

Real Engine

Andamento

Version 1.0 • Proposed



Nomes:
Matrículas:

Giovanni, Lucas e Rael
108307, 108347, 108344

Table of Contents

Real Engine	3
Class Diagrams	3
Graphics	3
Renderer Diagram	3
Low Level Diagram	3
Allocators Diagram	4
Buffers Diagram	5
Images Diagram	6
Managers Diagram	7
Scheduler	8
Buffer	8
ConfigurationManager	11
Device	11
GraphicsManager	14
Image	16
Instance	19
Memory	21
MemoryManager	22
PoolAllocator	23
Queue	26
Result	26
IAllocator	28
ErrorCode	28

Real Engine

Real Engine
Version Proposed

Class Diagrams

Class Diagrams
Version 1.0 Proposed

Graphics

O pacote Graphics da Real Engine é responsável pela manipulação de entes gráficos e controle dos dispositivos físicos que são responsáveis pela renderização. Portanto, são tarefas deste pacote:

1. Analisar o estado dos dispositivos físicos, geralmente GPUs, e verificar por erros durante a execução do programa;
2. Movimentar dados entre a memória principal e a memória do dispositivo, mantê-los sincronizados e coesos em relação às suas respectivas caches;
3. Executar programas, conhecidos como shaders, nestes dispositivos físicos que realizam operações matemáticas relevantes à renderização;
4. Renderizar formas geométricas na tela do usuário da aplicação através de uma combinação de estruturas conhecidas como buffers de vértices, buffers de índices e shaders.

Graphics
Version 1.0 Proposed

Renderer Diagram

Class Diagram in package 'Graphics'

Renderer
Version 1.0

Scheduler

Queue

Figure 1: Renderer

Low Level Diagram

Class Diagram in package 'Graphics'

O diagrama Low Level apresenta as relações entre os objetos de mais baixo nível da Real Engine e que são responsáveis por realizar as operações mais elementares.

Low Level
Version 1.0

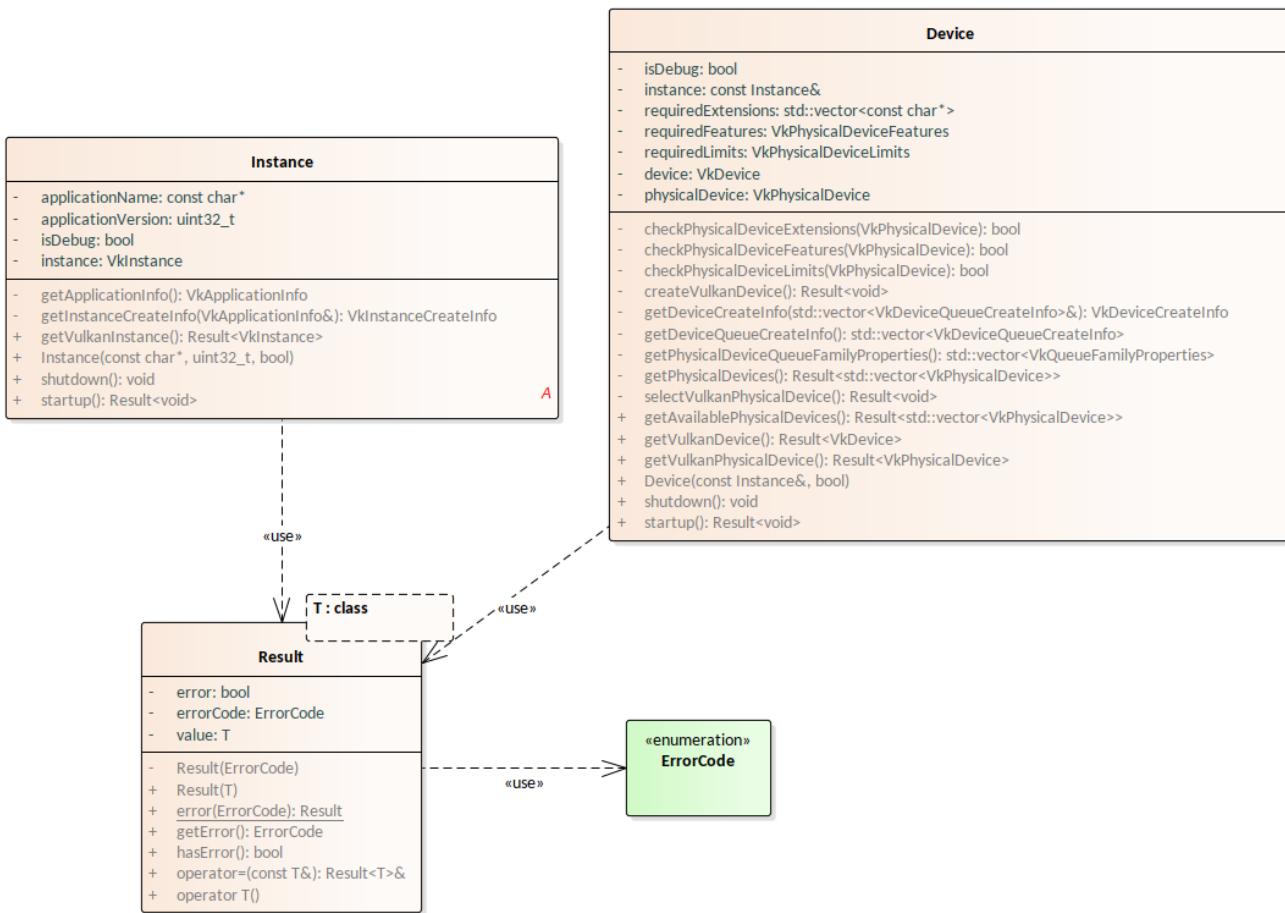


Figure 2: Low Level

Allocators Diagram

Class Diagram in package 'Graphics'

O diagrama Allocators demonstra as relações entre os alocadores e a memória. Todas essas referências a memória referem-se a memória do dispositivo de vídeo, ou seja, a memória de uma GPU.

