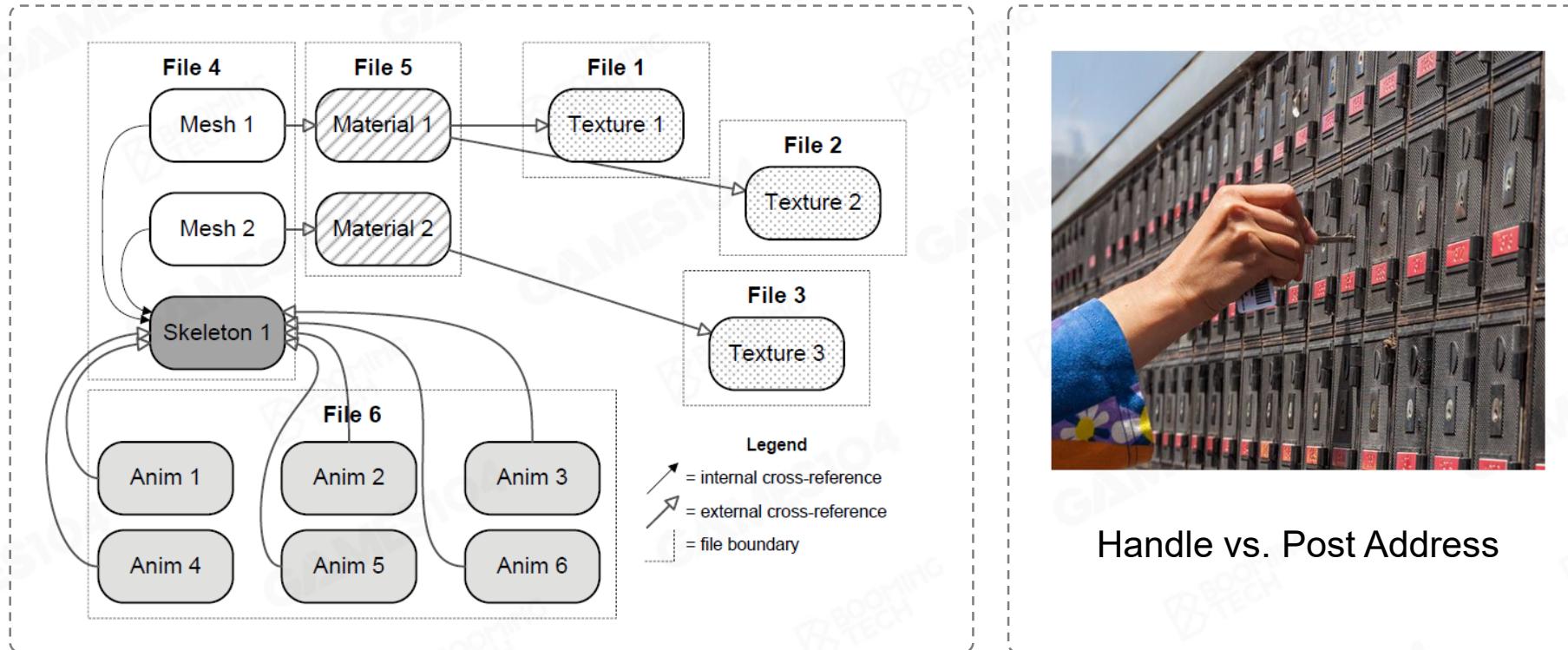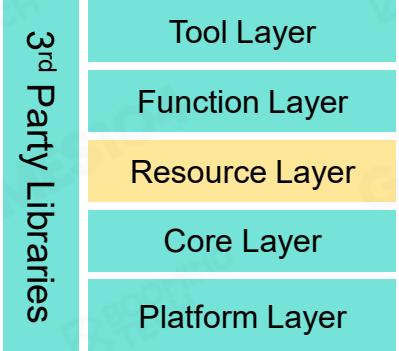
Resource – Runtime Asset Manager

Runtime Resource Management

- A virtual file system to load/unload assets by path reference
- Manage asset lifespan and reference by **handle** system

Handle vs. Post Address

3rd Party Libraries

| Tool Layer |
| Function Layer |
| Resource Layer |
| Core Layer |
| Platform Layer |

Resource – Manage Asset Life Cycle
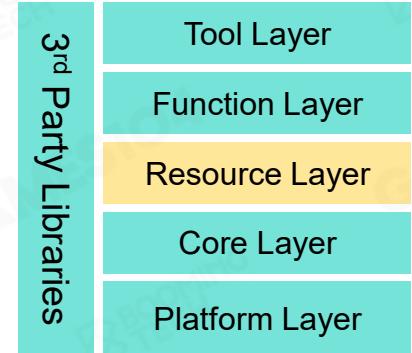
Character

