Lecture 02

Layered Architecture of Game Engine

Modern Game Engine - Theory and Practice

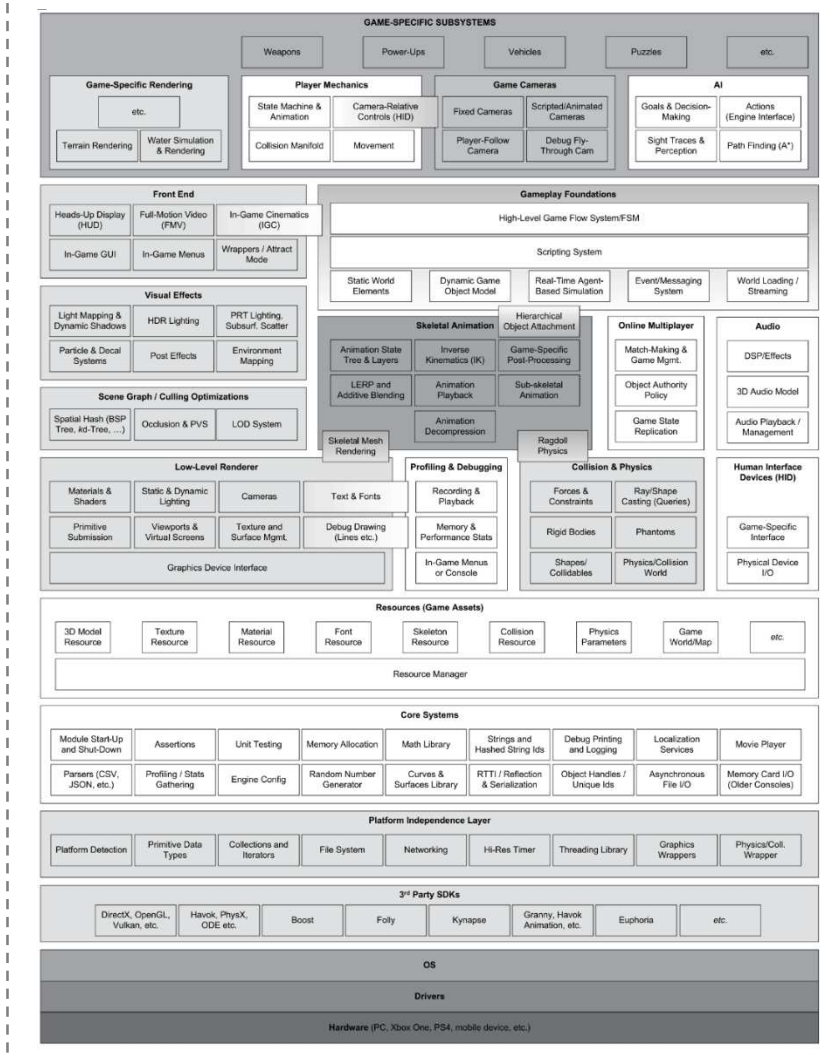
WANG XI

GAMES 104

2022

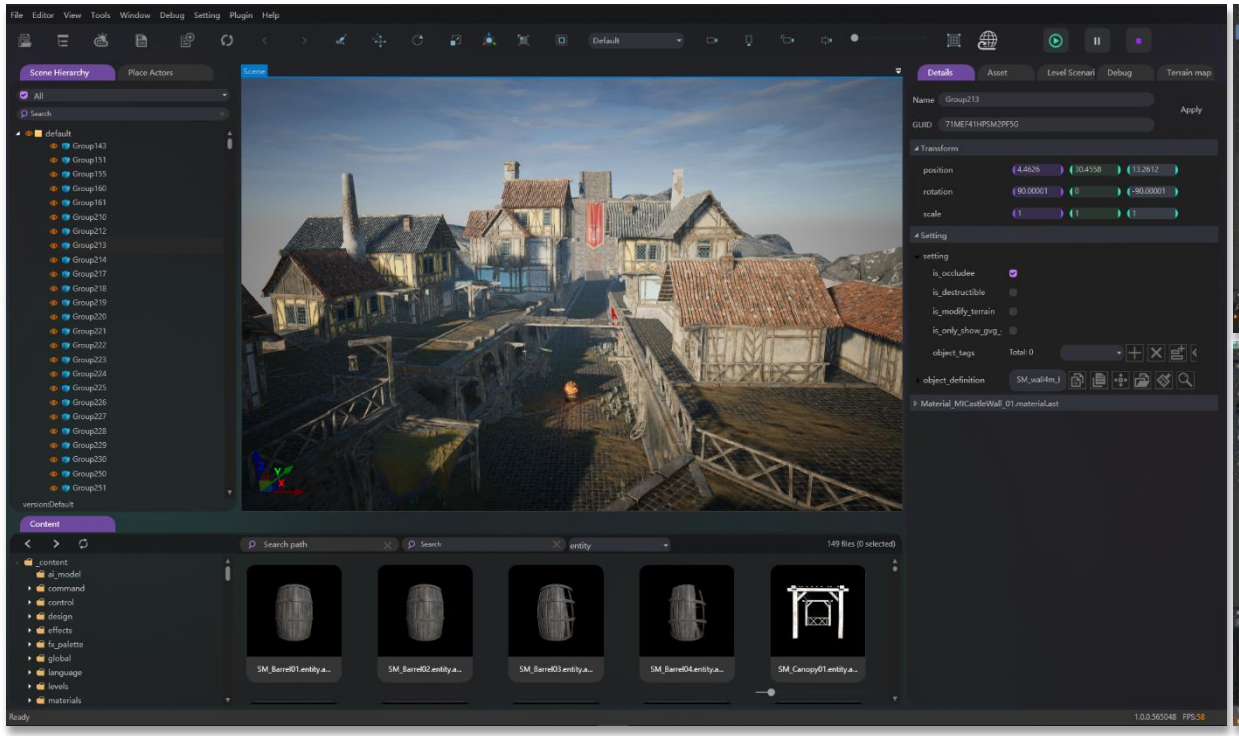
Sea of Codes

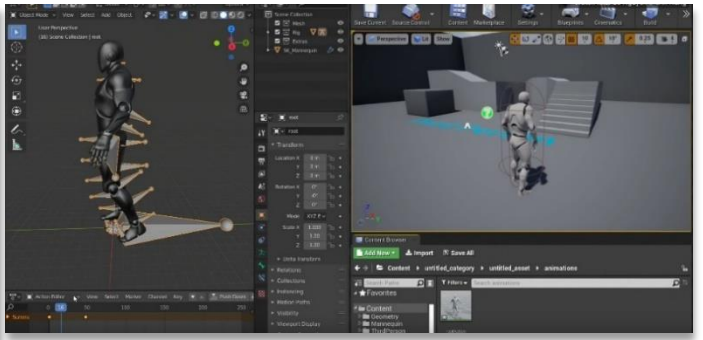

Where to begin?


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, 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 Independence Layer
Platform Detection
Primitive Data Types
Collections and Iterators
File System
Networking
Hi-Res Timer
Threading Library
Graphics Wrappers
Physics/Coll. Wrapper
3rd Party SDKs
DirectX, OpenGL, Vulkan, etc.
Havok, PhysX, ODE etc.
Boost
Folly
Kynapse
Granny, Havok Animation, etc.
Euphoria
etc.
OS
Drivers
Hardware (PC, Xbox One, PS4, mobile device, etc.)