Allocators
Version 1.0

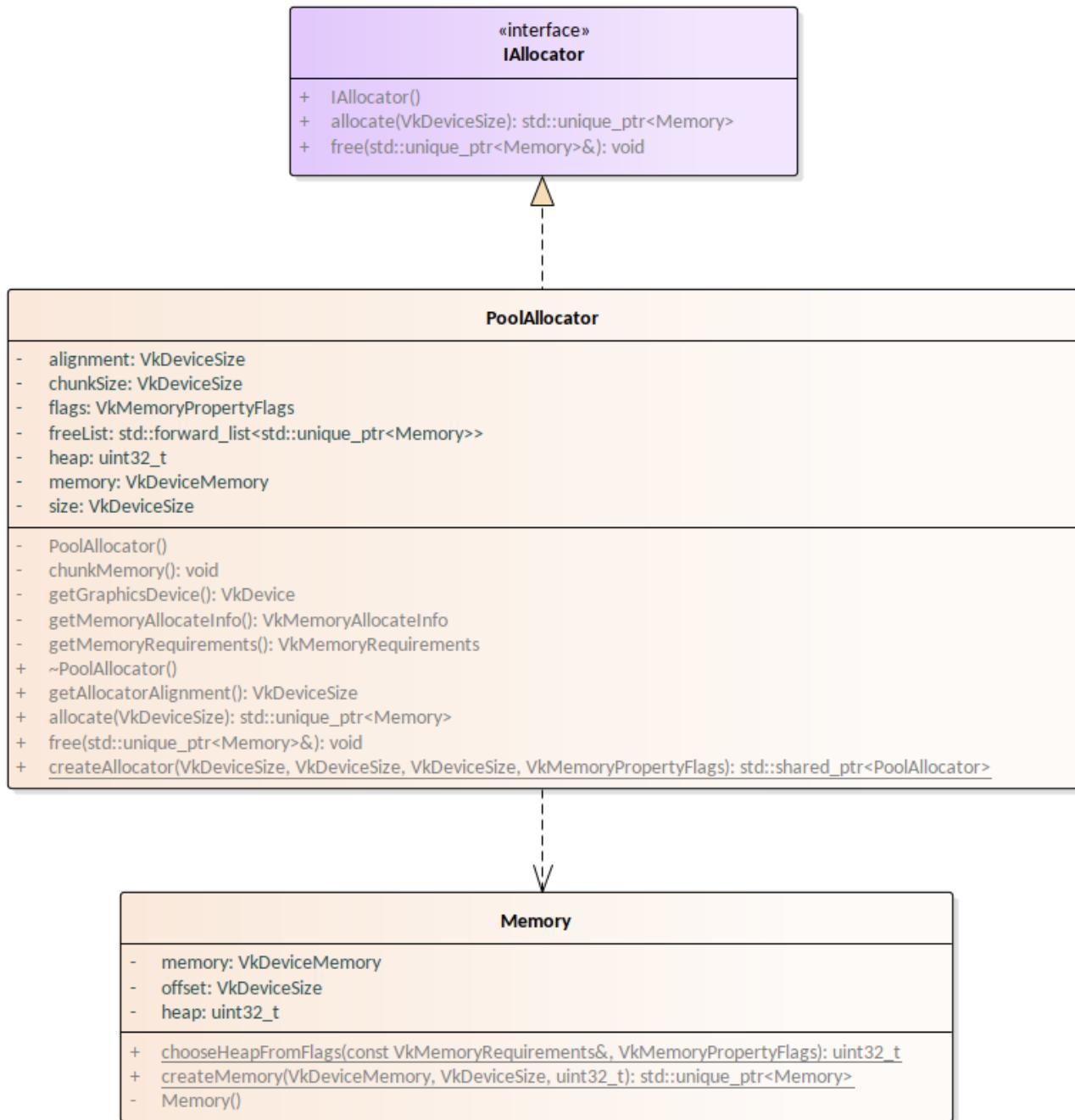


Figure 3: Allocators

Buffers Diagram

Class Diagram in package 'Graphics'

O diagrama de Buffers apresenta as relações entre a memória, as filas e um dos dois tipos de recursos disponíveis na Real Engine e na API Vulkan: Buffers.

Buffers
Version 1.0

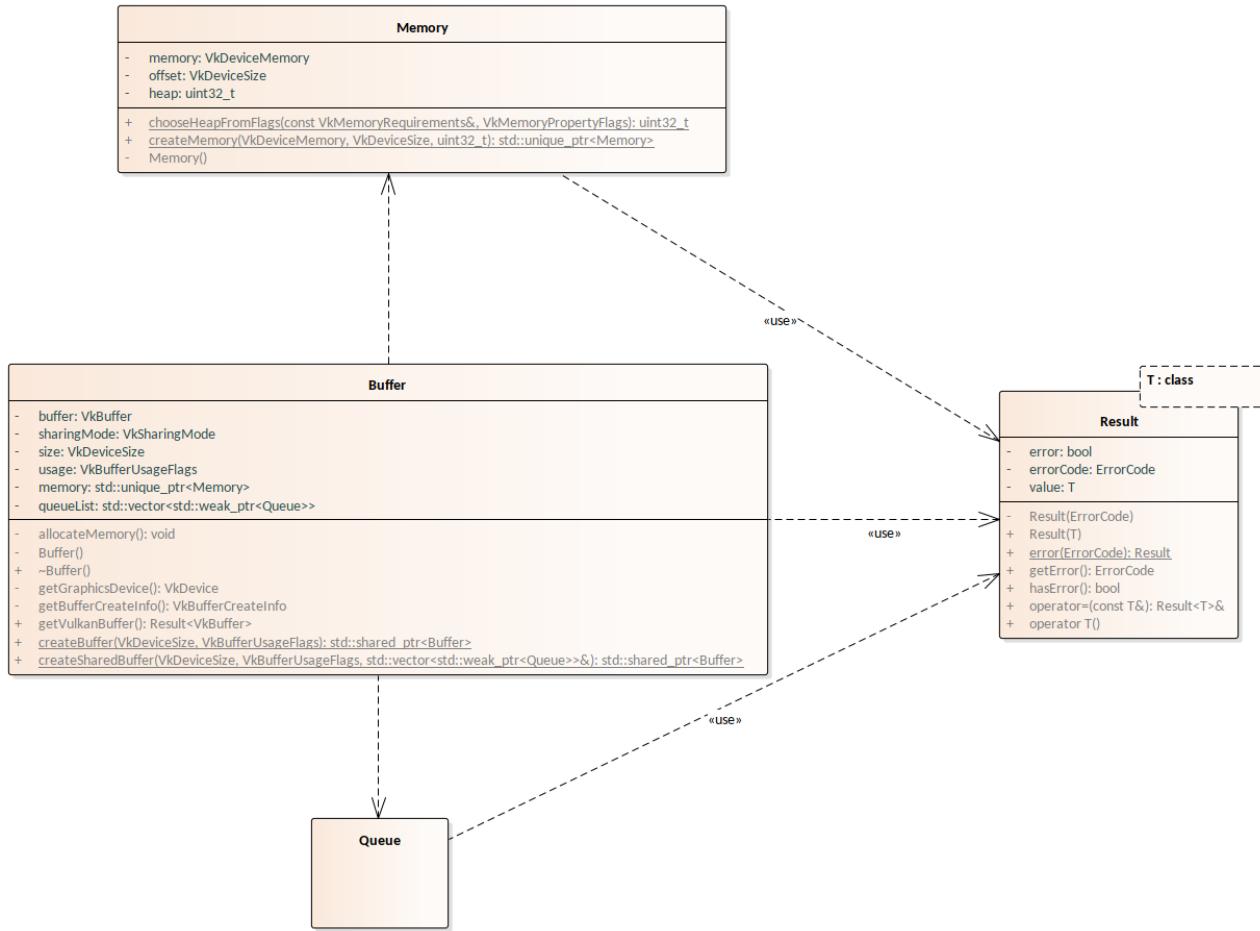


Figure 4: Buffers

Images Diagram

Class Diagram in package 'Graphics'

O diagrama de Images apresenta as relações entre a memória, as filas e um dos dois tipos de recursos disponíveis na Real Engine e na API Vulkan: Images.

Images
Version 1.0

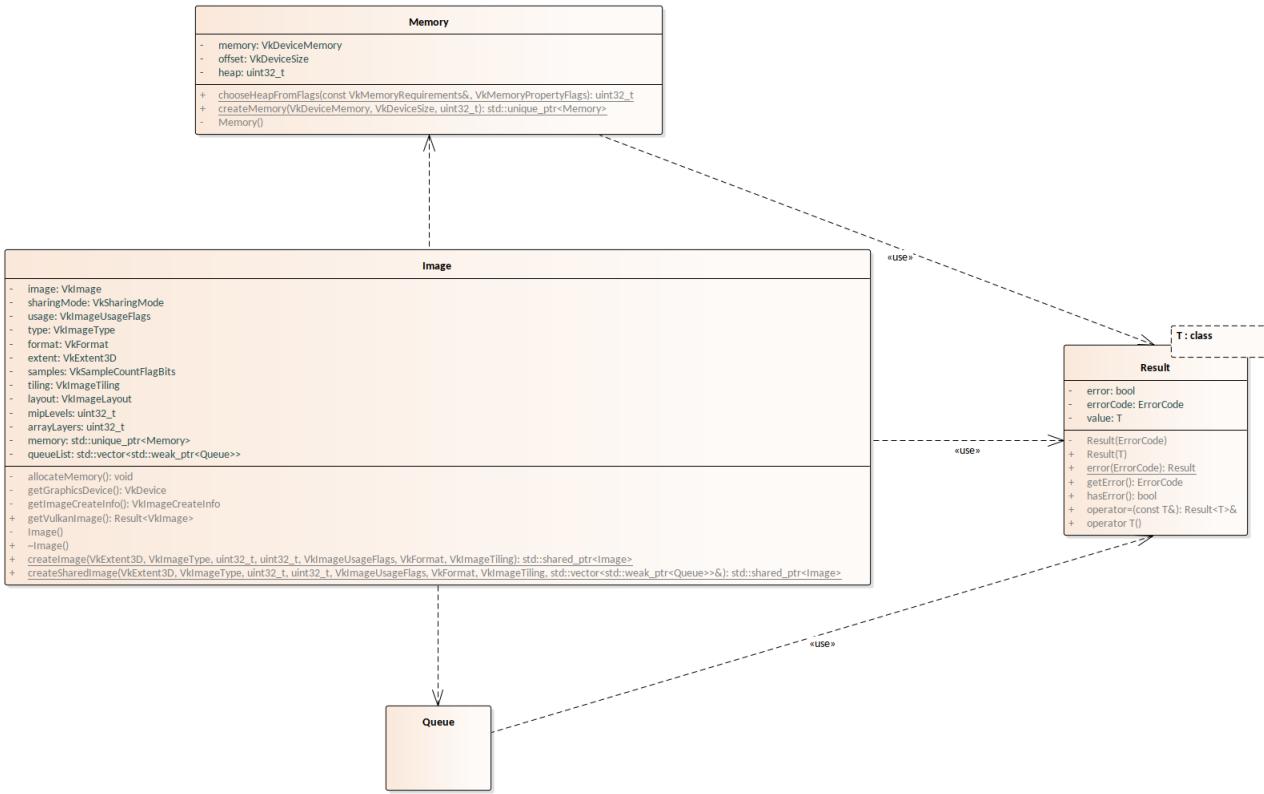


Figure 5: Images

Managers Diagram

Class Diagram in package 'Graphics'

O diagrama de Managers apresenta as relações entre as entidades de mais alto nível da Real Engine. Cada Manager opera e administra um subsistema da mesma e suas interrelações são fundamentais para determinar a ordem de inicialização destes subsistemas.