Scene

Cutscene

3rd Party Libraries	Tool Layer
	Function Layer
	Resource Layer
	Core Layer
	Platform Layer
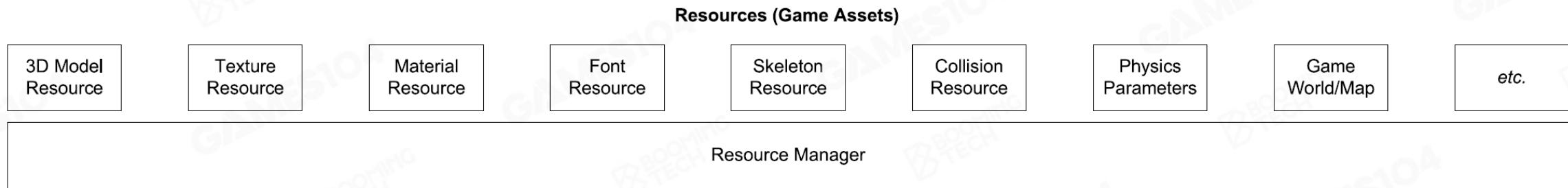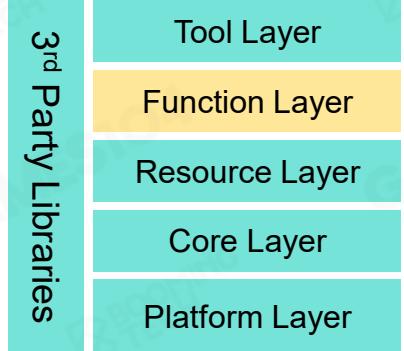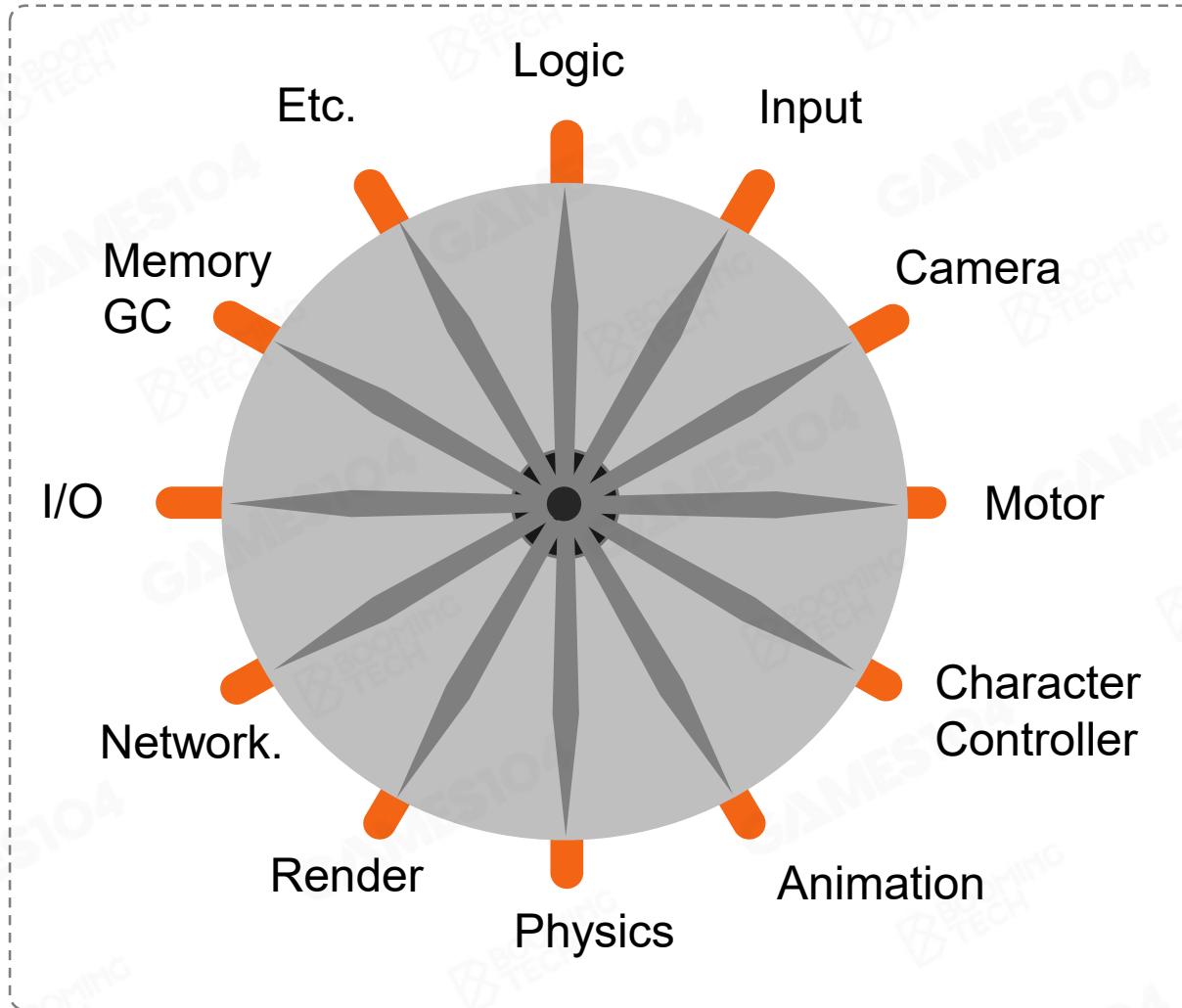
Memory management for Resources - life cycle

- Different resources have different life cycles

- Limited memory requires release of loaded resources when possible

- Garbage collection and deferred loading is critical features

Resources (Game Assets)

| 3D Model Resource | Texture Resource | Material Resource | Font Resource | Skeleton Resource | Collision Resource | Physics Parameters | Game World/Map | *etc.* |

| Resource Manager |
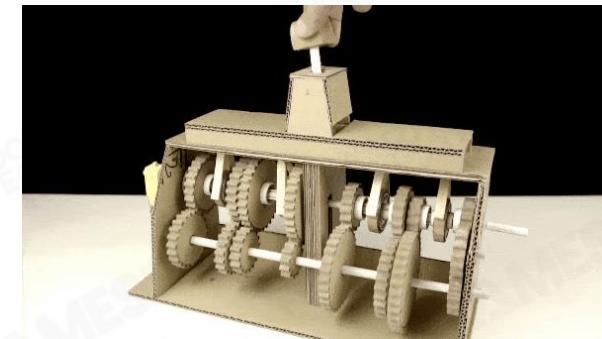
Function - How to Make the World Alive

3rd Party Libraries	Tool Layer
	Function Layer
	Resource Layer
	Core Layer
	Platform Layer

Logic

Etc.

Input

Memory GC

Camera

I/O

Motor

Network.

Character Controller

Render

Animation

Physics
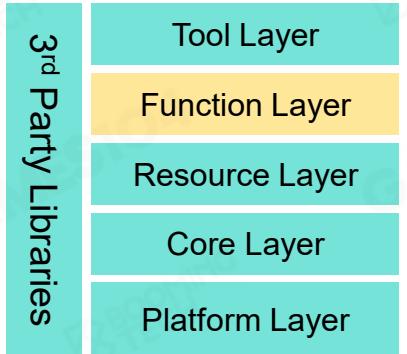
Unbelievably Simple!!!

