目录

Data Oriented Tech Stack Learning

Lei Xinyue

DOTS: data oriented tech stack.

We are aware that we are proposing a rather large change in how to write code. From MonoBehaviour.Update to ComponentSystem & using jobs.

Unity引擎将用C#重写。Previously most of our engine code was written in C++, which creates a disconnect with how our customers write code and how programmers at Unity write code. Due to the Burst compiler tech & ECS, we can achieve better than C++ with C# code and as a result we can all write code exactly the same way.

1 Useful Links

1) DOTS official website
2) Detail documentation of DOTS
3) 一图认识ECS
4) Unity Data Oriented Tech Stack principles and vision Unity Data-Oriented Tech Stack Cheat Sheet
5) SIMD
6) Unity Data-Oriented learning resources

2 DOTS Overview

https://unity.com/dots

Rebuilding Unity's core

We' re rebuilding the core foundation of Unity with the high-performance multithreaded Data-Oriented Technology Stack. DOTS makes it possible for your game to fully utilize the latest multicore processors without the heavy programming headache. DOTS includes the following features:

- The C# Job System for running multithreaded code efficiently.
- The Entity Component System (ECS) for writing high-performance code by default.
- The Burst Compiler for producing highly optimized native code.

2.1 ECS

The Entity Component System (ECS) is the core of the Unity Data-Oriented Tech Stack. As the name indicates, ECS has three principal parts:

Entities — the entities, or things, that populate your game or program

Components — the data associated with your entities, but organized by the data itself rather than by entity. (This difference in organization is one of the key differences between an object-oriented and a data-oriented design.)

Systems

— the logic that transforms the component data from its current state to its next state— for example, a system might update the positions of all moving entities by their velocity times the time interval since the previous frame.

2.2 Job System

The Unity C# Job System lets you write simple and safe multithreaded code that interacts with the Unity Engine for enhanced game performance.

You can use the C# Job System with the Unity Entity Component System (ECS), which is an architecture that makes it simple to create efficient machine code for all platforms.

2.3 Burst Compiler

Burst is a new LLVM based backend compiler technology that makes things easier for you. It takes C# jobs and produces highly-optimized machine code taking advantage of the particular capabilities of your platform. So you get a lot of the benefits of hand tuned assembler code, across multiple platforms, without all the hard work. The Burst compiler can be used to increase performance of jobs written for the C# Job System.

How to optimize for the Burst compiler

2.4 ECS principles

DOTS基于的一些原则(一些实现了，一些只是期望)

2.4.1 Performance by default

使在所有平台上创建高效的机器代码(efficient machine code)简单化。(We want to make it simple to create efficient machine code for all platforms.)

ECS (Entity Component System) + C# Job System + Burst compiler

1. The Entity Component System guarantees linear data layout when iterating entities in chunks by default.
2. The C# job system lets you write multithreaded code in a simple way. It is also safe. The C# Job Debugger detects any race conditions.
3. Burst is our compiler specifically for C# jobs. C# job code follows certain patterns that we can use to produce more efficient machine code. Code is compiled & optimized for each target platforms taking advantage of SIMD instructions.

2.4.2 Simple

Writing performant code must be simple. We believe that we can make writing fast code as simple as MonoBehaviour.Update.

2.4.3 One way of writing code

We want to define a single way of writing game code, editor code, asset pipeline code, engine code. We believe this creates a simpler tool for our users, and more ability to change things around. If physics engine code was written the same way as game code using ECS, it would make it easy to plug your own simulation code between existing physics simulation stages or take full control.

2.4.4 Networking

It is work in progress.

2.4.5 Determinism

Our build pipeline must be deterministic. Users can choose if all simulation code should run deterministically. You should always get the same results with the same inputs, no matter what device is being used. This is important for networking, replay features and even advanced debugging tools.
(浮点数问题)
To do this, we will leverage our Burst compiler to produce exact floating point math between different platforms.

2.4.6 Sandbox

Unity is a sandbox, safe and simple. A good example of sandbox behaviour is that our C# job system guarantees that none of your C# job code has race conditions.

2.4.7 Tiny

We want Unity to be usable for all content from ¡ 50kb executables + content, to gigabyte sized games. We want Unity to load in less than 1 second for small content.

2.4.8 Iteration time

We aim to keep iteration time for any common operations in a large project folder below 500ms.