Managers
Version 1.0

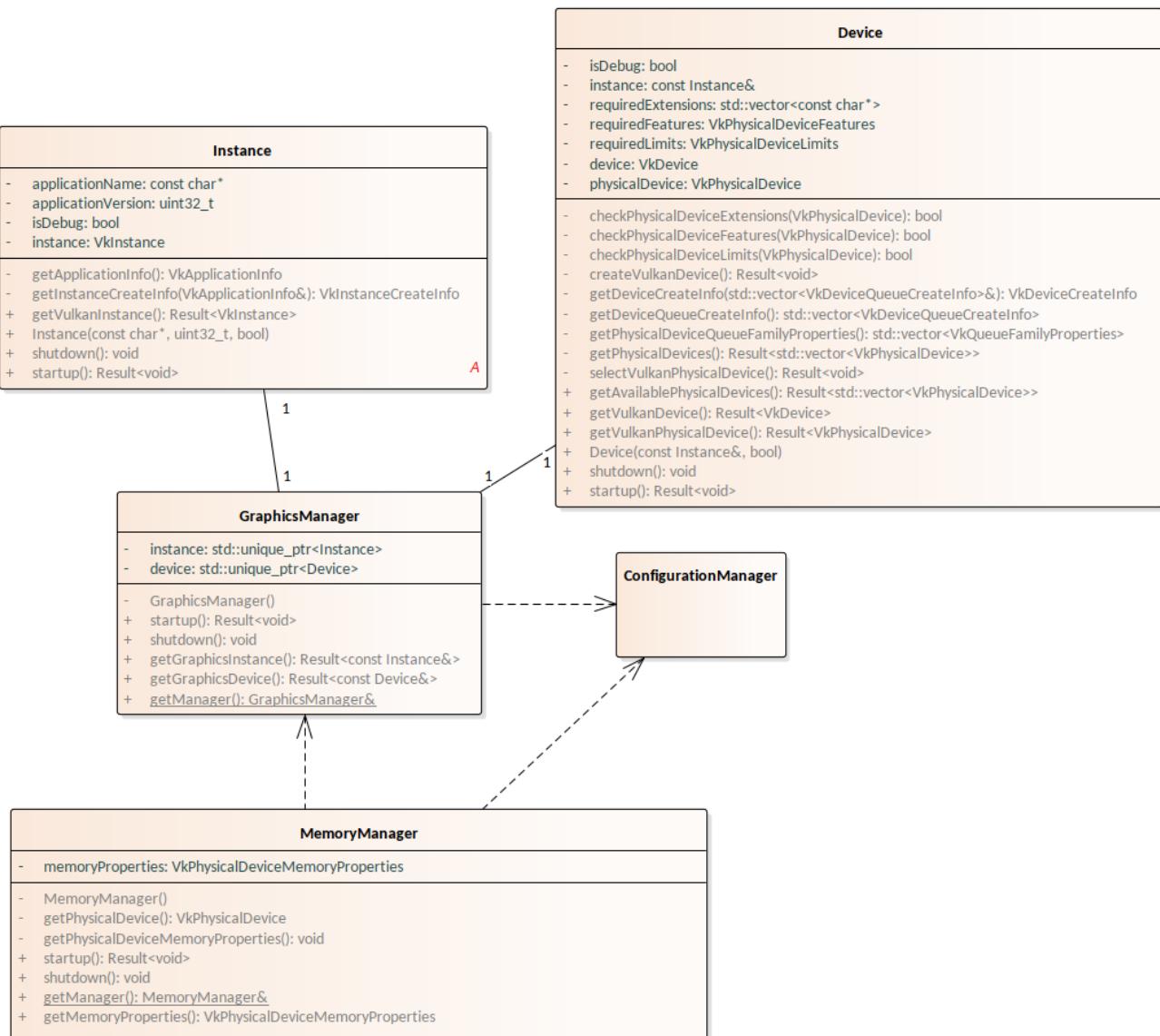


Figure 6: Managers

Scheduler

Class in package 'Graphics'

Scheduler
Version 1.0 Proposed

Buffer

Class in package 'Graphics'

A classe Buffer é responsável por criar uma fina camada de abstração sobre um dos dois tipos de recursos disponíveis na API Vulkan: o VkBuffer. O VkBuffer representa uma área contígua de memória de vídeo, onde podem ser armazenadas informações provenientes da memória utilizada diretamente pela CPU através do barramento PCI Express.

Os Buffers podem ser utilizados de múltiplas maneiras, como vertex buffers, index buffers, constant buffers, etc. Além disso, podem ser compartilhados entre múltiplas filas de processamento da GPU. Por fim, cada Buffer possui uma fatia de memória de vídeo, alocada durante sua criação através de um Allocator adequado.

É importante manter em mente que os Buffers, podem ser manipulados somente através de shared_ptr e weak_ptr, o

que permite que estes objetos sejam compartilhados entre vários objetos diferentes e ainda assim mantendo o grau de importância relativo a cada um.

A classe Buffer necessita aplicar a regra dos 5 em C++, efetuando a deleção dos seguintes métodos:

1. O construtor padrão que permite a criação de objetos resetados;
2. O construtor de cópia que permite copiar outros objetos do mesmo tipo;
3. O construtor de movimento que permite incorporar outros objetos através da std::move;
4. O operador de atribuição que permite copiar outros objetos do mesmo tipo;
5. O operador de atribuição que permite incorporar outros objetos através da std::move.

Buffer
Version 1.0 Proposed

ATTRIBUTES
<p>◆ buffer : VkBuffer Private</p> <p>O atributo que armazena a handle para o recurso do tipo VkBuffer na API Vulkan.</p>
<p>◆ sharingMode : VkSharingMode Private</p> <p>O atributo que armazena o modo de compartilhamento de um Buffer, ou seja, se ele é exclusivo ou concorrente.</p>
<p>◆ size : VkDeviceSize Private</p> <p>O atributo que armazena o tamanho do espaço que o Buffer ocupa na memória de vídeo, em bytes.</p>
<p>◆ usage : VkBufferUsageFlags Private</p> <p>O atributo que contém o uso para o qual este Buffer é destinado.</p>
<p>◆ memory : std::unique_ptr<Memory> Private</p> <p>O atributo que guarda o unique_ptr da memória de vídeo associada à este Buffer.</p>
<p>◆ queueList : std::vector<std::weak_ptr<Queue>> Private</p> <p>O atributo que lista as múltiplas filas de processamento que poderão utilizar este Buffer, se ele for compartilhado.</p>

OPERATIONS
<p>◆ allocateMemory () : void Private</p> <p>Este método auxiliar tem como função efetivar a alocação de memória de vídeo para sustentar o buffer, através de um Allocator apropriado, e de efetuar a ligação entre o buffer e tal região de memória.</p> <p>Após a execução deste método, que deve acontecer nos métodos que criam o buffer, este objeto terá memória reservada no dispositivo físico, onde poderá realizar múltiplas operações.</p>

<p>◆ Buffer () : Private</p> <p>O construtor padrão e privado de objetos do tipo Buffer, seu único objetivo é criar um objeto com seus atributos completamente resetados.</p>

Observação Importante: Para construir este objeto corretamente, deve-se utilizar para construção os seguintes métodos: `createBuffer` ou `createSharedBuffer`.