- Transmission shaft game_main.

  cppvoid tick(int delta_time)

- Endless loop

  while (true) { … }

Function - Dive into Ticks

3rd Party Libraries

| Tool Layer |
| Function Layer |
| Resource Layer |
| Core Layer |
| Platform Layer |

```cpp
void tickMain(float delta_time)
{
    while (!exit_flag)
    {
        tickLogic(delta_time);

        tickRender(delta_time);
    }
}
```

```cpp
void tickLogic(float delta_time)
{
    tickCamera(delta_time);

    tickMotor(delta_time);

    tickController(delta_time);

    tickAnimation(delta_time);

    tickPhysics(delta_time);

    /*...*/
}
```

```cpp
void tickRender(float delta_time)
{
    tickRenderCamera();

    culling();

    rendering();

    postprocess();

    present();
}
```

Function -Tick the Animation and Renderer

- In each tick (over-simplified version)
  - Fetch animation frame of character
  - Drive the skeleton and skin of character
  - Renderer process all rendering jobs in an iteration of render tick for each **frame**

|	
3rd Party Libraries	Tool Layer
	Function Layer
	Resource Layer
	Core Layer
	Platform Layer
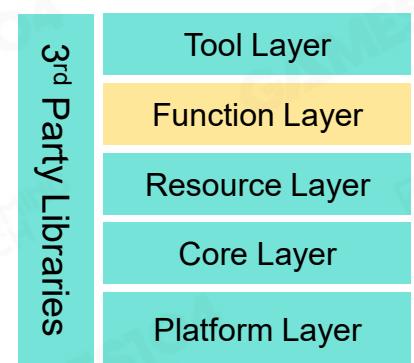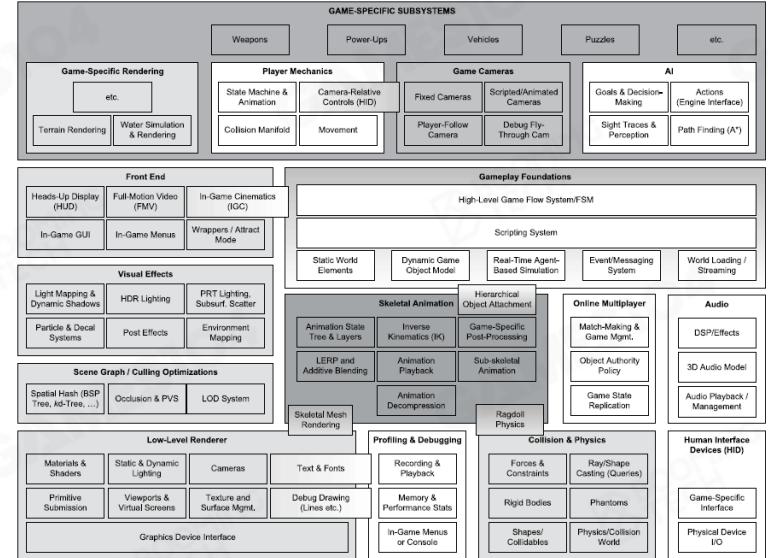
Function - Heavy-duty Hotchpotch
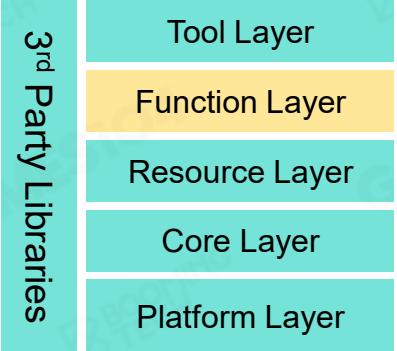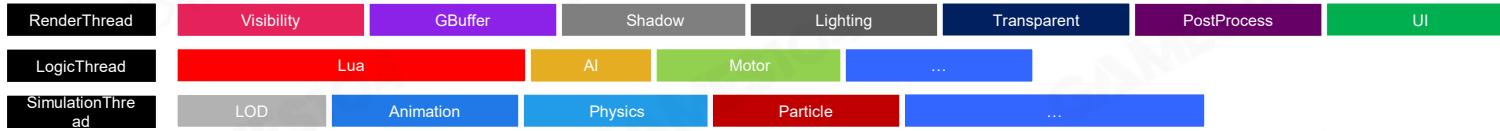
- Function Layer provides major function modules for the game engine
  - Object system (HUGE)

- Game Loop updates the systems periodically
  - Game Loop is the key of reading codes of game engines

- Blur the boundary between engine and game
  - Camera, character and behavior
  - Design extendable engine API for programmer

| 3rd Party Libraries | Tool Layer |
	Function Layer
	Resource Layer
	Core Layer
	Platform Layer
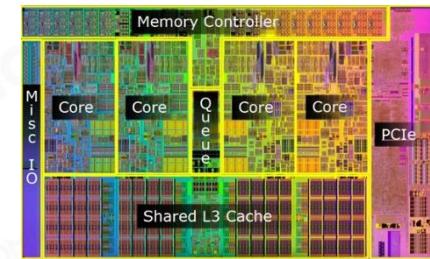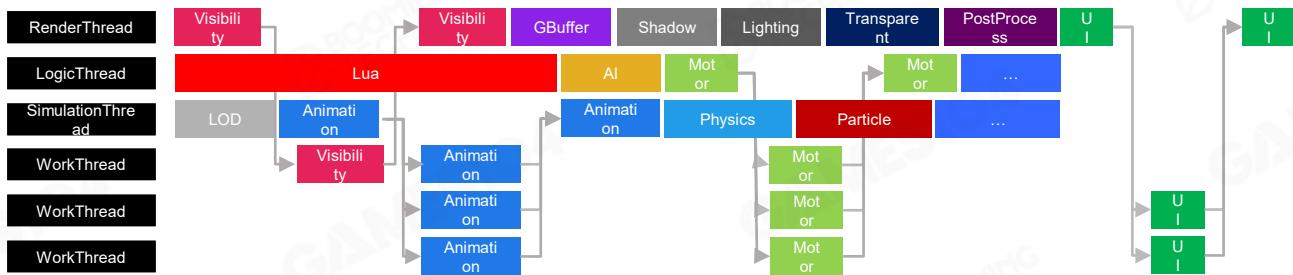
Function - Multi-Threading

Entry
Fixed Thread

Mainstream
Thread Fork/Join

Advanced
JOB System

3rd Party Libraries

- Tool Layer
- Function Layer
- Resource Layer
- Core Layer
- Platform Layer
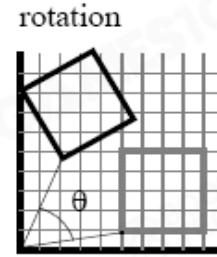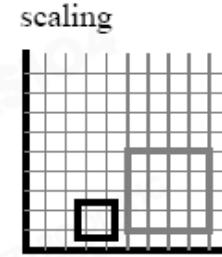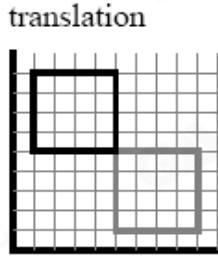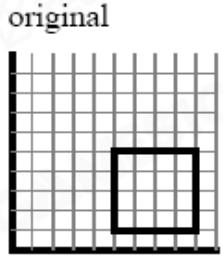
Multi-Core CPU

