

Introdução ao uso de Compute Shaders no Desenvolvimento de Jogos Digitais Usando Unity

Introduction to Compute Shaders in Game Development Using Unity

Jose Ricardo da Silva Junior¹, Andre Gomes¹

¹Instituto de Informática – Instituto Federal do Rio de Janeiro (IFRJ)
Engenheiro Paulo de Frontin – RJ – Brasil

jose.junior@ifrj.edu.br, andre.gs.1618@gmail.com

Abstract. Introduction: Modern game engines increasingly rely on GPU computing to support real-time simulations and procedural generation. Compute Shaders provide a practical entry point to GPGPU within graphics APIs, enabling developers to exploit massive parallelism without leaving the engine workflow. **Objective:** This paper presents a hands-on tutorial on Compute Shaders in Unity, focusing on the programming model, the CPU–GPU communication pipeline, and typical use cases in digital games. **Methodology or Steps:** The tutorial is organized into four modules: (i) fundamentals of GPU architecture and parallel execution; (ii) Compute Shader concepts (kernels, thread groups, memory access, and dispatch); (iii) data exchange between CPU and GPU using Unity resources (e.g., ComputeBuffers and textures); and (iv) incremental development of a Unity application that processes data in parallel and generates a visual output on the GPU. **Results:** Participants are expected to understand how to model problems for GPU execution, implement and integrate Compute Shaders in Unity, and extend the example project to applications such as particle simulation, image processing, VFX, and procedural content generation.

Keywords Compute shaders, GPU computing, Parallel processing, Unity game engine.

Resumo. Introdução: Motores de jogos modernos têm incorporado, de forma crescente, recursos de computação em GPU para viabilizar simulações e geração procedural em tempo real. Nesse contexto, Compute Shaders oferecem uma forma acessível de aplicar GPGPU diretamente no fluxo de trabalho de um motor de jogo. **Objetivo:** Este artigo apresenta um tutorial prático sobre Compute Shaders na Unity, abordando o modelo de programação, o fluxo de comunicação CPU–GPU e aplicações recorrentes no desenvolvimento de jogos digitais. **Metodologia ou Etapas:** O tutorial está estruturado em quatro módulos: (i) fundamentos de arquitetura de GPU e de execução paralela; (ii) conceitos de Compute Shader (kernels, grupos de threads, acesso à memória e dispatch); (iii) comunicação entre CPU e GPU com recursos da Unity (por exemplo, ComputeBuffers e texturas); e (iv) desenvolvimento incremental de uma aplicação na Unity que processa dados em paralelo e gera uma saída visual diretamente na GPU. **Resultados:** Espera-se que os participantes compreendam como modelar problemas para execução em GPU, implementem e integrem Compute Shaders na Unity e consigam evoluir o projeto-base para aplicações

como simulação de partículas, processamento de imagens, efeitos visuais e geração procedural de conteúdo.

Palavras-Chave *Compute shaders, Computação em GPU, Processamento paralelo, Motor de jogo Unity.*

1. Introdução

Nos últimos anos, a computação passou de uma solução majoritariamente sequencial para uma abordagem paralela, devido às restrições físicas e tecnológicas ao aumento da velocidade dos processadores. Dessa forma, o foco principal migrou do incremento da velocidade de um processador para o número de cores que esse processador poderia suportar. Assim, com essa mudança, paradigmas de programação precisaram se adaptar para promover soluções que possibilitem a solução de um problema de forma concorrente nos múltiplos núcleos de processamento a fim de extrair benefícios de uma arquitetura computacional paralela.