◆ `~Buffer ()` : Public

O destrutor padrão de Buffer, cujo objetivo é retornar a memória associada para o alocador que a providenciou, além de destruir o recurso de tipo `VkBuffer` junto à API Vulkan, nulificando, portanto, o atributo que armazena o handle: `buffer`.

◆ `getGraphicsDevice ()` : `VkDevice` Private

O método auxiliar `getGraphicsDevice` tem como objetivo adquirir o dispositivo lógico da aplicação através da API Vulkan.

Para atingir tal objetivo, o método obtém uma referência para o singleton de `GraphicsManager` e o utiliza para acessar o objeto de tipo `Device`. Por fim, requisita a handle do `VkDevice` diretamente ao objeto responsável por administrar o dispositivo lógico na Real Engine.

O método retornará o dispositivo lógico caso encontrado, senão irá retornar um código de erro que providencie maiores informações sobre o problema encontrado.

◆ `getBufferCreateInfo ()` : `VkBufferCreateInfo` Private

O método auxiliar que tem como função criar e preencher a estrutura do tipo `VkBufferCreateInfo`. Esta estrutura é necessária para requisitar a criação de um objeto `VkBuffer` ao dispositivo lógico, através da API Vulkan.

◆ `getVulkanBuffer ()` : `Result<VkBuffer>` Public

Este método tem como objetivo permitir a obtenção da handle ao objeto do tipo `VkBuffer` para que outros objetos possam realizar operações relacionadas à API Vulkan que necessitem utilizar a handle.

O método irá retornar a handle em um objeto do tipo `Result` caso ela exista, senão, irá retornar um código de erro no objeto.

◆ `createBuffer (size : VkDeviceSize , usage : VkBufferUsageFlags)` : `std::shared_ptr<Buffer>` Public

O método `createBuffer` é o que permite a criação de objetos do tipo `Buffer` que sejam exclusivos, ou seja, não precisam ser compartilhados entre múltiplas filas de processamento gráfico e, portanto, não necessitará receber uma estrutura que contenha uma lista de objetos do tipo `Queue`.

Este método retorna um `Buffer` criado a partir dos parâmetros especificados, realiza a alocação de memória necessária e efetua a ligação entre o `Buffer` e a região de memória alocada. Por fim, retorna o `Buffer` através de um `shared_ptr`, permitindo o compartilhamento de objetos do tipo `Buffer` entre múltiplos objetos enquanto evita leaks de memória do dispositivo físico.

◆ `createSharedBuffer (size : VkDeviceSize , usage : VkBufferUsageFlags , queues : std::vector<std::weak_ptr<Queue>> &)` : `std::shared_ptr<Buffer>` Public

O método `createSharedBuffer` é o que permite a criação de objetos do tipo `Buffer` que sejam concorrentes, ou seja, que precisam ser compartilhados entre múltiplas filas de processamento gráfico e, portanto, necessitará receber uma estrutura que contenha uma lista de objetos do tipo `Queue`.

Este método retorna um `Buffer` criado a partir dos parâmetros especificados, realiza a alocação de memória necessária e efetua a ligação entre o `Buffer` e a região de memória alocada. Por fim, retorna o `Buffer` através de um `shared_ptr`,

permitindo o compartilhamento de objetos do tipo Buffer entre múltiplos objetos enquanto evita leaks de memória do dispositivo físico.

ConfigurationManager

Class in package 'Graphics'

O ConfigurationManager é a classe que gerencia o subsistema de configuração. Sua função básica é ler arquivos de texto JSON e extrair as informações relevantes para o programa, posteriormente os outros subsistemas irão requisitar tais informações e elas deverão ser encaminhadas corretamente caso tenham sido extraídas com sucesso ou, então, retornar um erro apropriado.

A classe ConfigurationManager necessita aplicar a regra dos 5 em C++, efetuando a deletação dos seguintes métodos:

1. O construtor padrão que permite a criação de objetos resetados;
2. O construtor de cópia que permite copiar outros objetos do mesmo tipo;
3. O construtor de movimento que permite incorporar outros objetos através da std::move;
4. O operador de atribuição que permite copiar outros objetos do mesmo tipo;
5. O operador de atribuição que permite incorporar outros objetos através da std::move.

ConfigurationManager
Version 1.0 Proposed

Device

Class in package 'Graphics'

A classe Device é responsável por abstrair os tipos VkDevice e VkPhysicalDevice da API Vulkan. Devido a isto, esta classe é extensa e têm incubida para si, várias funções importantes para o funcionamento da Real Engine. Entre estas funcionalidades, destacam-se as seguintes:

1. Procurar por dispositivos físicos (GPUs) que atendam as necessidades da Real Engine, em termos de suporte às extensões pelos seus drivers, suporte as funcionalidades requeridas pelos processos de renderização e pelos limites inerentes ao hardware em questão;
2. Escolher entre os dispositivos físicos encontrados, aquele que é mais adequado a ser utilizado e dele extrair várias informações a respeito de suas filas de processamento e quais delas tem capacidade de processamento gráfico;
3. Montar uma lista linear com todas as filas do dispositivo físico que possuem capacidades gráficas e as fornecer para a criação de um dispositivo lógico;
4. A criação de um dispositivo lógico (VkDevice) que irá gerenciar todas as filas de processamento gráfico do dispositivo físico selecionado.

A classe Device necessita aplicar a regra dos 5 em C++, efetuando a deletação dos seguintes métodos:

1. O construtor padrão que permite a criação de objetos resetados;
2. O construtor de cópia que permite copiar outros objetos do mesmo tipo;
3. O construtor de movimento que permite incorporar outros objetos através da std::move;
4. O operador de atribuição que permite copiar outros objetos do mesmo tipo;
5. O operador de atribuição que permite incorporar outros objetos através da std::move.

Device
Version 1.0 Proposed

ATTRIBUTES

◆ isDebug : bool Private

O atributo que informa se as camadas de logging, depuração e perfilamento da API Vulkan serão ativadas no dispositivo lógico.

◆ instance : const Instance& Private

ATTRIBUTES
O atributo que armazena uma referência para a instância da aplicação, utilizada para acessar a instância Vulkan.
◆ requiredExtensions : std::vector<const char*> Private Um vetor com todas as extensões requeridas pela aplicação para serem procuradas no dispositivo físico, deve ser lido de um arquivo de texto.
◆ requiredFeatures : VkPhysicalDeviceFeatures Private O atributo que armazena a estrutura que contém todas as funcionalidades requeridas pela aplicação e pela engine.
◆ requiredLimits : VkPhysicalDeviceLimits Private O atributo que armazena os limites que a engine requer do dispositivo físico a ser escolhido.
◆ device : VkDevice Private O atributo que armazena a handle do dispositivo lógico da API Vulkan. O dispositivo lógico é utilizado para realizar a maior parte das operações, tanto de renderização como de movimentação de dados.
◆ physicalDevice : VkPhysicalDevice Private O atributo que armazena a handle do dispositivo físico da API Vulkan, pode ser utilizado para obter várias informações a respeito do dispositivo, como seus limites e funcionalidades disponíveis.

OPERATIONS
◆ checkPhysicalDeviceExtensions (pd : VkPhysicalDevice) : bool Private O método auxiliar que verifica se um dado dispositivo físico possui todas as extensões requisitadas pela engine e pela aplicação. Se este for o caso retornará verdadeiro, caso contrário, retornará falso.
◆ checkPhysicalDeviceFeatures (pd : VkPhysicalDevice) : bool Private O método auxiliar cujo objetivo é verificar se o dispositivo físico fornecido possui todas as funcionalidades que são requeridas pela engine e aplicação, e.g. multiamostragem, renderização indireta, etc. Se todas as funcionalidades necessárias, especificadas no atributo requiredFeatures, estiverem disponíveis, o método retornará verdadeiro, senão, retornará falso.
◆ checkPhysicalDeviceLimits (pd : VkPhysicalDevice) : bool Private O método auxiliar que verifica se o dispositivo físico especificado por parâmetro possui limites iguais ou maiores que os requeridos pela aplicação e engine e, que estão especificados no atributo requiredLimits do objeto.
◆ createVulkanDevice () : Result<void> Private

Um método auxiliar que tem como função criar um dispositivo lógico junto a API Vulkan.