- Multi-core processors become the mainstream
  - Many systems in game engine are built for parallelism

Core - Math Library

original · translation · scaling · rotation

$$T_{\mathbf{v}}\mathbf{p} = \begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x + v_x \\ p_y + v_y \\ p_z + v_z \\ 1 \end{bmatrix} = \mathbf{p} + \mathbf{v}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$S_v p = \begin{bmatrix} v_x & 0 & 0 & 0 \\ 0 & v_y & 0 & 0 \\ 0 & 0 & v_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} v_x p_x \\ v_y p_y \\ v_z p_z \\ 1 \end{bmatrix}.$$

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta.$$

3rd Party Libraries	Tool Layer
	Function Layer
	Resource Layer
	Core Layer
	Platform Layer

Linear algebra

- Rotation, translation, scaling

- Matrix splines, quaternion

Core - Math Efficiency

Quick and dirty hacks
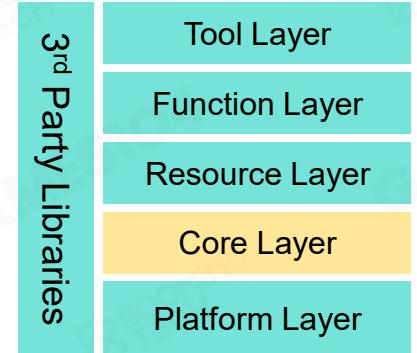
- Carmack's 1/sqrt(x)
- Magic number!

```
  float Q_rsqrt(float number)
  {
      long i;
      float x2, y;
      const float threehalfs = 1.5F;

      x2 = number * 0.5F;
      y  = number;
      i  = * ( long * ) &y;
      i  = 0x5f3759df - ( i >> 1 );
      y  = * ( float * ) &i;
      y  = y * ( threehalfs - ( x2 * y * y ) );

      #ifndef Q3_VM
      #ifdef __linux__
          assert( !isnan(y) );
      #endif
      #endif
      return y;
  }
```
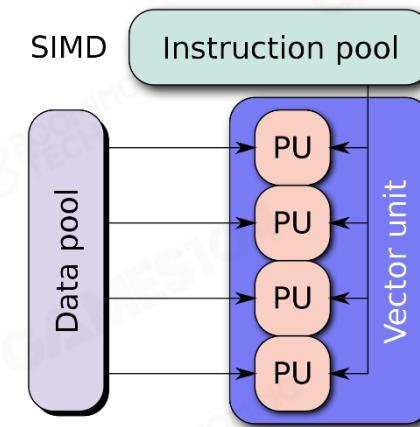
Quake III Engine

SIMD

SIMD

Instruction pool

Data pool → PU PU PU PU — Vector unit

3rd Party Libraries	Tool Layer
	Function Layer
	Resource Layer
	Core Layer
	Platform Layer
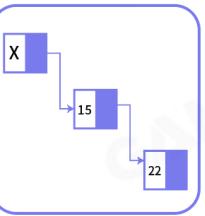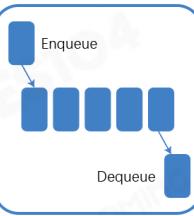
Core - Data Structure and Containers

|	
3rd Party Libraries	Tool Layer
	Function Layer
	Resource Layer
	Core Layer
	Platform Layer

Array
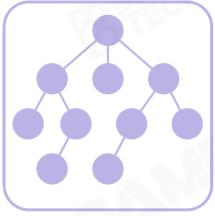
Link list

Queue

Heap
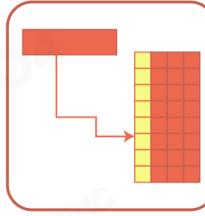
Tree

Graph
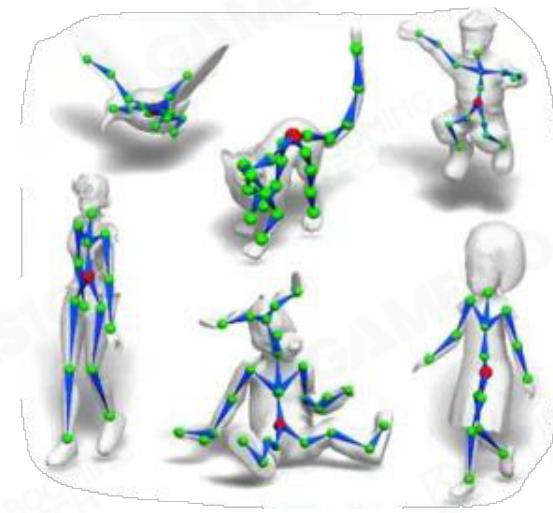
Stack
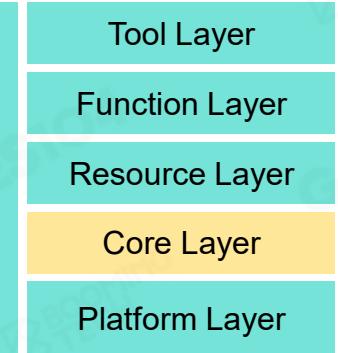
Hashing

- Vectors, maps, trees, etc.
- Customized outperforms STL
- Avoid FRAGMENT memory!

