Wei Dong

+86 15665123201 | isalexpro1615@qq.com | github.com/Away1615

Technical Skills

Languages: C++, Swift, Python, HLSL
Game Engines and Graphics: Unreal Engine 5, DirectX 12, Ray Tracing, Path Tracing, Rasterisation, BVH, SIMD, Multithreading
Engineering and Tools: Git, Visual Studio, Xcode

Education

University of Warwick — Coventry, United Kingdom
MSc Games Engineering — *Sep. 2025 – Jan. 2027*
- WMG Excellence Scholarship, 2025. Relevant coursework: Computer Graphics, Advanced Computer Graphics, Games Engine Design and Development, Games Engineering.

Wuhan University of Technology — Wuhan, China
BEng Software Engineering — *Sep. 2017 – Jun. 2021*
- Relevant coursework: Data Structures, Computer Networks, Operating Systems, Database Systems, Computer Organization.

Work Experience

ByteDance — Beijing / Shanghai, China
iOS Client Engineer — *Jul. 2021 – Jul. 2024*
- Developed and maintained Feishu and Doubao iOS features, translating product requirements and design drafts into practical client-side planning and implementation.
- Contributed to suite-wide Feishu client infrastructure, including dark mode, iPad multi-scene support, typography, sharing, push modules and reusable UI components.
- Collaborated across product, platform and business teams on Feed modularisation, interface abstraction, Rust-integrated client work and test framework improvements.

Projects

Rift | *C++, Unreal Engine 5, LAN Listen Server, Gameplay Ability System, AI* — 2026
- Engineered a 1-4 player third-person LAN co-op PvE ARPG prototype with room-code session flow, replicated lobby readiness, character appearance confirmation, map travel, teleport objective, boss fight, respawn and victory flow.
- Implemented GAS-based twin-sword combat including combo graph attacks, guard, dodge, RapidSlash, replicated health/resource attributes, poise, stagger, hit reactions and combat feedback.
- Built host-authoritative enemy encounters and teleport waves using spawn budgets, weighted enemy pools, spawn limits and configured spawn points.
- Implemented Behavior Tree enemy AI with server-side target selection, melee attack variants, shield block behavior, hit windows and replicated combat feedback.
- Implemented a replicated boss phase system with health-threshold phase transition, multicast phase VFX and GameMode-driven victory broadcast.

DX12-FPS | *C++, DirectX 12, HLSL* — 2026
- Built a first-person shooting demo on a custom DirectX 12 renderer with swapchain, frame resources, command lists, render/depth targets, root signatures, PSOs and shader management.
- Designed a data-driven level loading flow that creates textures, shaders, PSOs, materials, game objects and components from configuration files.
- Implemented a component-based gameplay framework with first-person camera, player input, weapon control, collision and rendering components.
- Implemented WASD movement, mouse look, AABB collision response, raycast shooting, weapon fire/reload animation states and NPC hit/death feedback.
- Supported static meshes, skeletal animated meshes, sky sphere rendering, normal maps, point lights and GPU instancing for a 3D scene with vegetation, characters and first-person weapons.

UE5 Vehicle Combat Team Project | *Unreal Engine 5, Blueprint/C++, Niagara* — 2026
- Collaborated in a 6-person team on a UE5 vehicle combat game with two playable forest and city levels; responsible for dynamic weather and gameplay VFX.
- Implemented a Blueprint-based Weather Manager with sunny, overcast and rainy presets, supporting smooth weather transitions triggered by level events.
- Used Niagara and materials to create rain, ground splash, lightning, speed lines, missiles and tire marks for vehicle, weapon and environment feedback.
- Implemented lightweight C++ particle components exposed to Blueprint for vehicle smoke and collision sparks, dynamically adjusting effects based on vehicle state and collision strength.

Optimised Software Rasterizer | *C++, AVX SIMD, Multithreading* — 2026
- Optimised a CPU software rasterizer with vertex caching, pre-normalised lighting, early depth testing, back-face/frustum culling and edge-function rasterisation.
- Used AVX SIMD to vectorise 8-pixel parallel shading and lighting calculations, reducing per-pixel scalar computation and improving pixel throughput.

- Implemented tile-based multithreaded rendering, splitting the screen into independent tasks and scheduling them through a custom thread pool to reduce single-thread bottlenecks.
- Compared optimisation results using FPS, P99 latency and speedup; SIMD combined with pipeline optimisations achieved approximately 2 to 4 x speedup over the baseline.