Este método auxiliar faz uso de outros dois métodos auxiliares: `getDeviceCreateInfo` e `getDeviceQueueCreateInfo`, para obter as informações necessárias que precisam ser informadas para a API.

❖ `getDeviceCreateInfo (deviceQueueCreateInfo : std::vector<VkDeviceQueueCreateInfo>&) : VkDeviceCreateInfo` Private

Um método auxiliar cujo objetivo é preencher a estrutura que detalha as informações necessárias para criar o dispositivo lógico junto a API Vulkan. É responsável por ativar funcionalidades, camadas e extensões junto ao dispositivo e, portanto, será afetado pelos limites do dispositivo físico e pelo atributo que especifica a ativação das camadas de logging, perfilamento e depuração.

❖ `getDeviceQueueCreateInfo () : std::vector<VkDeviceQueueCreateInfo>` Private

Um método auxiliar que percorre as filas de processamento do dispositivo físico para selecionar aqueles que são capazes de processamento gráfico e as retorna em um vetor para serem utilizadas na criação do dispositivo lógico, reservando-as para o uso da aplicação.

❖ `getPhysicalDeviceQueueFamilyProperties () : std::vector<VkQueueFamilyProperties>` Private

Um método auxiliar que obtém as propriedades das famílias de filas de processamento gráfico do dispositivo físico e as retorna em um vetor. Estas famílias podem, então, serem verificadas por capacidades gráficas e posteriormente utilizadas junto ao dispositivo lógico.

❖ `getPhysicalDevices () : Result<std::vector<VkPhysicalDevice>>` Private

Um método auxiliar que encontra todos os dispositivos físicos em uma instância e os coloca em um vetor de tamanho apropriado. Por fim, o método retorna esse vetor para que as propriedades dos dispositivos físicos possam ser analisadas e o mais apto entre eles ser escolhido.

❖ `selectVulkanPhysicalDevice () : Result<void>` Private

O método auxiliar responsável por selecionar o dispositivo físico mais apto a ser utilizado pela aplicação e engine.

O dispositivo escolhido deverá possuir todos os requerimentos mínimos de funcionalidades, extensões e limites físicos, além disso, possuirá o maior número de filas de processamento gráfico e todas elas serão exploradas pelo dispositivo lógico criado futuramente.

❖ `getAvailablePhysicalDevices () : Result<std::vector<VkPhysicalDevice>>` Public

Esse método tem como objetivo retornar todos os dispositivos físicos que são aptos a rodar a aplicação e serve para fornecer ao usuário uma lista caso ele deseje escolher outro dispositivo além do padrão.

❖ `getVulkanDevice () : Result<VkDevice>` Public

Esse método serve para retornar a handle para o dispositivo lógico da API Vulkan e, só irá retornar o valor caso este tenha sido apropriadamente criado, senão irá retornar um erro através do objeto `Result`.

Observação Importante: Este método só deve ser invocado após a inicialização deste objeto de tipo `Device` pelo método `startup`.

◆ **getVulkanPhysicalDevice () : Result<VkPhysicalDevice> Public**

Esse método serve para retornar a handle para o dispositivo físico da API Vulkan e, só irá retornar o valor caso este tenha sido apropriadamente criado, senão irá retornar um erro através do objeto Result.

Observação Importante: Este método só deve ser invocado após a inicialização deste objeto de tipo Device pelo método startup.

◆ **Device (inst : const Instance&, debug : bool) : Public**

O construtor padrão para objetos do tipo Device. Ele requer uma referência para um objeto Instance que não será modificado, apenas utilizado para acessar os dispositivos físicos contidos no computador.

Além disso, o construtor também recebe um booleano que determina se ativará as camadas de logging, perfilamento e depuração no dispositivo lógico.

Observação Importante: O construtor não inicializa os objetos Vulkan e, portanto, antes deste objeto ser utilizado, o seu método startup precisa ser chamado ou os outros métodos retornarão erros.

◆ **shutdown () : void Public**

O método cuja função é limpar e destruir objetos, incluindo objetos da API Vulkan, que são utilizados para suas funções. Em particular, irá destruir o dispositivo lógico e nullificar sua handle.

É importante manter em mente que esse método deve ser invocado antes do shutdown do objeto Instance ou irá gerar erros na API Vulkan, no entanto, este método não invoca o shutdown do objeto Instance que deverá ser invocado manualmente.

◆ **startup () : Result<void> Public**

O método startup tem o propósito de invocar os dois métodos auxiliares selectVulkanPhysicalDevice e createVulkanDevice e verificar se os dois rodaram sem erros.

Este método é o que de fato inicializa os objetos Vulkan e deve ser chamado após o startup do objeto Instance cuja referência está armazenada nesse objeto do tipo Device ou irá causar erros quando tentar obter os dispositivos físicos relacionados a instância Vulkan.

GraphicsManager

Class in package 'Graphics'

O GraphicsManager é a classe que gerencia o subsistema gráfico da Real Engine. Sua função é manter ponteiros exclusivos para os objetos fundamentais da API Vulkan que são necessários para a execução da aplicação e o funcionamento da engine.

Quando outros subsistemas ou objetos necessitarem acessar os objetos da API Vulkan, poderão requisitá-los facilmente ao obter uma referência a única instância existente do GraphicsManager.

A classe GraphicsManager necessita aplicar a regra dos 5 em C++, efetuando a deleção dos seguintes métodos:

1. O construtor padrão que permite a criação de objetos resetados;
2. O construtor de cópia que permite copiar outros objetos do mesmo tipo;
3. O construtor de movimento que permite incorporar outros objetos através da std::move;
4. O operador de atribuição que permite copiar outros objetos do mesmo tipo;

5. O operador de atribuição que permite incorporar outros objetos através da std::move.

GraphicsManager
Version 1.0 Proposed

ATTRIBUTES

◆ instance : std::unique_ptr<Instance> Private

O atributo responsável por guardar a instância única do objeto de tipo Instance que é inicializado pelo próprio objeto no método startup.

◆ device : std::unique_ptr<Device> Private

O atributo responsável por guardar a instância única do objeto de tipo Device que é inicializado pelo próprio objeto no método startup.

OPERATIONS

◆ GraphicsManager () : Private

O construtor padrão de GraphicsManager e que não pode ser utilizado. O seu único objetivo é resetar os valores dos atributos de tal maneira que eles possam ser setados apropriadamente no método startup.

◆ startup () : Result<void> Public

O método startup é fundamental e deve ser utilizado para inicializar o objeto do tipo GraphicsManager antes de qualquer tentativa de utilizar funções da API Vulkan.

Ao invocar o método, ele criará novas instâncias únicas de Instance e Device e, logo depois, irá inicializá-los, chamando seus próprios métodos startup.

◆ shutdown () : void Public

O método shutdown é importante para finalizar corretamente o uso da aplicação e deve ser chamado no fim da aplicação, após terem sido executadas todas as ações pendentes no dispositivo físico (GPU).

◆ getGraphicsInstance () : Result<const Instance&> Public

O método getGraphicsInstance serve para retornar uma referência, que não pode ser modificada, ao objeto de tipo Instance, permitindo que outros objetos possam, por exemplo, adquirir sua handle e utilizá-lo em outras chamadas da API Vulkan que necessitem da especificação do parâmetro VkInstance.

◆ getGraphicsDevice () : Result<const Device&> Public

O método getGraphicsDevice serve para retornar uma referência, que não pode ser modificada, ao objeto de tipo Device, permitindo que outros objetos possam, por exemplo, adquirir sua handle e utilizá-lo em outras chamadas da API Vulkan que necessitem da especificação do parâmetro VkDevice ou VkPhysicalDevice.

◆ `getManager () : GraphicsManager& Public`

O método `getManager` tem como objetivo retornar uma referência para a instância única do `GraphicsManager`, com a finalidade de que os seus métodos possam ser invocados de qualquer posição do código, até mesmo dentro de outros Managers.

Se a ocasião for a primeira vez em que o método `getManager` está sendo invocado, então o objeto de tipo `GraphicsManager` será criado uma única vez e, antes de ser utilizado, deve ser corretamente inicializado através do método `startup`.

Image

Class in package 'Graphics'

A classe `Image` é responsável por criar uma fina camada de abstração sobre um dos dois tipos de recursos disponíveis na API Vulkan: o `VkImage`. O `VkImage` representa uma estrutura que pode ser armazenada na GPU de maneiras extremamente otimizadas e em diversos formatos. Estas funcionalidades incluem uma distribuição otimizada dos dados na memória de vídeo, compreensão destes dados, etc.