A Glance of Game Engine Layers

Chain of Editors



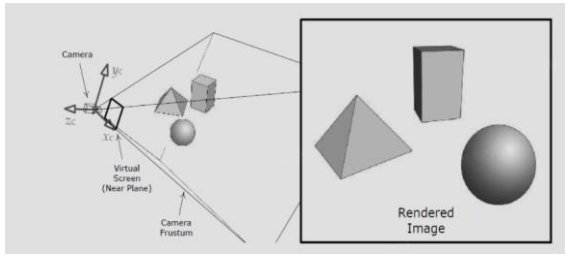


Tool Layer

Make It Visible, Movable and Playable

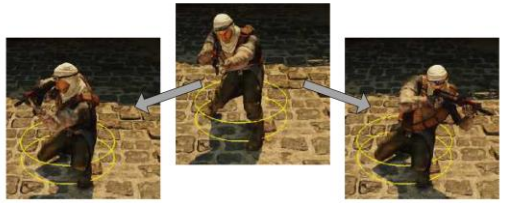
Tool Layer

Function Layer



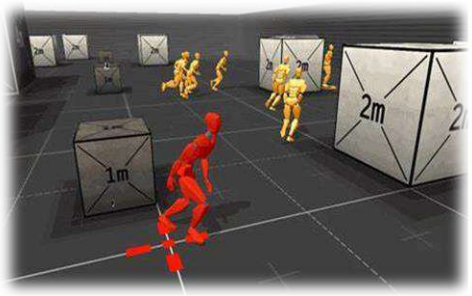
Rendering

Camera, HUD and Input



Animation

Physics



Script, FSM and AI

Data and Files

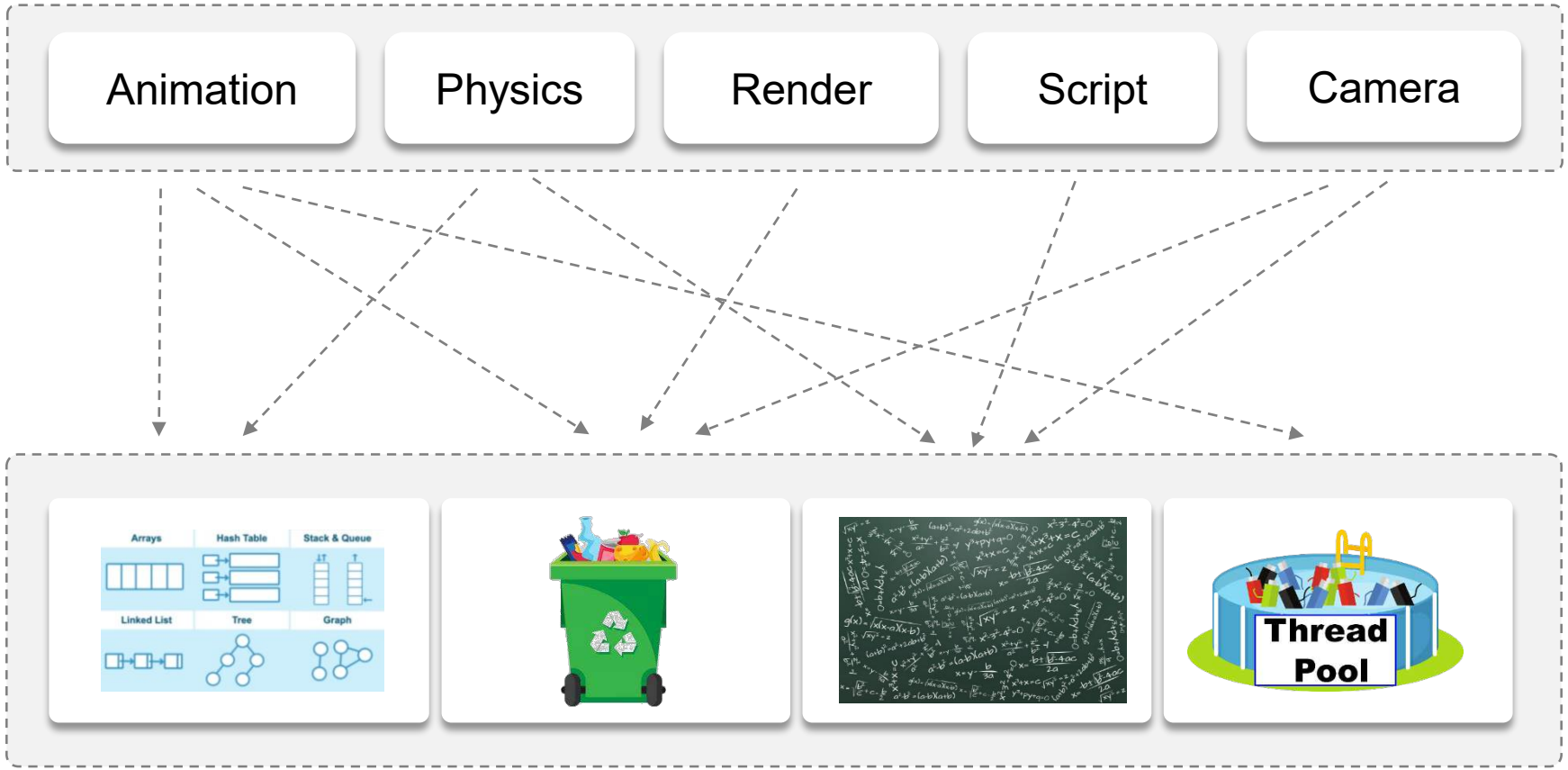
Scene and Level
Script and Graph
Game Logic Data