- Skeleton tree
- Animation frame sequence
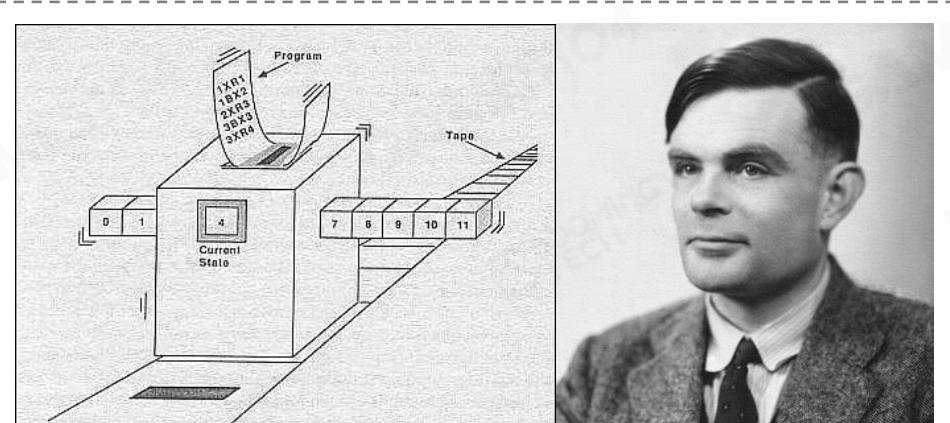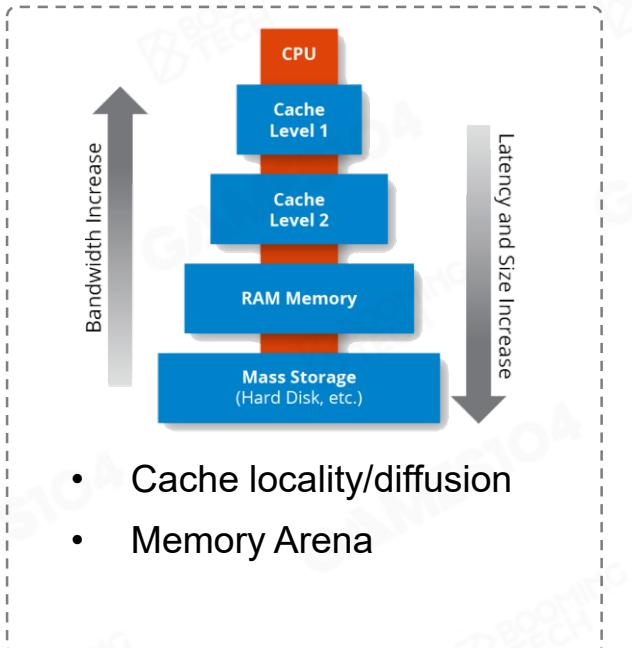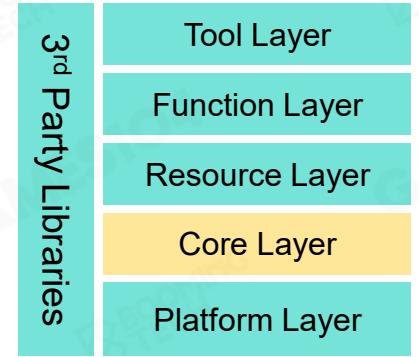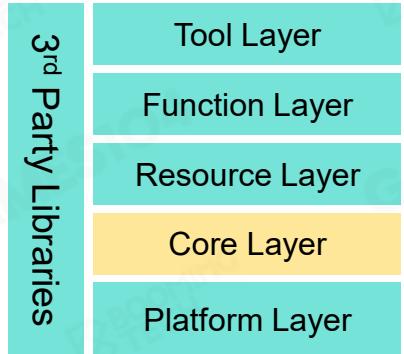
Core - Memory Management

- Major bottlenecks of game engine performance
  - Memory Pool / Allocator
  - Reduce cache miss
 - Memory alignment

- Polymorphic Memory Resource (PMR)


- Cache locality/diffusion
- Memory Arena

- Put data together
- Access data in order
- Allocate and de-allocate as a block

Tool Layer
Function Layer
Resource Layer
Core Layer
Platform Layer

Core - Foundation of Game Engine

- Core layers provide utilities needed in various function modules
- Super high performance design and implementation
- High standard of coding
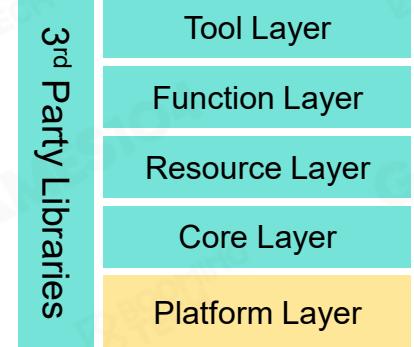
Core Systems								
Module Start-Up and Shut-Down	Assertions	Unit Testing	Memory Allocation	Math Library	Strings and Hashed String Ids	Debug Printing and Logging	Localization Services	Movie Player
Parsers (CSV, JSON, etc.)	Profiling / Stats Gathering	Engine Config	Random Number Generator	Curves & Surfaces Library	RTTI / Reflection & Serialization	Object Handles / Unique Ids	Asynchronous File I/O	Memory Card I/O (Older Consoles)

Platform - Target on Different Platform

Compatibility of different platforms, provides platform-independent services and information for upper layers

- File system
  - Path: Slash/backslash, Environment variables
  - Directory Traversal

	3rd Party Libraries
	Tool Layer
	Function Layer
	Resource Layer
	Core Layer
	Platform Layer

S:\Main Folder\Folder1\Folder2\FinalFolder

/Volumes/Share/Main Folder/Folder1/Folder2/FinalFolder

Time to show off
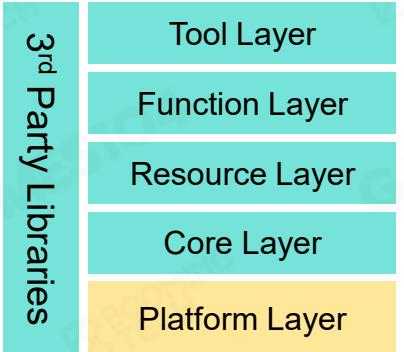
Platform - Graphics API

Render Hardware Interface (RHI)

- Transparent different GPU architectures and SDK
- Automatic optimization of target platforms