As `Images` podem ser utilizadas de várias maneiras, sendo a principal utilidade delas a criação de texturas e sprites. As `Images` podem possuir múltiplos `mipLevels`, podem sofrer operações de anti-aliasing como `multisampling`, podem ser compartilhadas entre múltiplas filas de processamento gráfico e podem ser utilizadas para vários propósitos.

É importante manter em mente que as `Images` podem ser manipuladas somente através de `shared_ptr` e `weak_ptr`, o que permite que estes objetos sejam compartilhados entre vários objetos diferentes e ainda assim mantendo o grau de importância relativo a cada um.

A classe `Image` necessita aplicar a regra dos 5 em C++, efetuando a deleção dos seguintes métodos:

1. O construtor padrão que permite a criação de objetos resetados;
2. O construtor de cópia que permite copiar outros objetos do mesmo tipo;
3. O construtor de movimento que permite incorporar outros objetos através da `std::move`;
4. O operador de atribuição que permite copiar outros objetos do mesmo tipo;
5. O operador de atribuição que permite incorporar outros objetos através da `std::move`.

Image

Version 1.0 Proposed

ATTRIBUTES

◆ `image : VkImage Private`

O atributo que guarda a handle do objeto do tipo `VkImage` que representa o recurso dentro da API Vulkan.

◆ `sharingMode : VkSharingMode Private`

O atributo que armazena o modo de compartilhamento de um `Image`, ou seja, se ele é exclusivo ou concorrente.

◆ `usage : VkImageUsageFlags Private`

O atributo que contém o uso para o qual este `Image` é destinado.

◆ `type : VkImageType Private`

O atributo que armazena se trata-se de um `Image` 1D, 2D ou 3D.

◆ `format : VkFormat Private`

O atributo que armazena o formato referente aos texels do `Image`.

ATTRIBUTES
◆ extent : VkExtent3D Private O atributo que guarda o tamanho do Image, determinado por largura, altura e profundidade e, deve-se especificar os tamanhos em texels.
◆ samples : VkSampleCountFlagBits Private O atributo que armazena a intensidade do antisserilhamento que deve ser aplicado à este Image.
◆ tiling : VkImageTiling Private O atributo que guarda se deve ser feito uma disposição otimizada ou linear do Image na memória de vídeo.
◆ layout : VkImageLayout Private O atributo que determina o layout no qual o Image se encontra atualmente, este será alterado durante as operações.
◆ mipLevels : uint32_t Private O atributo que determina quantos mipmap terá este Image.
◆ arrayLayers : uint32_t Private O atributo que armazena quantos valores estarão presentes na lista de Image, só pode ser utilizado com Images de tipo 1D ou 2D, caso contrário deve ser 1.
◆ memory : std::unique_ptr<Memory> Private O atributo que guarda o unique_ptr da memória de vídeo associada à este Image.
◆ queueList : std::vector<std::weak_ptr<Queue>> Private O atributo que lista as múltiplas filas de processamento que poderão utilizar este Image, se ele for compartilhado.

OPERATIONS
◆ allocateMemory () : void Private Este método auxiliar tem como função efetivar a alocação de memória de vídeo para sustentar a image, através de um Allocator apropriado, e de efetuar a ligação entre a image e tal região de memória. Após a execução deste método, que deve acontecer nos métodos que criam a image, este objeto terá memória reservada no dispositivo físico, onde poderá realizar múltiplas operações.

◆ getGraphicsDevice () : VkDevice Private O método auxiliar getGraphicsDevice tem como objetivo adquirir o dispositivo lógico da aplicação através da API Vulkan.
--

Para atingir tal objetivo, o método obtém uma referência para o singleton de GraphicsManager e o utiliza para acessar o objeto de tipo Device. Por fim, requisita a handle do VkDevice diretamente ao objeto responsável por administrar o dispositivo lógico na Real Engine.

O método retornará o dispositivo lógico caso encontrado, senão irá retornar um código de erro que providencie maiores informações sobre o problema encontrado.

◆ `getImageCreateInfo () : VkImageCreateInfo Private`

O método auxiliar que tem como função criar e preencher a estrutura do tipo `VkImageCreateInfo`. Esta estrutura é necessária para requisitar a criação de um objeto `VkImage` ao dispositivo lógico, através da API Vulkan.

◆ `getVulkanImage () : Result<VkImage> Public`

Este método tem como objetivo permitir a obtenção da handle ao objeto do tipo `VkImage` para que outros objetos possam realizar operações relacionadas à API Vulkan que necessitem utilizar a handle.

O método irá retornar a handle em um objeto do tipo `Result` caso ela exista, senão, irá retornar um código de erro no objeto.

◆ `Image () : Private`

O construtor padrão e privado de objetos do tipo `Image`, seu único objetivo é criar um objeto com seus atributos completamente resetados.

Observação Importante: Para construir este objeto corretamente, deve-se utilizar para construção os seguintes métodos: `createlImage` ou `createSharedImage`.

◆ `~Image () : Public`

O destrutor padrão de `Image`, cujo objetivo é retornar a memória associada para o alocador que a providenciou, além de destruir o recurso de tipo `VkImage` junto à API Vulkan, nulificando, portanto, o atributo que armazena o handle: `image`.

◆ `createlImage (extent : VkExtent3D , type : VkImageType , arrayLayers : uint32_t , mipLevels : uint32_t , usage : VkImageUsageFlags , format : VkFormat , tiling : VkImageTiling) : std::shared_ptr<Image> Public`

O método `createlImage` é o que permite a criação de objetos do tipo `Image` que sejam exclusivos, ou seja, não precisam ser compartilhados entre múltiplas filas de processamento gráfico e, portanto, não necessitará receber uma estrutura que contenha uma lista de objetos do tipo `Queue`.

Este método retorna um `Image` criado a partir dos parâmetros especificados, realiza a alocação de memória necessária e efetua a ligação entre o `Image` e a região de memória alocada. Por fim, retorna o `Image` através de um `shared_ptr`, permitindo o compartilhamento de objetos do tipo `Image` entre múltiplos objetos enquanto evita leaks de memória do dispositivo físico.

◆ `createSharedImage (extent : VkExtent3D , type : VkImageType , arrayLayers : uint32_t , mipLevels : uint32_t , usage : VkImageUsageFlags , format : VkFormat , tiling : VkImageTiling , queues : std::vector<std::weak_ptr<Queue>>&) : std::shared_ptr<Image> Public`

O método `createSharedImage` é o que permite a criação de objetos do tipo `Image` que sejam concorrentes, ou seja, que precisam ser compartilhados entre múltiplas filas de processamento gráfico e, portanto, necessitará receber uma estrutura que contenha uma lista de objetos do tipo `Queue`.

Este método retorna um Image criado a partir dos parâmetros especificados, realiza a alocação de memória necessária e efetua a ligação entre o Image e a região de memória alocada. Por fim, retorna o Image através de um shared_ptr, permitindo o compartilhamento de objetos do tipo Image entre múltiplos objetos enquanto evita leaks de memória do dispositivo físico.

Instance

Class in package 'Graphics'

A classe Instance é responsável por definir uma fina camada de abstração sobre a sua equivalente definida pela especificação Vulkan, VkInstance, permitindo uma criação simplificada do objeto que preenche por padrão informações relacionadas à Real Engine e seu funcionamento interno.

Essa classe deve ser inicializada e encerrada através dos métodos startup e shutdown, respectivamente, ou não irá efetuar a criação do objeto subliminar através das chamadas da API Vulkan, se houver tentativa de utilização do objeto neste estado, então um erro será propagado através de um objeto do tipo Result e que, portanto, deverá ser verificado pela presença de erros.

A classe Instance necessita aplicar a regra dos 5 em C++, efetuando a deleção dos seguintes métodos:

1. O construtor padrão que permite a criação de objetos resetados;
2. O construtor de cópia que permite copiar outros objetos do mesmo tipo;
3. O construtor de movimento que permite incorporar outros objetos através da std::move;
4. O operador de atribuição que permite copiar outros objetos do mesmo tipo;
5. O operador de atribuição que permite incorporar outros objetos através da std::move.