Tool Layer

Function Layer

Resource Layer

Swiss Knife of Game Engine



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

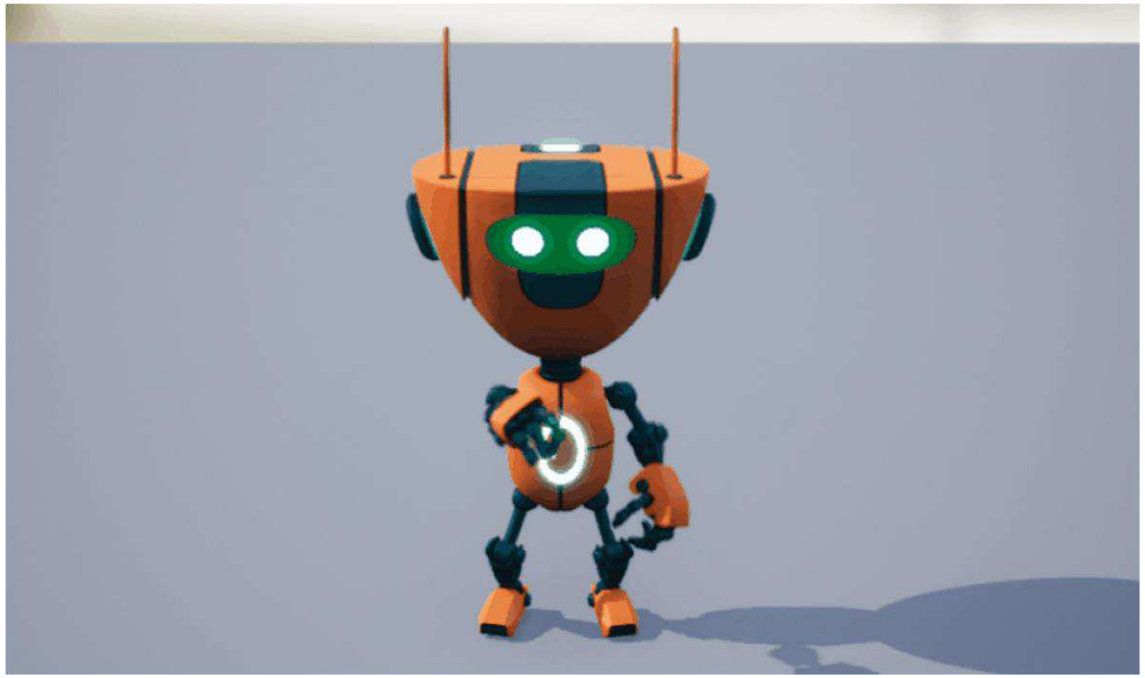
Middleware and 3rd Party Libraries

3rd Party Libraries

- Tool Layer
- Function Layer
- Resource Layer
- Core Layer
- Platform Layer

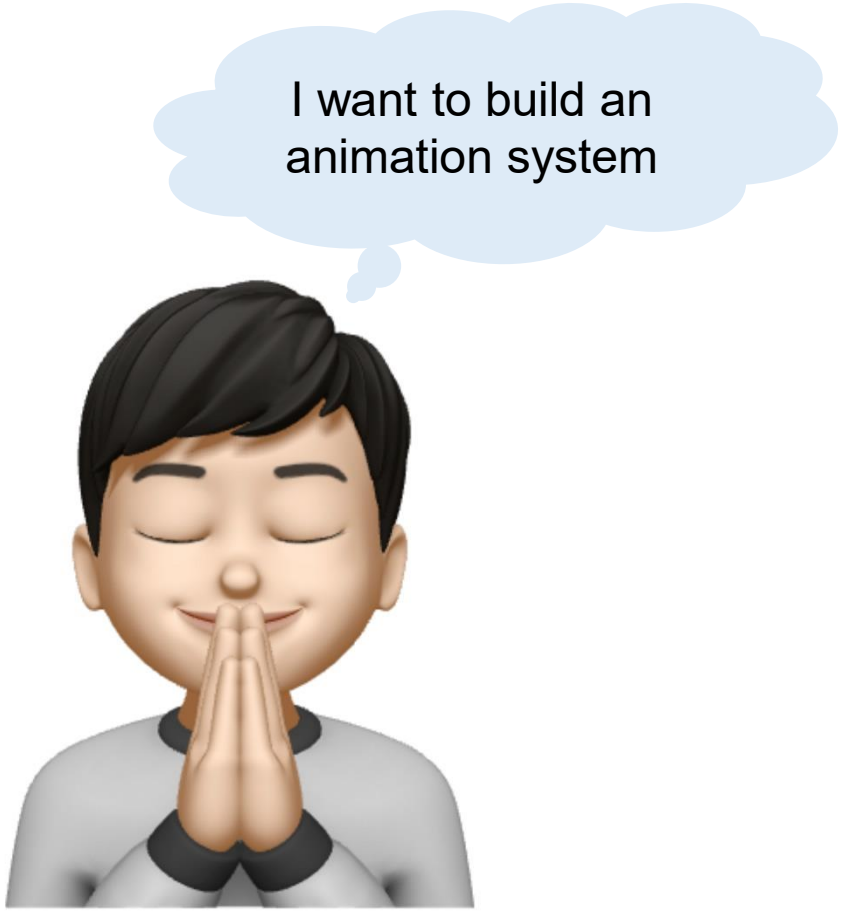
Explore Game Engine Layers

Practice is the Best Way to Learn



Simple Animated Character Challenge

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



Resource - How to Access My Data

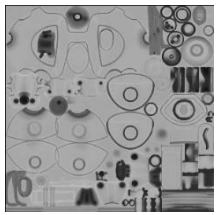
```
<character name="robot" GUID="068db040-f110-4a06-92dc-8ac380705344">
  <geometry>
    <mesh file_path="robot.mesh.ast"/>
    <texture>
      <ableda_texture file_path="robot_ambient.texture.ast"/>
      <roughness_texture file_path="robot_roughness.texture.ast"/>
    </texture>
    <material file_path="robot.material.ast"/>
  </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>
```