Seguindo uma arquitetura massivamente paralela, a GPU (Unidade de Processamento Gráfico) foi originalmente concebida para acelerar a rasterização em jogos e simulações em tempo real, uma vez que possui um número de núcleos de processamento superior ao de CPUs tradicionais. Inicialmente, surgiu como uma arquitetura de pipeline fixo, na qual não era possível programar os estágios do pipeline gráfico (estágios de vértice e fragmento, por exemplo), limitando-se apenas à ativação ou desativação de funcionalidades. Com a evolução da arquitetura, alguns dos estágios do pipeline de renderização puderam ser programados por meio de linguagens de *shaders* (GLSL e Cg, por exemplo), nas quais os processadores eram especializados na execução desses estágios. Porém, nesse momento, o uso da GPU transcendeu a tradicional renderização de imagens para se tornar uma poderosa aliada no processamento paralelo de tarefas complexas, sendo cunhado o termo GPGPU (*General-purpose computing on Graphics Processor Unit*). Assim, diversas áreas da ciência, tais como física [Spridon 2023] e medicina [Tahmasebi et al. 2020], se beneficiaram deste processador gráfico, mesmo sendo necessária a modelagem do problema seguindo o fluxo de dados estabelecido pelo pipeline gráfico.

A ampla adoção da GPU para a solução de problemas não gráficos motivou o surgimento da arquitetura unificada, com núcleos de processamento genéricos, capazes de ser utilizados em qualquer estágio do pipeline gráfico. Além disso, ferramentas como a linguagem CUDA [Sanders e Kandrot 2010] e OpenCL introduziram a possibilidade de uso da GPU para propósitos gerais, independentemente do fluxo de dados no pipeline gráfico. Atualmente, as GPUs deixaram de priorizar exclusivamente a rasterização, evoluindo sua arquitetura para abarcar funcionalidades específicas para GPUGPU. Um exemplo disso é a introdução dos *Tensor Cores*, processadores específicos para aplicações de inteligência artificial. Porém, as linguagens e frameworks mais especializados de GPGPU exigem uma curva de aprendizado mais acentuada e estão mais voltados para aplicações científicas, sendo, muitas vezes, descartados no desenvolvimento de jogos e simulações em tempo real.

Dessa forma, o *Compute Shader* [Alda 2023] surgiu como uma alternativa mais acessível para o processamento de propósito geral na GPU, no contexto do desenvolvimento de jogos. Um *compute shader* executa muitas *threads* organizadas em

grupos de threads e é disparado explicitamente pela CPU, em vez de ser invocado por meio da renderização de geometria. Ao contrário dos *shaders* de vértice/fragmento, os *compute shaders* não estão vinculados à rasterização; eles podem ler e escrever em *buffers* e texturas arbitrários e realizar computação paralela independente. Assim, o *compute shader* permite que desenvolvedores de jogos aproveitem o poder da GPU de forma integrada e intuitiva, utilizando APIs amplamente adotadas como DirectX, Vulkan ou OpenGL. Com isso, o uso de *compute shader* para simulações físicas mais realistas [Bui 2021], até inteligência artificial [Alda 2023], pode abrir um novo leque de possibilidades criativas e técnicas para quem desenvolve jogos.

2. Conteúdo Programático

O conteúdo programático desta proposta está dividido em quatro módulos. Nos três primeiros módulos, são apresentados conceitos importantes sobre a arquitetura de GPU e o funcionamento dos *shaders*. No último módulo, os conceitos apresentados são aplicados à construção de um exemplo real de simulação física que utiliza *compute shaders*.

2.1. Módulo 1 – Introdução às GPUs e à evolução do pipeline gráfico

Este módulo tem por objetivo apresentar os conceitos fundamentais sobre GPUs e contextualizar a evolução da computação gráfica até o surgimento dos *Compute Shaders*.

1. **O que é uma GPU:** Apresentação da evolução das GPUs e de seu papel no processamento gráfico moderno, destacando sua capacidade de executar milhares de operações em paralelo.
2. **CPU versus GPU:** Comparação entre CPU e GPU, com ênfase nas diferenças de arquitetura, modelo de execução e tipos de problemas mais adequados a cada processador.
3. **Evolução do pipeline gráfico:** Apresentação da evolução do pipeline gráfico, desde o Pipeline Fixo até a Arquitetura Unificada, mostrando como essa transformação possibilitou o uso da GPU para tarefas além da renderização.