OpenGL|ES. | Microsoft DirectX11 | DirectX 12 | Vulkan | M

```cpp
// shader
virtual RHIVertexShader*        createVertexShader(const DynamicArray<UByte>& shader_bin_code) = 0;
virtual RHIHullShader*          createHullShader(const DynamicArray<UByte>& shader_bin_code) = 0;
virtual RHIDomainShader*        createDomainShader(const DynamicArray<UByte>& shader_bin_code) = 0;
virtual RHIGeometryShader*      createGeometryShader(const DynamicArray<UByte>& shader_bin_code) = 0;
virtual RHIPixelShader*         createPixelShader(const DynamicArray<UByte>& shader_bin_code) = 0;
virtual RHIComputeShader*       createComputeShader(const DynamicArray<UByte>& shader_bin_code) = 0;

// buffer
virtual RHIVertexBuffer*        createVertexBuffer(RHIResourceCreateInfo& create_info) = 0;
virtual void*                   lockVertexBuffer(RHIVertexBuffer* vertex_buffer, UInt offset, UInt size, EResourceLockMode lock_mode) = 0;
virtual void                    unlockVertexBuffer(RHIVertexBuffer* vertex_buffer) = 0;

virtual RHIIndexBuffer*         createIndexBuffer(RHIResourceCreateInfo& create_info) = 0;
virtual void*                   lockIndexBuffer(RHIIndexBuffer* index_buffer, UInt offset, UInt size, EResourceLockMode lock_mode) = 0;
virtual void                    unlockIndexBuffer(RHIIndexBuffer* index_buffer) = 0;
```
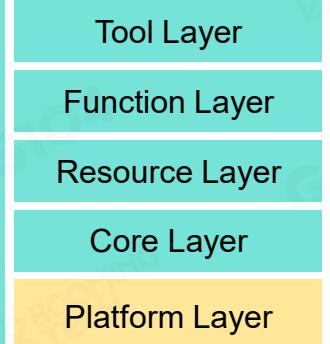
Platform - Hardware Architecture

| PPU @ 3.2 GHz | SPU 0 @ 3.2 GHz | SPU 1 @ 3.2 GHz | ... | SPU 6 @ 3.2 GHz |

L1 D$ 32 KiB 4-way | L1 I$ 32 KiB 2-way

Local Store 256 KiB | Local Store 256 KiB | Local Store 256 KiB

L2 Cache 512 KiB

DMA Ring Bus

DMA Controller

System RAM 256 MiB XDR

Video RAM 256 MiB GDDR3

Nvidia RSX GPU @ 550 MHz

- Core variants: PPU and SPUs
- UMA: unified memory access
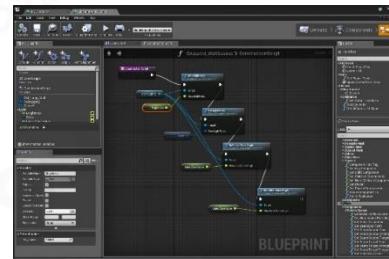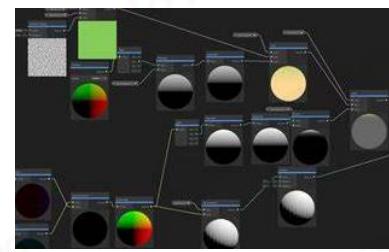
# Tool - Allow Anyone to Create Game

Level Editor
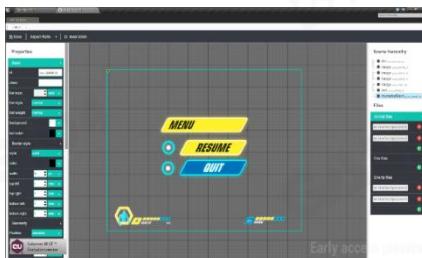
Logical Editor

Shader Editor

Animation Editor

UI Editor

Unleash the Creativity

- Build upon game engine
- Create, edit and exchange game play assets

Flexible of coding languages
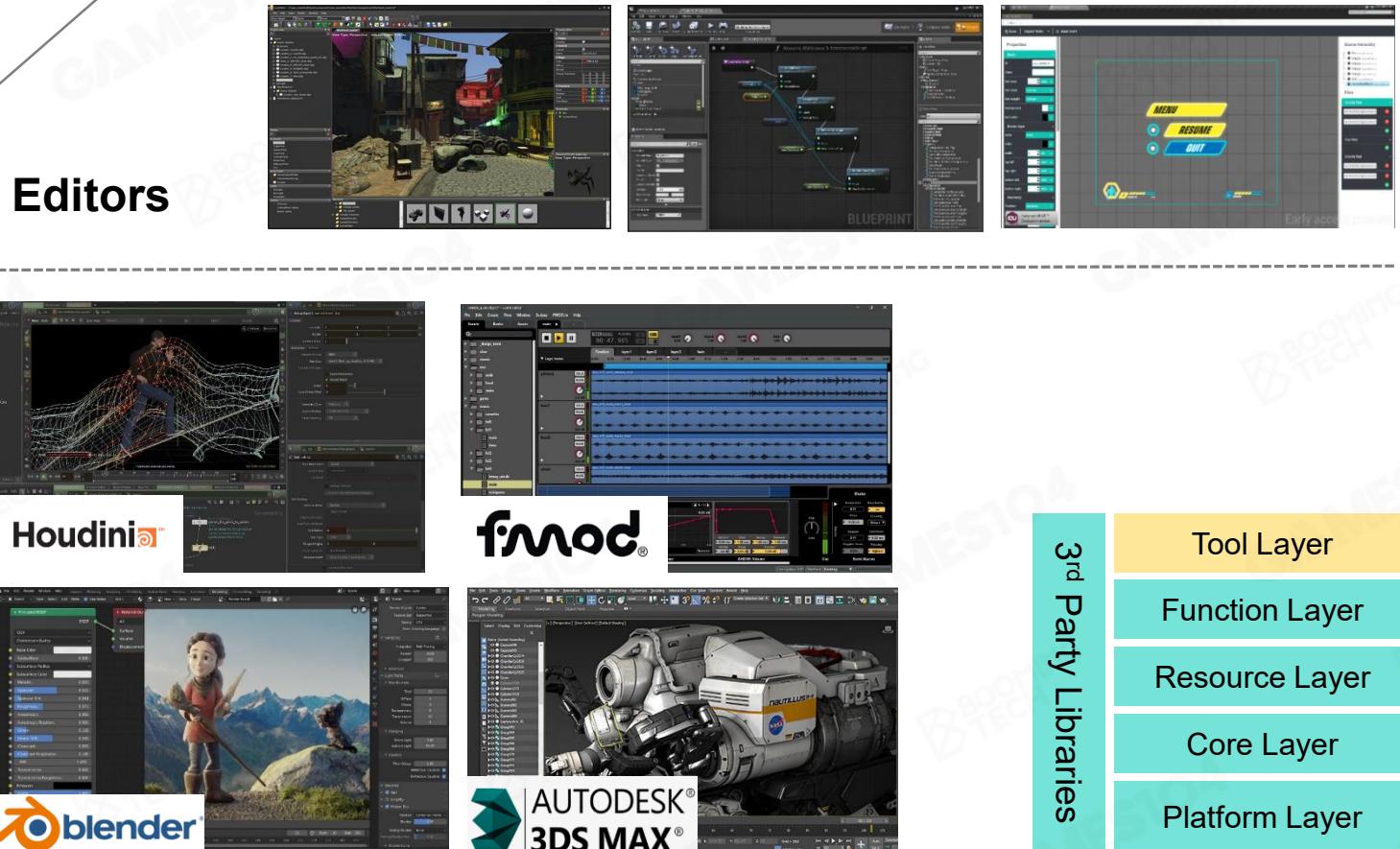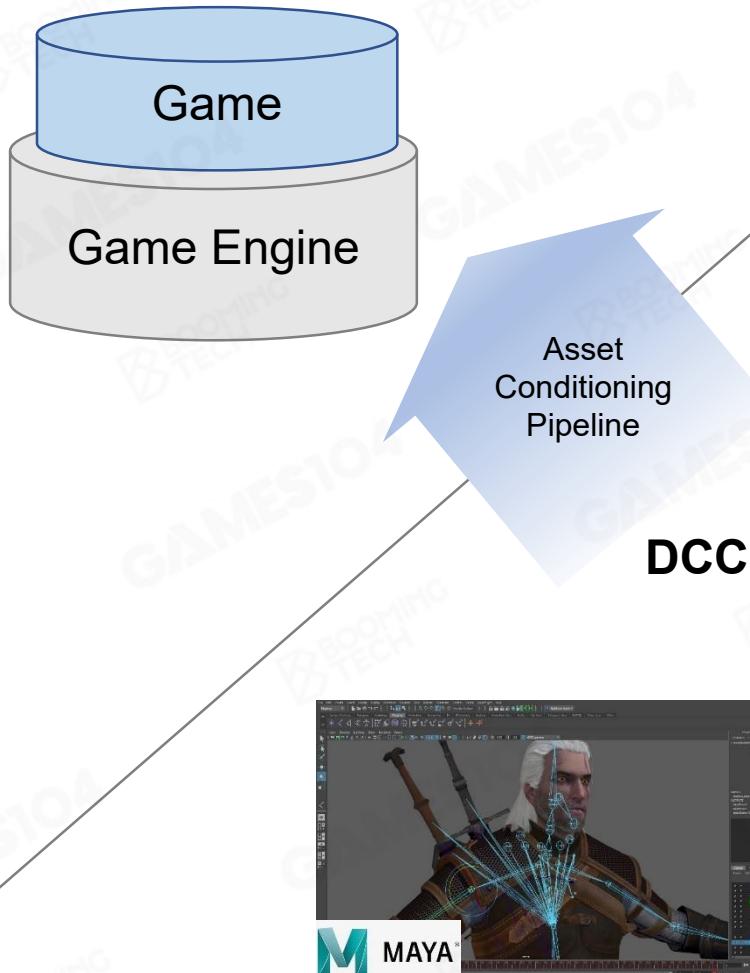
| 3rd Party Libraries | Tool Layer |
	Function Layer
	Resource Layer
	Core Layer
	Platform Layer