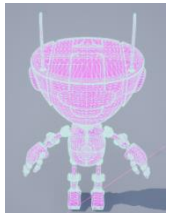
Offline Resource Importing

- 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

GUID


texture.ast


character.ast

mesh.ast animation.ast

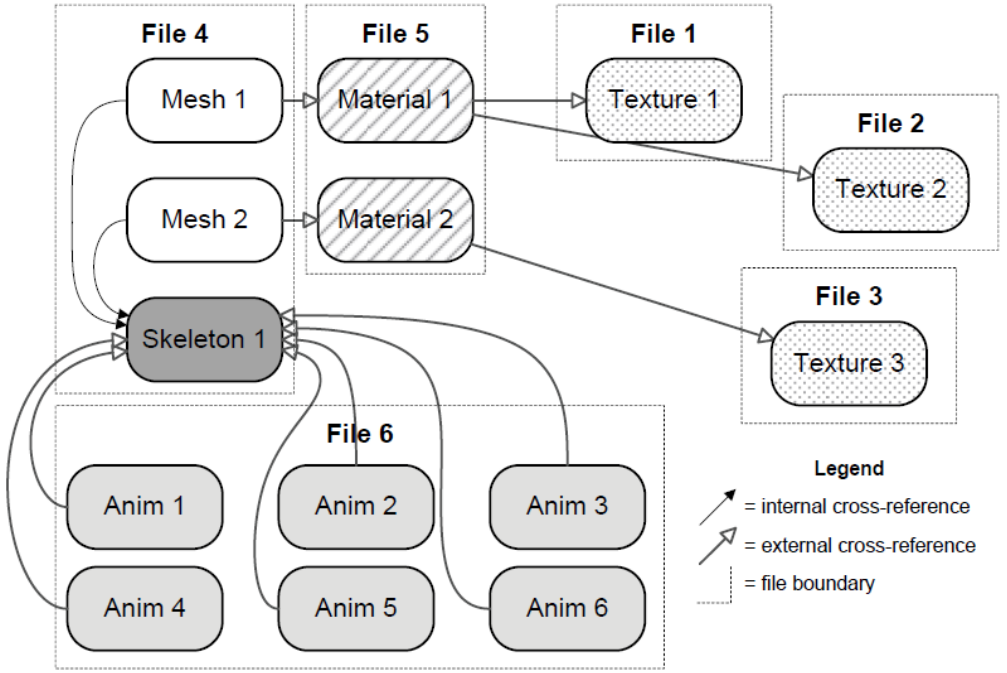
3rd Party Libraries

- 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



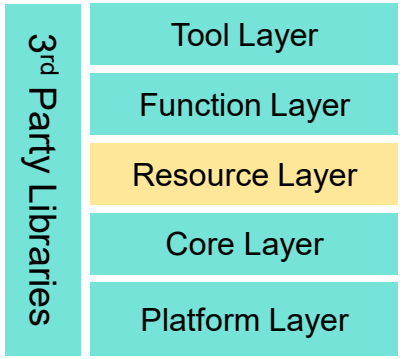
Handle vs. Post Address

Resource – Manage Asset Life Cycle

Character

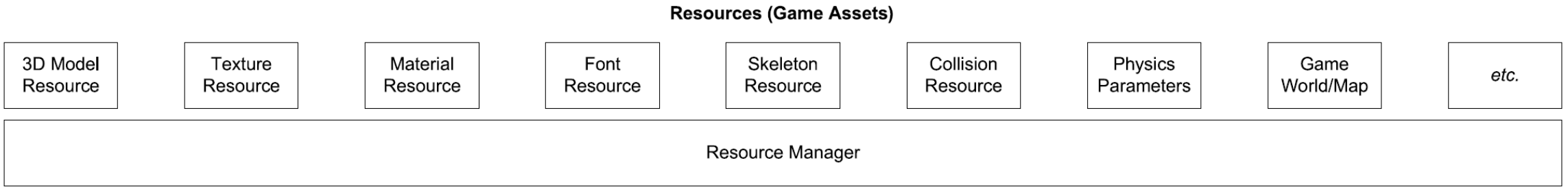
Scene

Cutscene

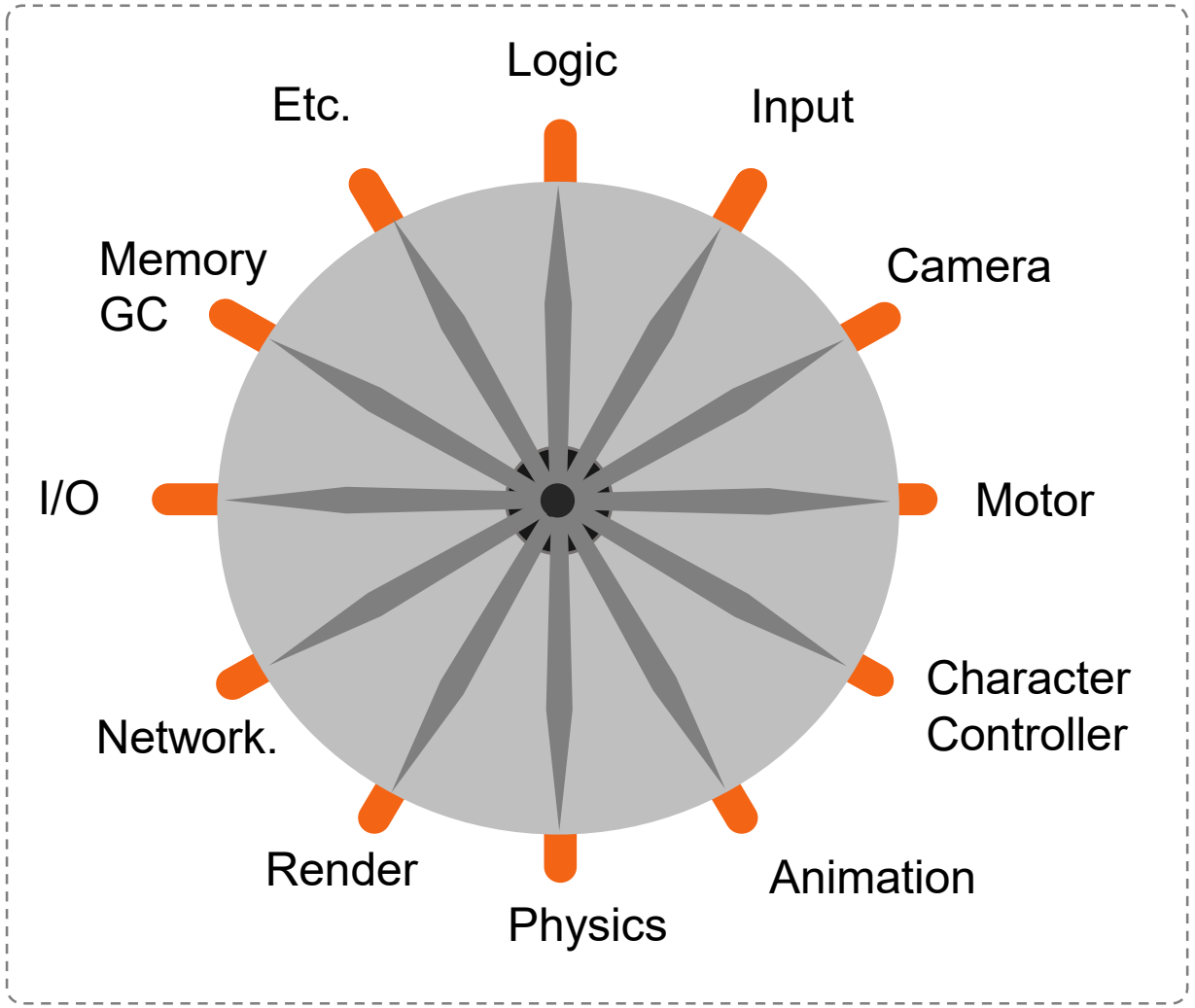


Memory management for Resources - life cycle

- Different resources have different life cycles
- Limited memory requires release of loaded resources when possible
- Garbage collection and deferred loading is critical features



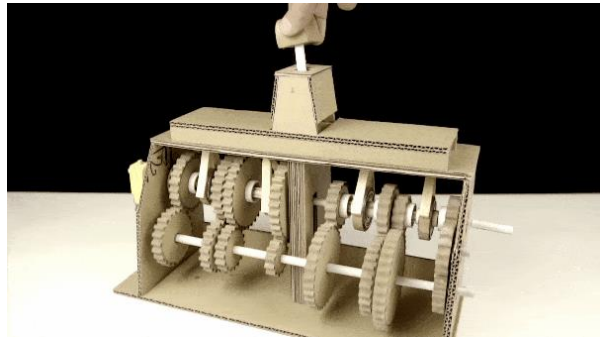
Function - How to Make the World Alive



3rd Party Libraries

- Tool Layer
- Function Layer
- Resource Layer
- Core Layer
- Platform Layer

Unbelievably Simple!!!



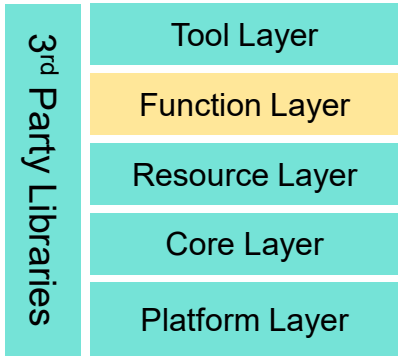
- Transmission shaft game_main.

  cppvoid tick(int delta_time)

- Endless loop

  while (true) { … }

Function - Dive into Ticks




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

        tickRender(delta_time);
    }
}
```

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

    tickMotor(delta_time);

    tickController(delta_time);

    tickAnimation(delta_time);

    tickPhysics(delta_time);

    /*...*/
}
```

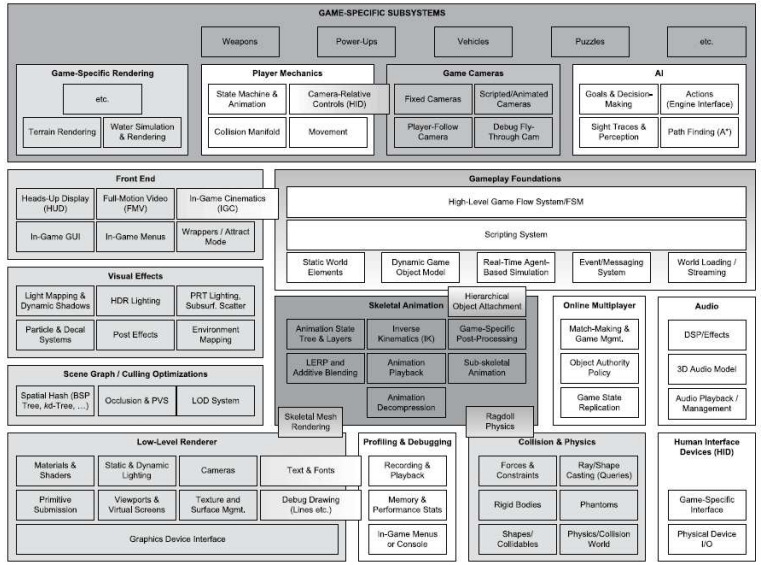
```
void tickRender(float delta_time)
{
    tickRenderCamera();

    culling();

    rendering();

    postprocess();

    present();
}
```

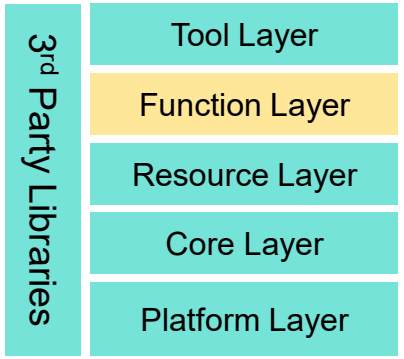
Function -Tick the Animation and Renderer