2.2. Módulo 2 – Fundamentos da arquitetura de GPUs

Compreender os principais conceitos da arquitetura de GPUs necessários para entender o funcionamento dos *Compute Shaders*.

1. Organização da GPU: Visão geral da organização interna da GPU, apresentando *Compute Units* (ou *Streaming Multiprocessors*), núcleos de processamento e execução em paralelo.
2. Modelo de execução: Introdução aos conceitos de *Threads*, *Thread Groups* (*Workgroups*) e execução paralela, enfatizando como milhares de *threads* podem ser executadas simultaneamente.
3. Hierarquia de memória: Apresentação simplificada dos diferentes tipos de memória disponíveis na GPU e de sua influência no desempenho das aplicações.
4. Eficiência dos *Compute Shaders*: Discussão das características arquiteturais que tornam a GPU adequada para tarefas altamente paralelizáveis.

2.3. Módulo 3 – Introdução aos *Compute Shaders*

Apresentar o modelo de programação baseado em *Compute Shaders* e o fluxo de execução utilizado no motor de jogo Unity.

1. O que são *Compute Shaders*: Definição, características e diferenças em relação aos *shaders* utilizados no pipeline gráfico tradicional.
2. Modelo de programação: Apresentação dos principais conceitos envolvidos: *Kernels*, *Threads*, *Thread Groups*, *Dispatch*, índices globais e locais.
3. Comunicação entre CPU e GPU: Introdução ao fluxo de envio e recebimento de dados por meio de *buffers* e texturas.
4. Estrutura de um *Compute Shader* na Unity: Apresentação da estrutura básica de um arquivo de *Compute Shader* e da integração com scripts em C#.

2.4. Módulo 4 – Desenvolvimento prático de uma aplicação utilizando *Compute Shaders* na Unity

Consolidar os conceitos apresentados nos módulos anteriores por meio do desenvolvimento incremental de uma aplicação na Unity, demonstrando o fluxo completo de comunicação entre CPU e GPU e sua aplicação a problemas típicos do desenvolvimento de jogos digitais.

1. Configuração do projeto e estrutura da aplicação: Apresentação da estrutura de um projeto que utiliza *Compute Shaders* na Unity, incluindo a criação do *shader*, sua integração com scripts em C# e a configuração dos recursos necessários para sua execução.
2. Primeiros passos com *Compute Shaders*: Implementação de um *Compute Shader* simples para apresentar o fluxo básico de execução, desde a definição do *kernel* até sua invocação por meio do comando *Dispatch*. Nesta etapa, serão apresentados os principais elementos da linguagem e do modelo de programação.
3. Comunicação entre CPU e GPU: Introdução ao uso de *ComputeBuffers* para transferência de dados entre CPU e GPU. Serão demonstrados o envio de dados para processamento paralelo, a execução do *Compute Shader* e a recuperação dos resultados gerados pela GPU, destacando o fluxo completo de processamento.
4. Geração procedural com *Compute Shaders*: Expansão da aplicação para gerar uma textura diretamente na GPU, utilizando um *buffer* de textura, permitindo visualizar os resultados do processamento paralelo em tempo real. Também serão apresentados mecanismos para enviar parâmetros da CPU ao *Compute Shader*, tornando a aplicação dinâmica e interativa.
5. Aplicações no desenvolvimento de jogos: Discussão sobre como os conceitos implementados ao longo do projeto podem ser aplicados a diferentes sistemas presentes em jogos digitais, como a geração procedural de conteúdo, a simulação de partículas, o processamento de imagens, os efeitos visuais e outras aplicações aceleradas por GPU.

3. Abordagem Metodológica

O tutorial será conduzido por meio da integração entre exposição teórica e desenvolvimento prático, priorizando uma abordagem orientada à aplicação dos conceitos. Inicialmente, serão apresentados os fundamentos necessários para

compreender o funcionamento das GPUs e o modelo de programação dos *Compute Shaders*. Em seguida, esses conceitos serão consolidados por meio da construção incremental de uma aplicação com a Unity.