Tool - Digital Content Creation

Game

Game Engine

Asset Conditioning Pipeline

Editors

DCC

Houdini

fmod

MAYA

blender

AUTODESK 3DS MAX

3rd Party Libraries

Tool Layer

Function Layer

Resource Layer

Core Layer

Platform Layer
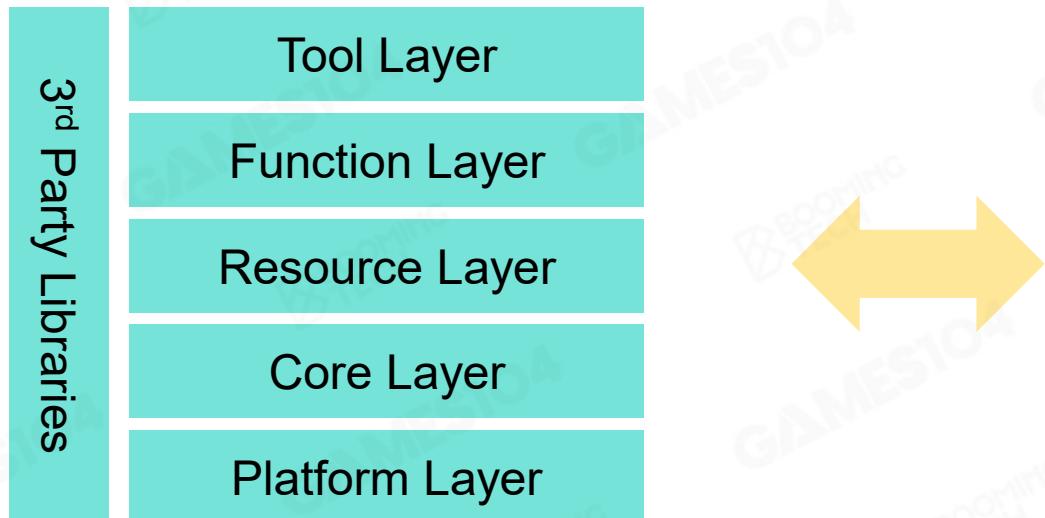
Why Layered Architecture?