- 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**

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

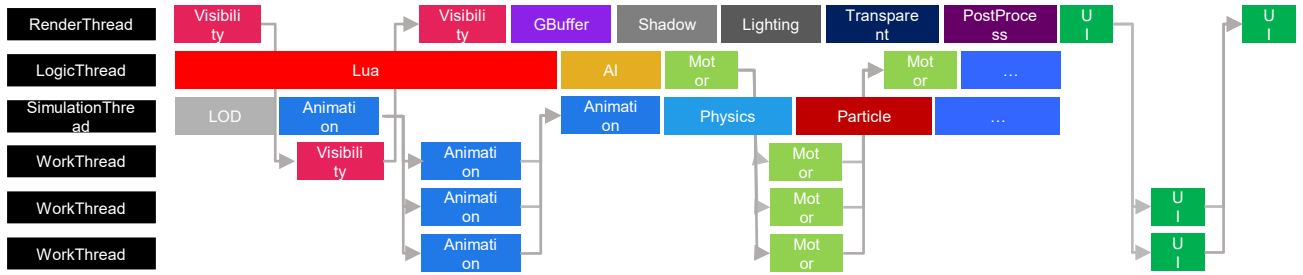


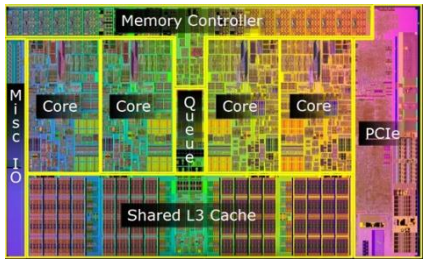


Function - Multi-Threading

Entry
Fixed Thread

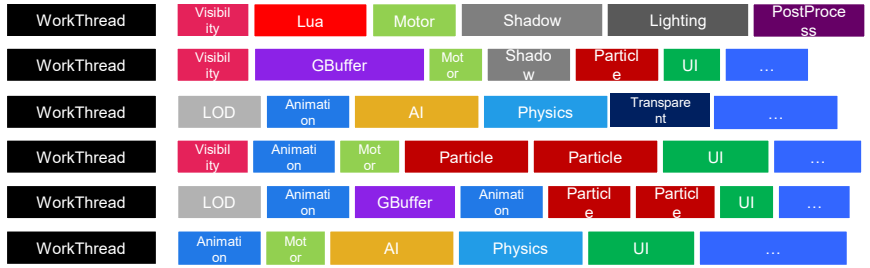
Mainstream
Thread Fork/Join





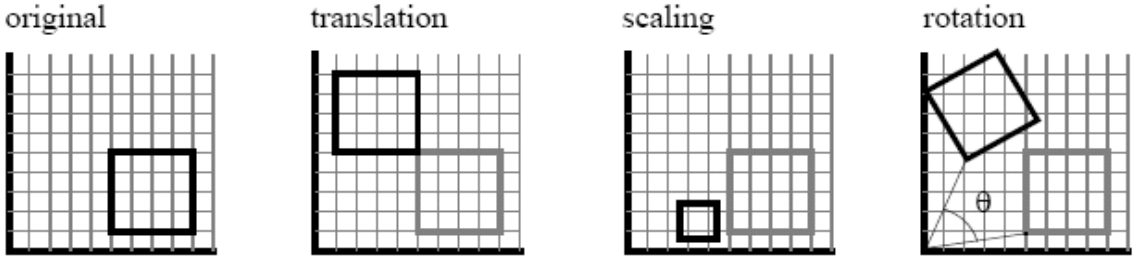
Multi-Core CPU

Advanced
JOB System



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

Core - Math Library

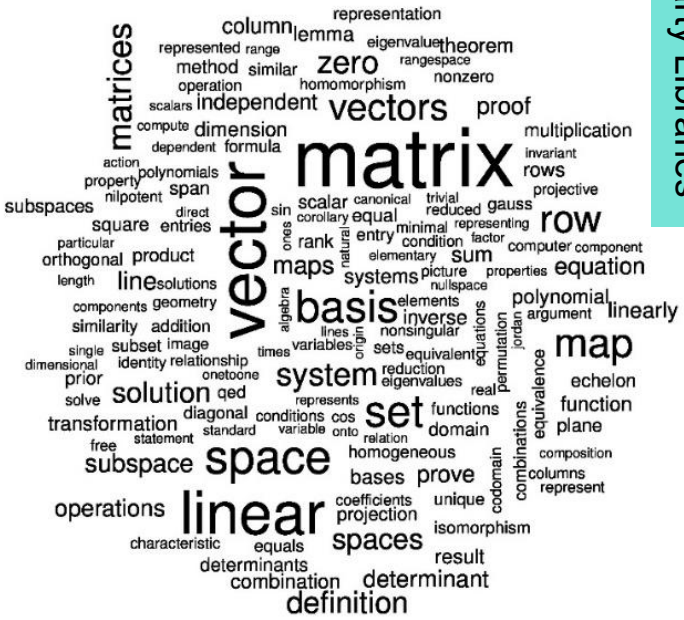


$$T_{\mathbf{v}}\mathbf{p} = \begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x + v_x \\ p_y + v_y \\ p_z + v_z \\ 1 \end{bmatrix} = \mathbf{p} + \mathbf{v}$$

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

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

$$\begin{aligned} x' &= x\cos\theta - y\sin\theta \\ y' &= x\sin\theta + y\cos\theta. \end{aligned}$$



Linear algebra

- Rotation, translation, scaling
- Matrix splines, quaternion

3rd Party Libraries

- Tool Layer
- Function Layer
- Resource Layer
- Core Layer
- Platform Layer

Core - Math Efficiency

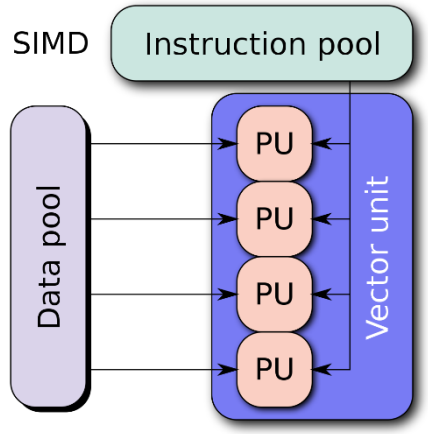
Quick and dirty hacks

- Carmack's 1/sqrt(x)
- Magic number!