Instance
Version 1.0 Proposed

ATTRIBUTES

◆ applicationName : const char* Private

O atributo que define o nome da aplicação, este atributo é importante e deve corretamente identificar a família de aplicações ao qual pertence, visto que será usado pela API Vulkan para identificar melhorias presentes no driver.

◆ applicationVersion : uint32_t Private

A versão da aplicação que será passada a API Vulkan, este campo também será utilizado para detectar melhorias disponíveis nos drivers gráficos e recomenda-se que se gere esse atributo através da macro: VK_MAKE_VERSION.

◆ isDebug : bool Private

O atributo que determina se haverá a ativação das camadas Vulkan que são responsáveis por funções de logging, perfilamento e depuração.

◆ instance : VkInstance Private

O atributo que armazena a instância Vulkan criada através da chamada para a API. Este atributo só será setado após a execução do método startup e será limpo após a chamada do método shutdown.

OPERATIONS

◆ getApplicationInfo () : VkApplicationInfo Private

O método cujo objetivo é criar e completar a estrutura VkApplicationInfo que contém informações sobre a engine utilizada, a

OPERATIONS

aplicação e a versão da API Vulkan que está sendo utilizada.

Especificamente, as informações que serão preenchidas em sua totalidade são:

1. Nome da aplicação;
2. Versão da aplicação;
3. Nome da engine;
4. Versão da engine;
5. Versão da API Vulkan.

As versões devem ser geradas utilizando a macro da API Vulkan: VK_MAKE_VERSION, enquanto os nomes devem ser passados como arrays fixos de caracteres.

◆ getInstanceCreateInfo (applicationInfo : VkApplicationCreateInfo&) : VkInstanceCreateInfo Private

O método responsável por preencher a estrutura VkInstanceCreateInfo que determina as propriedades da instância Vulkan que será criada. Este método precisa receber a estrutura VkApplicationCreateInfo preenchida através de um parâmetro referencial para que seu endereço possa ser alimentado na estrutura.

A estrutura VkInstanceCreateInfo especifica para a API Vulkan as extensões e camadas a serem habilitadas para a instância, além das informações sobre a aplicação contidas em sua estrutura complementar VkApplicationCreateInfo.

As camadas a serem habilitadas por esse método irão divergir com base no atributo isDebug, onde se este estiver avaliado como verdadeiro, serão ligadas as camadas de logging, perfilamento e depuração da API Vulkan.

As extensões dependerão única e exclusivamente das capacidades mínimas da GPU e das necessidades mínimas da Real Engine, caso um meio termo não seja encontrado a aplicação não será executado e retornará um erro.

◆ getVulkanInstance () : Result<VkInstance> Public

O método cujo objetivo é retornar a handle do objeto VkInstance para que ele possa ser utilizado em outros objetos e em chamadas para a API Vulkan. Irá retornar um erro caso o método startup não tiver sido invocado anteriormente ou se o mesmo tiver retornado um erro que não foi tratado.

◆ Instance (appName : const char* , appVersion : uint32_t , debug : bool) : Public

O construtor principal de objetos do tipo Instance. Seu objetivo é armazenar as informações requisitadas nos parâmetros que são:

1. O nome da aplicação;
2. A versão atual da aplicação;
3. Se a aplicação está em modo de depuração ou não.

Observação Importante: Esse construtor não cria o objeto Vulkan do tipo VkInstance e, portanto, o objeto do tipo Instance não deve ser utilizado diretamente sem antes ser invocado o método startup. Além disso, ao terminar de utilizar este objeto, deve-se chamar o método shutdown para efetuar a limpeza do objeto na memória e na API Vulkan.

Também é importante manter em mente ao utilizar este objeto que o método shutdown deve ser invocado após o desligamento de outros objetos de nível de abstração mais elevado, como o Device. Mais especificamente, deve-se invocar o método shutdown do objeto Instance como o último dentre os objetos que abstraem a API Vulkan.

◆ shutdown () : void Public

O método com o propósito de destruir o objeto de tipo VkInstance na API Vulkan e limpar a memória. Esse método deve ser chamado e na ordem correta ou irá acarretar em problemas devido à destruição aleatória de objetos por parte do C++.

◊ startup () : Result<void> Public

O método responsável por criar a instância Vulkan propriamente dita. Construindo-se em cima dos métodos auxiliares `getApplicationInfo` e `getInstanceCreateInfo`, este método fornece todas as informações que a API Vulkan necessita para inicializar um objeto do tipo `VkInstance` e por fim, o método armazena a handle retornada no atributo `instance` do objeto de tipo `Instance`.

Memory

Class in package 'Graphics'

A classe Memory cria uma abstração simples sobre o objeto `VkDeviceMemory` da API Vulkan. A ideia básica desta classe é armazenar a handle para uma área de memória em particular e o offset da memória em questão, visto que várias regiões de memória irão compartilhar o mesmo handle para minimizar o número de alocações.

Uma constatação importante é que objetos do tipo Memory podem ser utilizados somente através de `unique_ptr`, o que garante que cada objeto seja exclusivo e tenha apenas um dono em um determinado momento.

A classe Memory necessita aplicar a regra dos 5 em C++, efetuando a deletação dos seguintes métodos:

1. O construtor padrão que permite a criação de objetos resetados;
2. O construtor de cópia que permite copiar outros objetos do mesmo tipo;
3. O construtor de movimento que permite incorporar outros objetos através da `std::move`;
4. O operador de atribuição que permite copiar outros objetos do mesmo tipo;
5. O operador de atribuição que permite incorporar outros objetos através da `std::move`.

Memory

Version 1.0 Proposed

ATTRIBUTES

◊ memory : `VkDeviceMemory` Private

O atributo que armazena a handle para a memória de vídeo diretamente. Este é o mesmo endereço do alocador que obteve este pedaço específico de memória.

◊ offset : `VkDeviceSize` Private

O offset que deve-se adicionar ao handle da memória de vídeo para chegar-se na região de memória representada por este objeto.

◊ heap : `uint32_t` Private

A heap do dispositivo físico em que esta região de memória está localizada.

OPERATIONS

◊ chooseHeapFromFlags (`memoryRequirements` : const `VkMemoryRequirements&` , `requiredFlags` : `VkMemoryPropertyFlags`) : `uint32_t` Public

Um método cujo objetivo é determinar a heap ideal da memória do dispositivo físico (GPU) na qual deve-se estabelecer um objeto que tenha dados requerimentos de memórias (`VkMemoryRequirements`) e necessidades específicas para essa memória, como localização compartilhada entre host e dispositivo, localização somente no dispositivo, etc. (`VkMemoryPropertyFlags`)

Este método é utilizado principalmente pelos alocadores quando precisam decidir em qual das várias heaps da GPU irão realizar a larga alocação de memória que estes objetos necessitam.

OPERATIONS

❖ createMemory (mem : VkDeviceMemory , off : VkDeviceSize , hp : uint32_t) : std::unique_ptr<Memory> Public

O método createMemory é a forma padrão de criar objetos do tipo Memory, configurando os atributos conforme os parâmetros recebidos pelo método e retornando um unique_ptr que impede a replicação do objeto.

❖ Memory () : Private

O construtor padrão de objetos Memory, sua única função é dar valores nulos aos atributos da classe. O construtor é privado de maneira que o objeto pode ser criado apenas através do método estático createMemory que retorna um unique_ptr, impedindo duplicações deste tipo de objeto.

MemoryManager

Class in package 'Graphics'

O MemoryManager é a classe que gerencia o subsistema de memória. Suas principais funcionalidades são as seguintes:

1. Gerenciar os diversos alocadores e seus tipos, sendo capaz de providenciar os alocadores adequados quando forem requisitados por outros objetos;
2. Manter as propriedades físicas da memória para que outros objetos, principalmente alocadores, possam consultá-los e decidir qual das heaps da GPU é mais adequada para se localizarem;
3. Vigiar alocadores que estejam sem realizar alocações há um determinado tempo e eliminá-los, liberando a memória de vídeo que haviam alocado para distribuição.

A classe MemoryManager necessita aplicar a regra dos 5 em C++, efetuando a deleção dos seguintes métodos:

1. O construtor padrão que permite a criação de objetos resetados;
2. O construtor de cópia que permite copiar outros objetos do mesmo tipo;
3. O construtor de movimento que permite incorporar outros objetos através da std::move;
4. O operador de atribuição que permite copiar outros objetos do mesmo tipo;
5. O operador de atribuição que permite incorporar outros objetos através da std::move.