Decoupling and Reducing Complexity

- Lower layers are independent from upper layers

- Upper layers don't know how lower layers are implemented

Response for Evolving Demands

- Upper layers evolve fast, but lower layers are stable

3rd Party Libraries

Tool Layer

Function Layer

Resource Layer

Core Layer

Platform Layer

Buildings

Information Systems

Population

Transportation

Energy

# Mini Engine- Pilot

Neat PILOT Engine

Build by C /C++
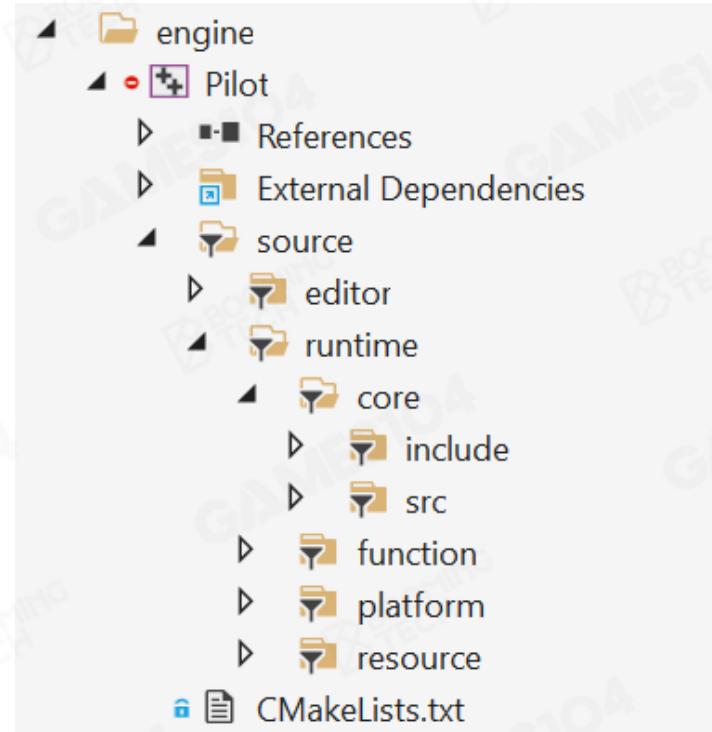
- Runtime: ~13,000 lines

- Editor: ~2,000 lines

Follow Engine Layers

- Source code still improving

Support Platform

- Windows

- Linux

- MacOS (working on M1)

PILOT Editor and Runtime

Basic Editing

- Add/Delete objects
- Move/Scale/Rotate objects

Simple Functions

- Character control
- Camera

Release Plan

1st Release (3/25/2022)
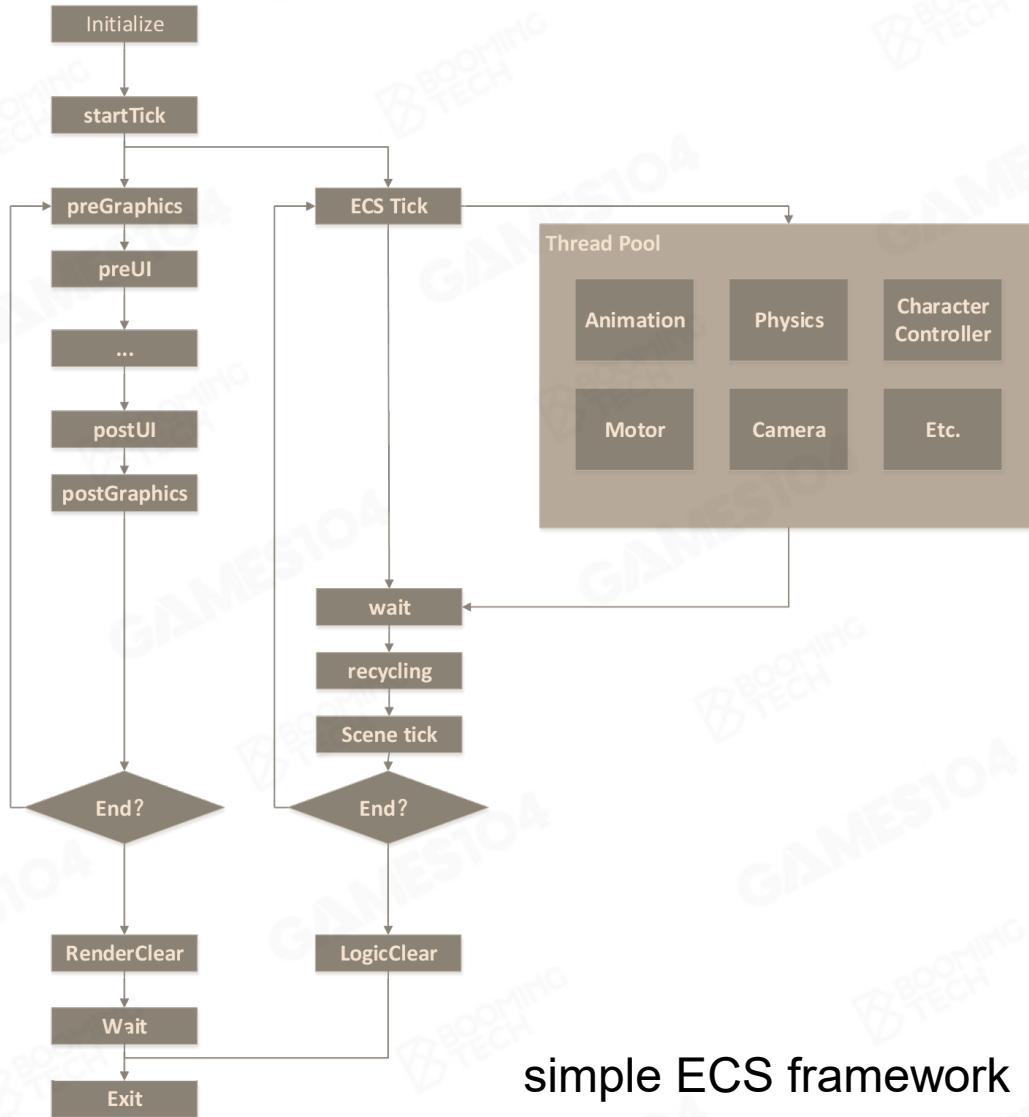
- Editor
- Character/Camera
- Renderer
- Resource system
- Play in editor (PIE)

To be Released with Course

- Animation system
- Collision System
- Gameplay and script systems
- Simple parameter editing
- More graphics features
- …

How to download

- https://github.com/BoomingTech/Pilot

simple ECS framework

Takeaways

- Engine is designed with a layered architecture

- Lower levels are more stable and upper levels are more customizable

- Virtual world is composed by a set of clocks - ticks

Lecture 02 Contributor

- 一将
- Hoya
- 喵小君
- 呆呆兽
- Olorin
- 靓仔

- 爵爷
- Jason
- 砚书
- BOOK
- MANDY
- 俗哥

- 金大壮
- Leon
- 梨叔
- Shine
- 邓导
- Judy

- QIUU
- C佬
- 阿乐
- 阿熊
- CC
- 大喷

PILOT Engine Contributor

- 靓仔
- 达啦崩吧
- Shy

- Adam
- 人工非智能
- KUN

- o果蝇o
- 弥漫
- 陈皮

Enjoy ;)
Coding

Course Wechat

Please follow us for further information

Course Survey

Please scan here to let us know what you think

Please note that all videos and images and other media
are cited from the Internet for demonstration only.