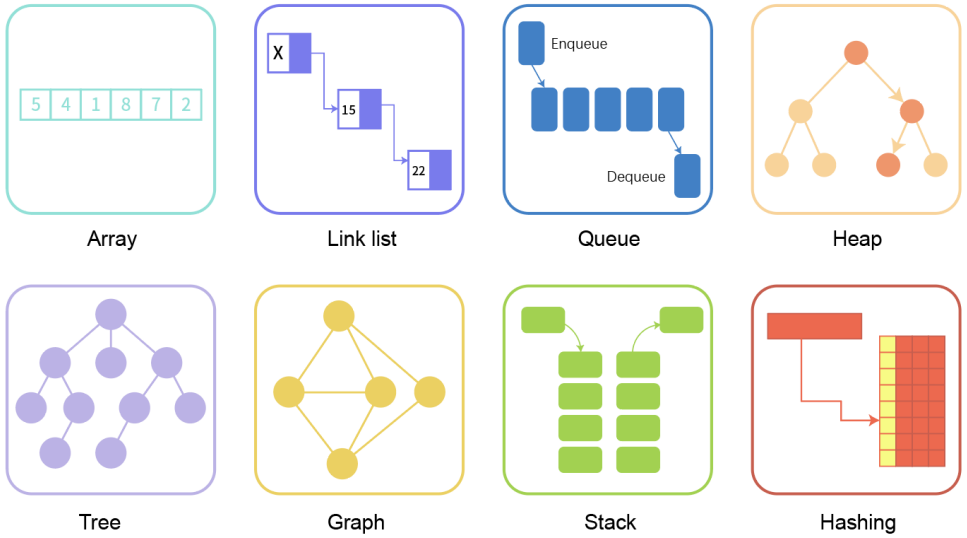
```
float Q_rsqrt(float number)
{
	long i;
	float x2, y;
	const float threehalfs = 1.5F;

	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

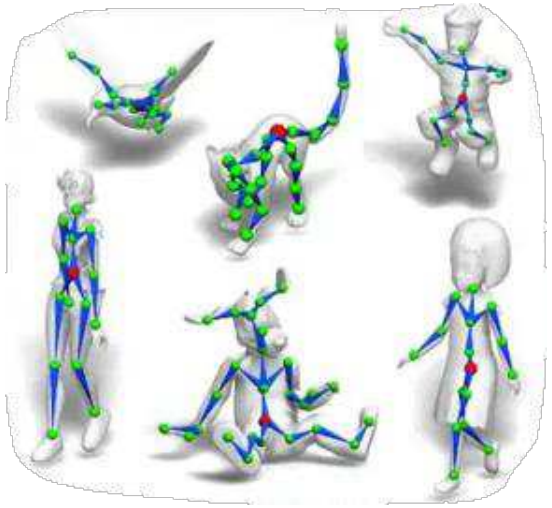
SIMD



Core - Data Structure and Containers



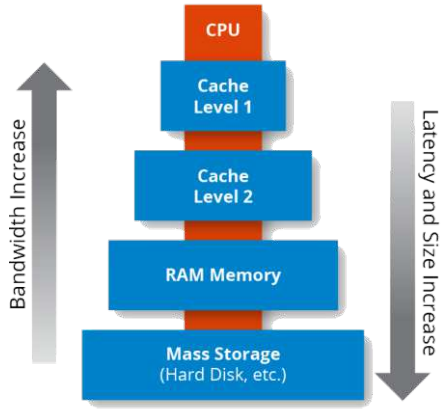
- Vectors, maps, trees, etc.
- Customized outperforms STL
- Avoid memory fragmentation!



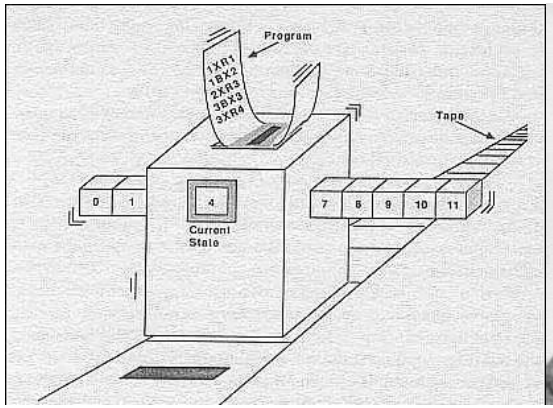
- Skeleton tree
- Animation frame sequence

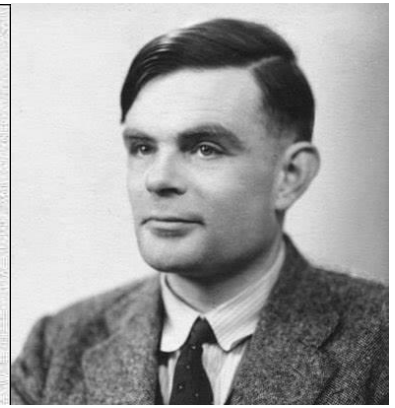
Core - Memory Management

- Major bottlenecks of game engine performance
  - Memory Pool / Allocator
  - Reduce cache miss
  - Memory alignment
- Polymorphic Memory Resource (PMR)

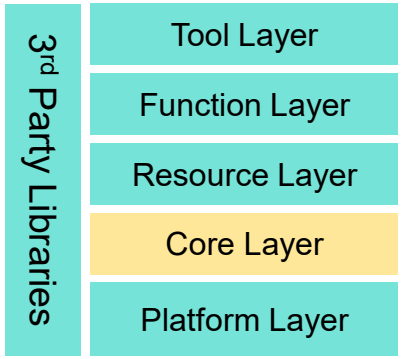


- Cache locality/diffusion
- Memory Arena





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



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)

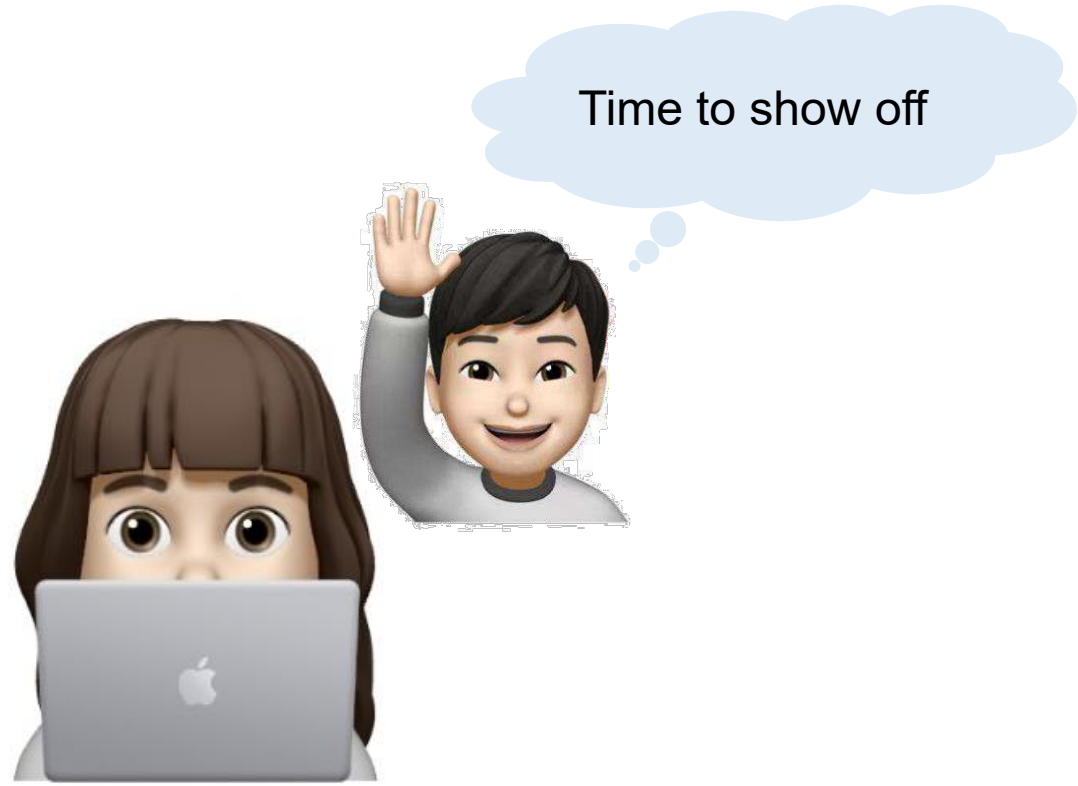
Platform - Target on Different Platform

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

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

S:\Main Folder\Folder1\Folder2\FinalFolder

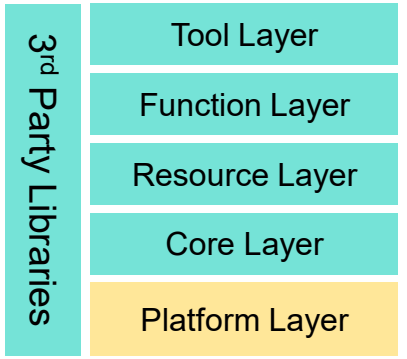
/Volumes/Share/Main Folder/Folder1/Folder2/FinalFolder



Platform - Graphics API

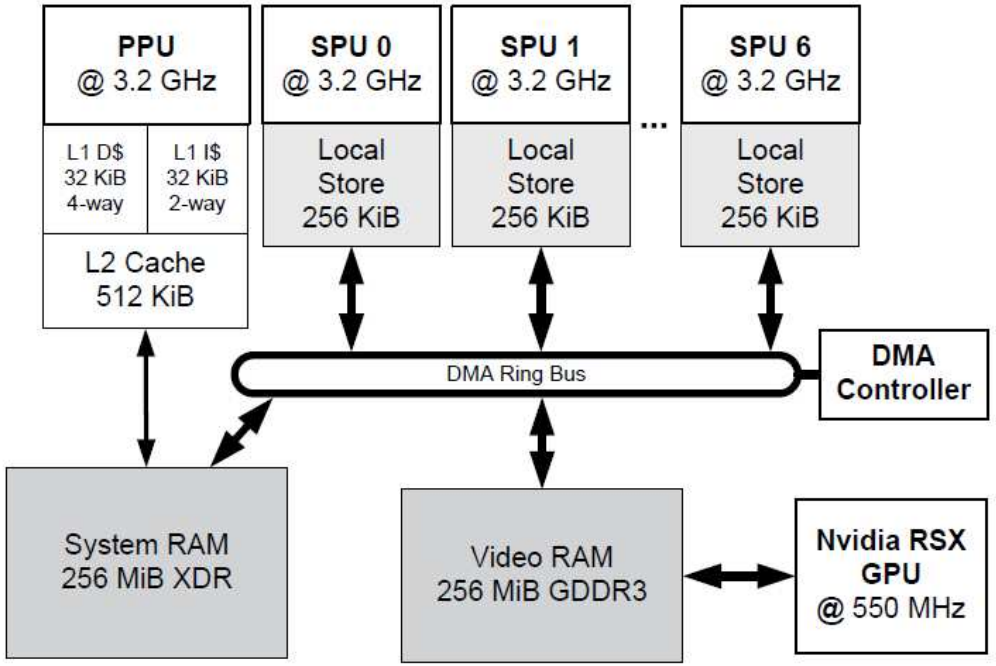
Render Hardware Interface (RHI)

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




