Análise dos ganhos de performance na utilização do padrão Entity Component Systems contra implementações orientadas a objetos

Yuri Moraes Gavilan[1]
Bianca de Almeida Dantas[1]

[1]Faculdade de Computação – Universidade Federal do Mato Grosso do Sul (UFMS)

yuri.moraes.gavilan@ufms.com, bianca.dantas@ufms.br

Abstract. *This paper describes the problem of complex data configurations in game development – solved distinctively by the ECS and OO patterns – showing the difference in performance between them. Here it will be explored the premise of ECS and the experiments made to measure the performance gain compared to tradicional object oriented implementations.*

Resumo. *Este artigo descreve o problema de arranjo de dados complexos de entidades no desenvolvimento de jogos – solucionado de forma distinta pelos padrões ECS e OO – demonstrando a diferença de performance entre eles. Nele será explorada a premissa do ECS e os experimentos realizados para medir o ganho de desempenho comparado a implementações tradicionalmente orientadas a objetos.*

1. Introdução

No desenvolvimento de jogos eletrônicos, um problema recorrente é o da estruturação dos objetos de jogo que serão operados no processo de simulação do mundo virtual. Uma proposta amplamente aceita é a de objetos definidos por componentes [Nystrom 2011]: representações reutilizáveis de dados que definem comportamentos específicos. A partir disto, surgiu o padrão *Entity Component Systems* (ECS), que tem como objetivo aliar a flexibilidade das entidades descritas por componentes, com o desempenho computacional da programação Orientada a Dados [Fabian 2018], que leva em conta os padrões de acesso de memórias *cache* do processador no projeto das estruturas de dados a serem utilizadas.

Para entender os possíveis ganhos de desempenho a partir da utilização do padrão ECS e sua proposta de melhor utilização das memórias de baixa latência, foram construídas duas simulações semelhantes: uma baseada em ECS e outra no padrão orientado a objetos. O objetivo é analisar as métricas de *frametime* (tempo de quadro) em situações com altas quantidades de entidades.

2. Premissa

A criação dos ECS parte da ideia de garantir agrupamento eficiente de dados na memória de acordo com o seu uso a fim de minimizar erros de *cache* por parte do processador. Os conceitos principais são três: **entidades** que são apenas índices dos vetores de componentes; **componentes** que são estruturas de dados sem comportamento (métodos); **sistemas** que são funções simples que operam sobre conjuntos homogêneos de componentes.

Dessa forma, caso seja necessário referenciar a um componente de uma entidade de índice 1, basta olhar a posição 1 do respectivo vetor.

Idealmente, os componentes são armazenados em vetores lineares e contíguos. Contudo, em situações reais isso não é possível ser garantido, afinal nem todas entidades possuirão todos componentes e isso acaba gerando lacunas nos vetores de dados. Para solucionar este problema, foi criado o conceito de **arquétipos**.

Arquétipos são as combinações possíveis de componentes existentes em uma simulação. Em uma situação imaginária existem dois componentes, chamados "A" e "B". Neste mundo, três entidades estão vivas, $E0$ e $E2$ abrigam o conjunto $[A, B]$ enquanto $E1$ apenas $[A]$. Dessa forma, $[A, B]$ e $[A]$ são diferentes arquétipos e serão armazenados em locais distintos na memória do computador. Isso permite que não existam lacunas nos vetores de componentes, visto que é obrigatória a posse de uma entidade sobre os mesmos, existentes em certo arquétipo [Unity-Technologies 2024a].

3. Resultados experimentais

Para entender melhor o comportamento, foram criadas duas simulações no motor de jogo *Unity* [Unity-Technologies 2024b] que implementam os mesmos algoritmos e regras de mundo. Entretanto, ambas foram criadas com paradigmas diferentes, sendo uma tipicamente orientada a objetos, e outra baseada em ECS. Estas simulações foram executadas diversas vezes com o número crescente de entidades simuladas, com a finalidade de entender o comportamento computacional em situações com quantidades altíssimas de dados a serem processados. A unidade de medida utilizada nos testes é o *frametime*, que é o tempo (em milissegundos) de duração de cada passo de simulação, ou seja, menores *frametimes* indicam melhor desempenho. É importante dizer que *frametimes* acima de $33{,}3$ milissegundos não são considerados aceitáveis para jogos comercializáveis, por não oferecerem fluidez de simulação para os jogadores, logo, este é o padrão a ser atingido.