A implementação prática será realizada em conjunto com os participantes, permitindo acompanhar todas as etapas do desenvolvimento, desde a configuração inicial do projeto até a execução de um *Compute Shader* capaz de processar dados e gerar resultados visualmente observáveis. Ao longo da implementação, serão discutidas boas práticas, o fluxo de comunicação entre CPU e GPU e possibilidades de extensão do projeto para diferentes problemas encontrados no desenvolvimento de jogos digitais.

A metodologia privilegia a aprendizagem baseada na prática (*learning by doing*), estimulando a participação ativa dos participantes, a resolução de dúvidas durante a implementação e a associação constante entre os conceitos teóricos e suas aplicações em cenários reais do desenvolvimento de jogos.

4. Discussões e Contribuições Esperadas

Espera-se que o tutorial estimule discussões sobre o papel das GPUs no desenvolvimento de jogos digitais e sobre as oportunidades oferecidas pelos *Compute Shaders* para a implementação de sistemas com elevado grau de paralelismo. Nesse contexto, pretende-se discutir os critérios para identificar problemas adequados ao processamento em GPU, bem como as limitações e os desafios associados à comunicação entre CPU e GPU e ao desenvolvimento de algoritmos paralelos. Dessa forma, torna-se importante que o participante desenvolva um senso crítico na avaliação dos cenários que mais se beneficiam do uso de GPU.

Também se espera promover reflexões sobre a crescente utilização de *Compute Shaders* em motores gráficos modernos e em diferentes áreas do desenvolvimento de jogos, como a geração procedural de conteúdo, a simulação física, o processamento de imagens e os efeitos visuais. Essas discussões visam aproximar os participantes das tecnologias empregadas na indústria e incentivar sua adoção em projetos acadêmicos e profissionais. Usando uma linguagem mais próxima daquelas empregadas na construção dos *shaders* tradicionais de renderização, os *compute shaders* têm a capacidade de oferecer uma maior flexibilidade de processamento de dados com uma baixa barreira de entrada quando comparada a outras abordagens tradicionais de GPGPU, como a própria tecnologia CUDA.

Como principal contribuição, o tutorial busca fornecer uma base conceitual e prática para que os participantes iniciem o desenvolvimento de aplicações utilizando *Compute Shaders* na Unity, compreendendo o fluxo de trabalho envolvido e adquirindo conhecimentos que possam ser posteriormente transferidos para outras engines e APIs gráficas. Espera-se, ainda, despertar o interesse pelo aprofundamento em programação para GPUs e em computação paralela aplicada ao desenvolvimento de jogos digitais.

Além da implementação desenvolvida durante o tutorial, pretende-se apresentar estratégias para a evolução do projeto construído em sala, indicando possibilidades de extensão para aplicações mais complexas. Dessa forma, os participantes poderão utilizar o exemplo desenvolvido como ponto de partida para seus próprios experimentos e projetos, favorecendo a continuidade da aprendizagem após o término do tutorial.

Referências

- Alda, Á. (2023). Introduction to shaders. In *Beginner's Guide to Unity Shader Graph: Create Immersive Game Worlds Using Unity's Shader Tool*, pages 1–21. Springer.
- Bui, Q. (2021). Physics engine on the gpu with opengl compute shaders. Master's thesis, California Polytechnic State University.
- Sanders, J. e Kandrot, E. (2010). *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional.
- Spridon, D. (2023). Advances in cuda for computational physics. *Bulletin of the Transilvania University of Brasov. Series III: Mathematics and Computer Science*, pages 227–236.
- Tahmasebi, N., Boulanger, P., Yun, J., Fallone, G., Noga, M., e Punithakumar, K. (2020). Real-time lung tumor tracking using a cuda enabled nonrigid registration algorithm for mri. *IEEE journal of translational engineering in health and medicine*, 8:1–8.