```
// 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;
```

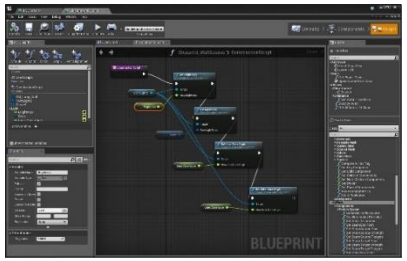
Platform - Hardware Architecture

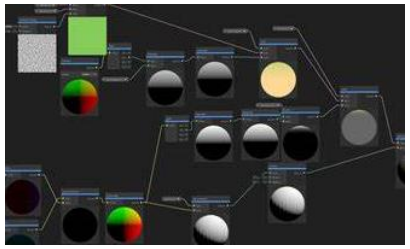


- Core variants: PPU and SPUs
- UMA: unified memory access

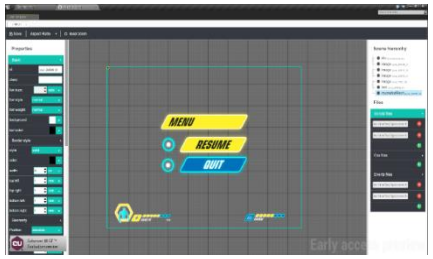
Tool - Allow Anyone to Create Game


Level Editor


Logical Editor


Shader Editor


Animation Editor



UI Editor

Unleash the Creativity

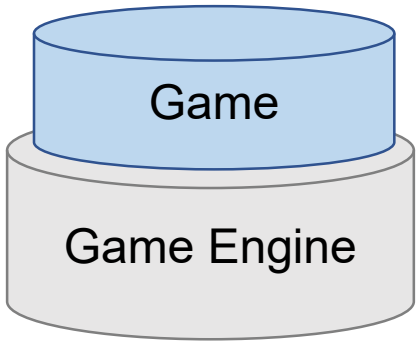
- Build upon game engine
- Create, edit and exchange game play assets

Flexible of coding languages

3rd Party Libraries

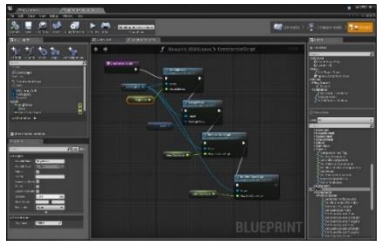
- Tool Layer
- Function Layer
- Resource Layer
- Core Layer
- Platform Layer

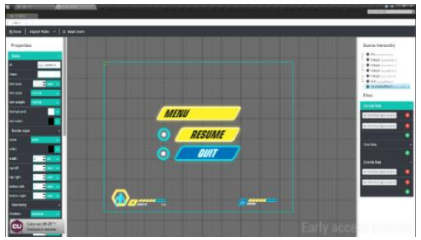
Tool - Digital Content Creation



Asset Conditioning Pipeline