O motor Unity foi escolhido para a construção destas simulações por oferecer recursos oficiais de ECS baseado em arquétipos nas ferramentas integradas do motor, sem a necessidade de integração de bibliotecas de terceiros, visto que soluções externas podem ramificar arquiteturalmente e em suas bases de arranjos de dados.

Nesta simulação, existem três tipos de objetos de jogo a serem processados: cubos, que aumentam linearmente sua escala até um certo limiar superior aleatório no intervalo de $[10; 12]$ e, então, diminuem até um limiar inferior aleatório no intervalo $[0{,}1; 1]$; esferas, que circulam um certo ponto no espaço determinado aleatoriamente no início da simulação; e cilindros que têm o comportamento de ambos os tipos anteriores (são escaláveis e rotacionáveis). A escolha das formas geométricas não influenciam de modo relevante no desempenho por serem triviais para qualquer GPU moderna renderizar, portanto, são diferentes unicamente para facilitar a visualização por parte do usuário.

Para a implementação ECS dessa simulação, foram definidos dois tipos de componentes, um chamado *ScalingComponent* que define o comportamento de escala de um objeto, e outro *RotatingComponent* que compõe o comportamento de rotação em torno de um ponto. Para o processamento, existem dois sistemas: *ScalingSystem* e *RotatingSystem*, que fazem a consulta aos vetores de componentes e realizam o processamento sobre os dados, sem a necessidade de consultar a entidade proprietária de cada um deles. Dessa forma, a tendência é a computação ser feita de forma simples e direta.

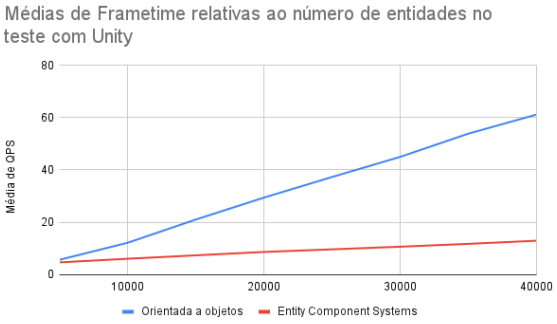

Figura 1. Diferenças de *frametime* utilizando *Unity OO* e *Unity ECS*.

Nesta situação, um ganho de flexibilidade arquitetural é a facilidade de criar objetos baseados em componentes. Para criar os cilindros, não é necessário duplicação de código, apenas atribuí-lo aos componentes de rotação e escala e os sistemas automaticamente já os incluirão aos processamentos a serem feitos.

A Figura 1 mostra que ambas simulações iniciam com desempenhos próximos, mas que a diferença cresce conforme a quantidade de dados a serem computados aumenta. A versão orientada a objetos inicia os testes (de cinco mil entidades) com um *frametime* de $5{,}6$ milissegundos, enquanto a ECS inicia com $4{,}6$ milissegundos. Contudo, ao chegar em quarenta mil entidades, a diferença de tempo de quadro é expressiva: a versão orientada a objetos apresenta $61{,}2$ milissegundos enquanto a versão ECS exibe $12{,}9$ milissegundos de *frametime*.

4. Conclusão

Os resultados mostram que ambas simulações apresentam crescimento linear de *frametime* ao aumentar o número de entidades, contudo, soluções baseadas em ECS oferecem boas métricas para simulações que necessitam altíssimas quantidades de dados a serem processados enquanto simultaneamente precisam de boa flexibilidade arquitetural provida pelo modelo de componentes. Já os resultados de teste da versão orientada a objetos apresentam alta taxa de crescimento de *frametime*, finalizando com $61{,}2$ milissegundos no experimento com quarenta mil entidades, o que é muito acima do padrão mínimo aceitado para um jogo eletrônico, que é de $33{,}3$ milissegundos.

Em trabalho futuros, propomos investigar estratégias de paralelização de processamento, podendo até buscar aceleração via GPUs, buscando garantir ainda melhor desempenho, e buscar entender o comportamento da memória nestes casos.

Referências

Fabian, R. (2018). *Data-Oriented Design*. Richard Fabian.

Nystrom, R. (2011). *Game Programming Patterns*. Genever Benning.

Unity-Technologies (2024a). Website. `https://docs.unity3d.com/Packages/com.unity.entities@0.2/manual/ecs_core.html`. [online: acesso em 03-Outubro-2024].

Unity-Technologies (2024b). Website. `https://unity.com/`. [online: acesso em 29-Maio-2024].