Editors

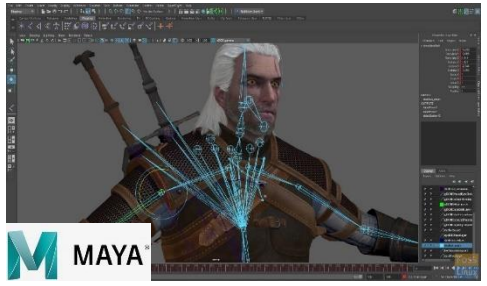





DCC


Houdini


fmod


MAYA


blender


AUTODESK 3DS MAX

3rd Party Libraries

- Tool Layer
- Function Layer
- Resource Layer
- Core Layer
- Platform Layer

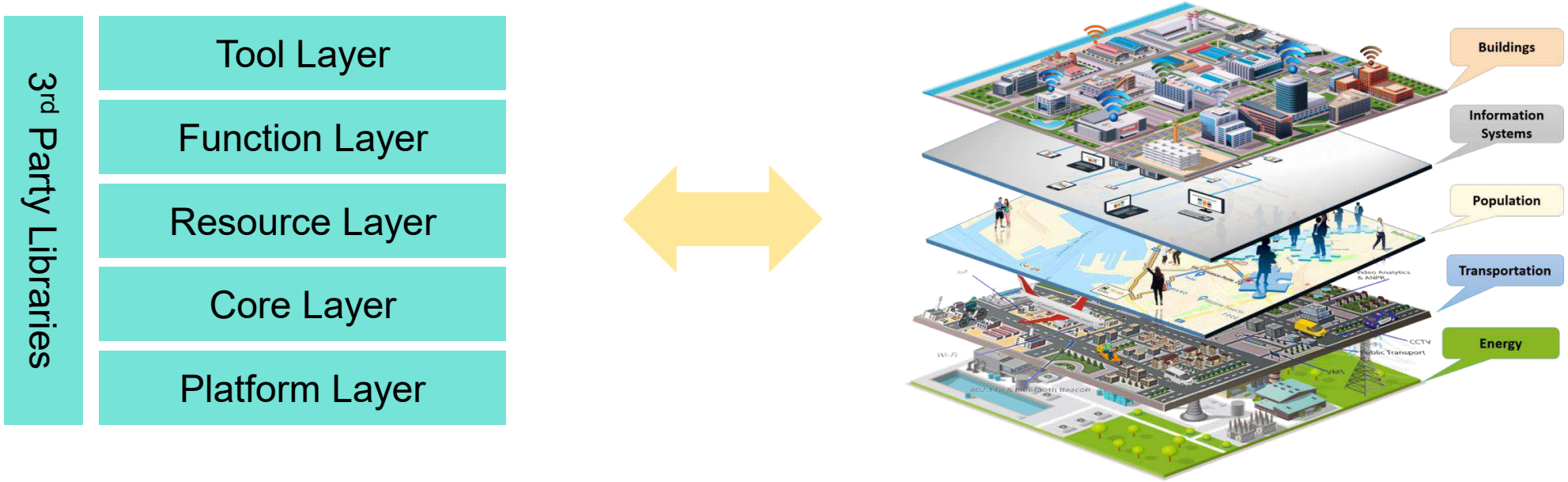
Why Layered Architecture?

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



Mini Engine- Pilot

Neat PILOT Engine

Build by C /C++

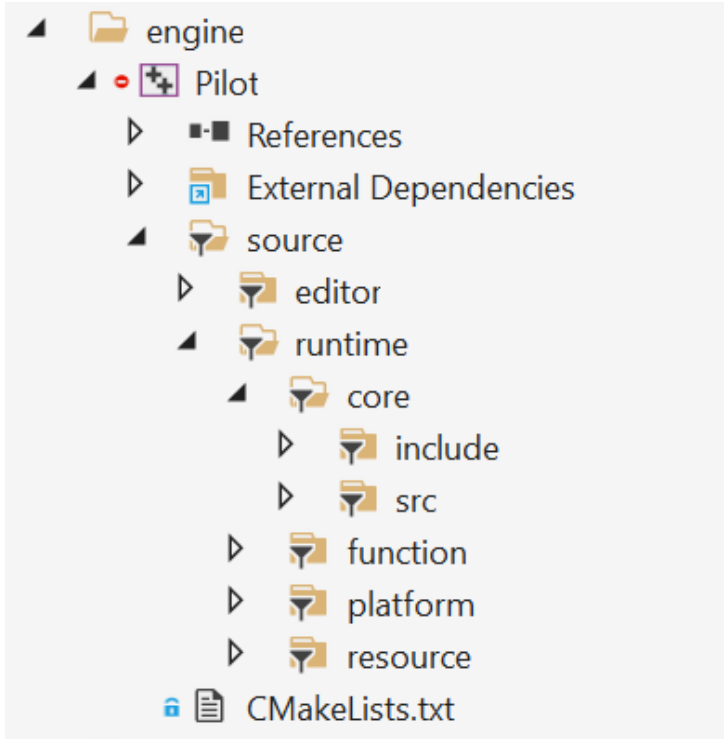
- Runtime: ~13,000 lines
- Editor: ~2,000 lines

Follow Engine Layers

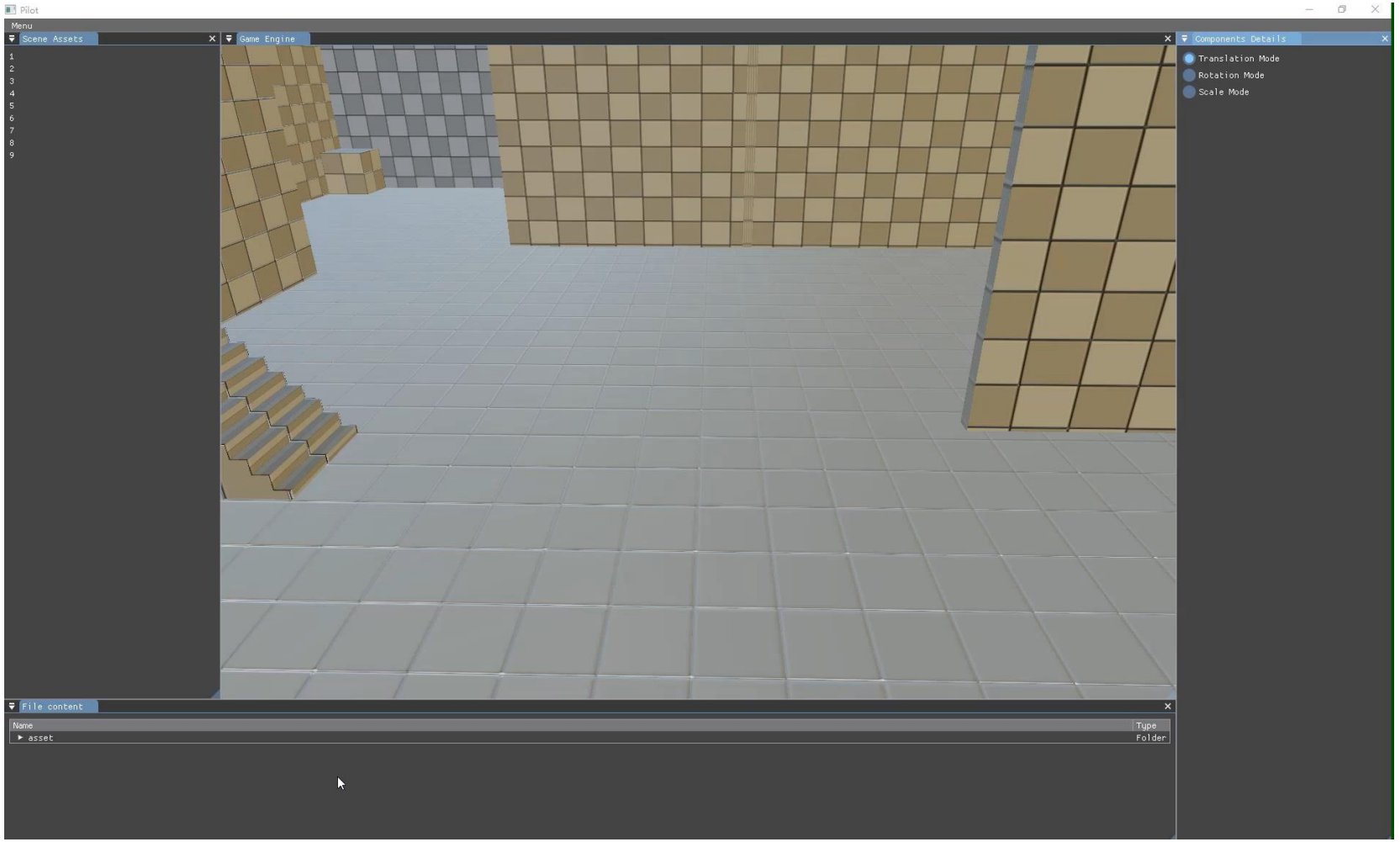
- Source code still improving

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)

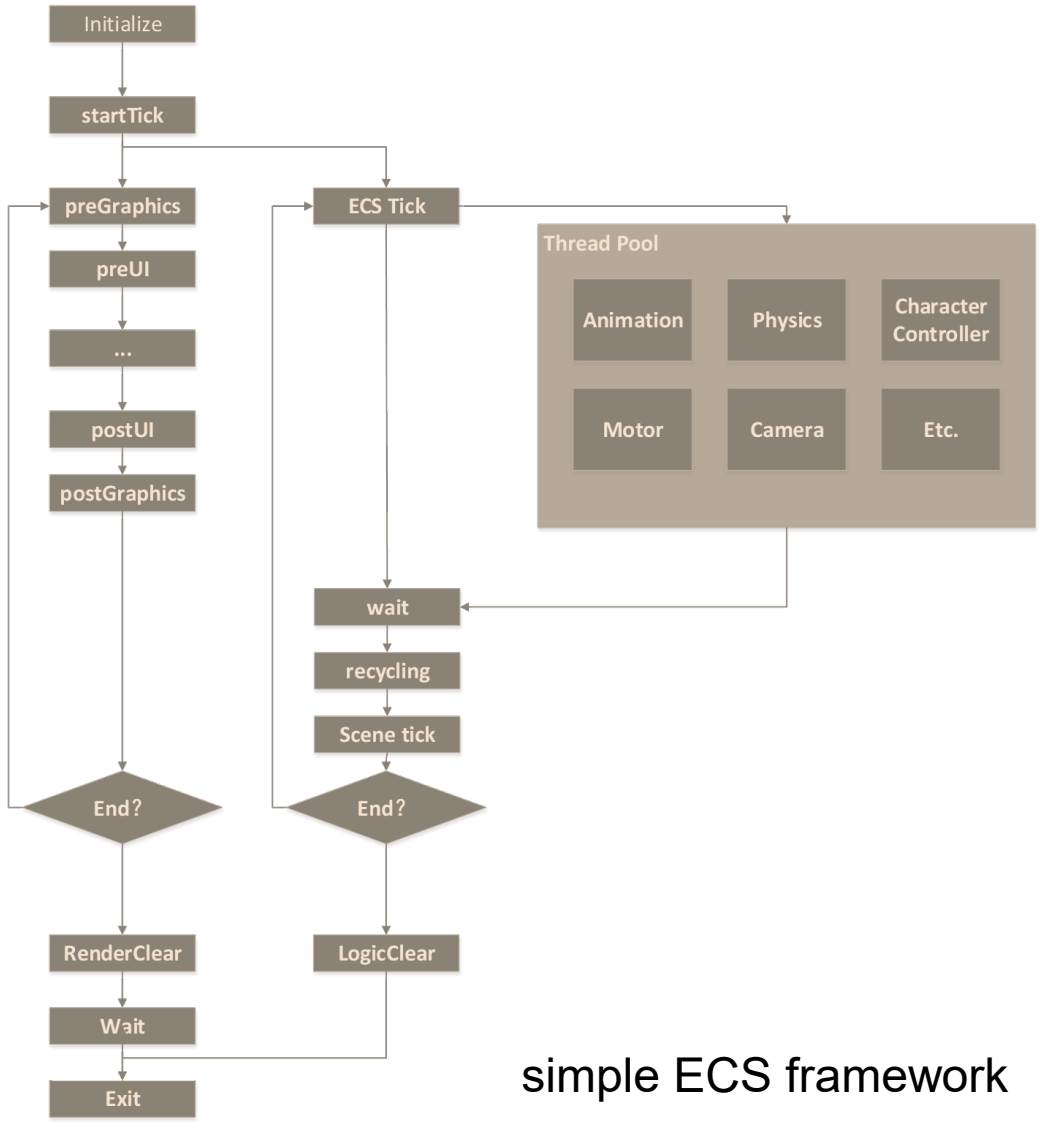
- Editor
- Character/Camera
- Renderer
- Resource system
- Play in editor (PIE)

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.