MemoryManager
Version 1.0 Proposed

ATTRIBUTES

❖ memoryProperties : VkPhysicalDeviceMemoryProperties Private

O atributo que guarda as propriedades da memória do dispositivo físico escolhido para rodar a aplicação.

OPERATIONS

❖ MemoryManager () : Private

O construtor padrão de MemoryManager e que não pode ser utilizado. O seu único objetivo é resetar os valores dos atributos de tal maneira que eles possam ser setados apropriadamente no método startup.

❖ getPhysicalDevice () : VkPhysicalDevice Private

O método auxiliar getPhysicalDevice tem como objetivo adquirir o dispositivo físico da aplicação através da API Vulkan.

Para atingir tal objetivo, o método obtém uma referência para o singleton de GraphicsManager e o utiliza para acessar o objeto de tipo Device. Por fim, requisita a handle do VkPhysicalDevice diretamente ao objeto responsável por administrar o dispositivo lógico na Real Engine.

O método retornará o dispositivo físico caso encontrado, senão irá retornar um código de erro que providencie maiores informações sobre o problema encontrado.

◆ getPhysicalDeviceMemoryProperties () : void Private

O método getPhysicalDeviceMemoryProperties é um método auxiliar responsável por utilizar o dispositivo físico obtido através do GraphicsManager para requisitar as propriedades da memória deste dispositivo à API Vulkan.

◆ startup () : Result<void> Public

O método startup é fundamental e deve ser utilizado para inicializar o objeto do tipo MemoryManager antes de qualquer tentativa de utilizar alocadores da Real Engine.

◆ shutdown () : void Public

O método shutdown é importante para finalizar corretamente o uso da aplicação e deve ser chamado no fim da aplicação, após terem sido executadas todas as ações pendentes no dispositivo físico (GPU).

◆ getManager () : MemoryManager& Public

O método getManager tem como objetivo retornar uma referência para a instância única do MemoryManager, com a finalidade de que os seus métodos possam ser invocados de qualquer posição do código, até mesmo dentro de outros Managers.

Se a ocasião for a primeira vez em que o método getManager está sendo invocado, então o objeto de tipo MemoryManager será criado uma única vez e, antes de ser utilizado, deve ser corretamente inicializado através do método startup.

◆ getMemoryProperties () : VkPhysicalDeviceMemoryProperties Public

O método getMemoryProperties tem como função retornar a estrutura VkPhysicalDeviceMemoryProperties que descreve os tipos de memória e as heaps que existem no dispositivo físico escolhido pelo GraphicsManager para executar a aplicação.

PoolAllocator

Class in package 'Graphics'

O PoolAllocator é a classe do tipo de alocador mais fundamental presente na Real Engine. Devido ao fato da Real Engine funcionar em um sistema de tiles, em que todos os sprites ocuparão o mesmo espaço de memória, o PoolAllocator é a solução mais eficiente para distribuir memória de vídeo com grande eficiência e sem perigo de fragmentação.

Os PoolAllocators podem ser manejados somente através de shared_ptr e weak_ptr e devem ser criados e distribuídos pelo subsistema de memória de vídeo, ou seja, o MemoryManager.

A classe PoolAllocator necessita aplicar a regra dos 5 em C++, efetuando a deleção dos seguintes métodos:

1. O construtor padrão que permite a criação de objetos resetados;
2. O construtor de cópia que permite copiar outros objetos do mesmo tipo;

3. O construtor de movimento que permite incorporar outros objetos através da std::move;
4. O operador de atribuição que permite copiar outros objetos do mesmo tipo;
5. O operador de atribuição que permite incorporar outros objetos através da std::move.

PoolAllocator
Version 1.0 Proposed

ATTRIBUTES

◆ alignment : VkDeviceSize Private

O atributo que guarda o alinhamento da memória do PoolAllocator, importante para que as operações sejam realizadas em velocidade máxima.

◆ chunkSize : VkDeviceSize Private

O atributo que determina o tamanho de cada uma das regiões de memória que serão distribuídas.

◆ flags : VkMemoryPropertyFlags Private

O atributo que guarda as propriedades da zona de memória alocada.

◆ freeList : std::forward_list<std::unique_ptr<Memory>> Private

A lista linkada que armazena as regiões de memória que estão disponíveis para serem alocadas como unique_ptr sob a autoridade do alocador.

◆ heap : uint32_t Private

O atributo que guarda em qual heap da memória do dispositivo físico a memória do PoolAllocator está armazenada.

◆ memory : VkDeviceMemory Private

O atributo que armazena o handle da memória do PoolAllocator, esta é a origem de todas as regiões de memória que serão distribuídas pelo objeto.

◆ size : VkDeviceSize Private

O atributo que guarda o tamanho do bloco de memória que o PoolAllocator alocou. Ao dividir-se o size pelo chunkSize se tem o número de blocos de memória que podem ser distribuídos pelo alocador.

OPERATIONS

◆ PoolAllocator () : Private

O construtor padrão e privado de objetos do tipo PoolAllocator. O objetivo deste construtor é criar um objeto resetado com todos os seus atributos setados para valores padrões.

◆ chunkMemory () : void Private

Um método auxiliar cujo propósito está em partitionar a extensa fatia de memória que foi alocada durante a criação do objeto. Este método irá fatiar a memória de tamanho size em (chunkSize / size) pedaços de memória de tamanho chunkSize.

Cada pedaço de memória fatiado pelo método será representado por uma estrutura do tipo Memory, manipulada através de um unique_ptr garantindo que apenas uma cópia do objeto possa existir. Estes vários pedaços serão então utilizados para alimentar a lista linkada freeList como memória disponível para o PoolAllocator alocar à objetos que requisitarem.

◆ getGraphicsDevice () : VkDevice Private

O método auxiliar getGraphicsDevice tem como objetivo adquirir o dispositivo lógico da aplicação através da API Vulkan.

Para atingir tal objetivo, o método obtém uma referência para o singleton de GraphicsManager e o utiliza para acessar o objeto de tipo Device. Por fim, requisita a handle do VkDevice diretamente ao objeto responsável por administrar o dispositivo lógico na Real Engine.

O método retornará o dispositivo lógico caso encontrado, senão irá retornar um código de erro que providencie maiores informações sobre o problema encontrado.

◆ getMemoryAllocateInfo () : VkMemoryAllocateInfo Private

O método auxiliar que tem como função criar e preencher a estrutura do tipo VkMemoryAllocateInfo. Esta estrutura é necessária para requisitar uma alocação de memória ao dispositivo físico, através da API Vulkan.

◆ getMemoryRequirements () : VkMemoryRequirements Private

O método auxiliar que tem como função criar e preencher a estrutura do tipo VkMemoryRequirements. Esta estrutura é necessária para escolher a heap correta na qual a memória deste PoolAllocator deve se localizar ao chamar o método estático de chooseHeapFromFlags.

◆ ~PoolAllocator () : Public

O destrutor de objetos do tipo PoolAllocator. Este é um método fundamental destes objetos, visto que é responsável por dealocar a memória junto a API Vulkan e também limpar a lista de memória disponível.

É importante notar que, como objetos do tipo PoolAllocator são manipulados por shared_ptr, este destrutor será chamado se e, somente se, houver a destruição de todos os shared_ptr que referenciam este objeto.

Deve-se manter em mente que os objetos do tipo weak_ptr que referenciarem este objeto não impedirão sua destruição, no entanto, a partir do momento em que ele for destruído perderão suas referências.

◆ getAllocatorAlignment () : VkDeviceSize Public

Este método tem como objetivo simplesmente retornar o valor de alinhamento da memória neste alocador. O alinhamento está guardado no atributo alignment.

◆ allocate (siz : VkDeviceSize) : std::unique_ptr<Memory> Public

O método que os alocadores necessitam para que outros objetos possam efetuar requisições de alocações de espaços de memória de vídeo.

A memória é retornada na forma de um unique_ptr de forma a manter garantida a exclusividade do objeto Memory e de evitar leaks de memória de vídeo.

Por se tratar de um PoolAllocator, este método retornará o primeiro elemento da lista linkada freeList, removendo-o da lista

e transferindo a propriedade do objeto do tipo Memory para o requisitante. Este pedaço de memória transferido terá tamanho igual a chunkSize, portanto, o parâmetro do método é ignorado.

◆ free (mem : std::unique_ptr<Memory>&) : void Public

O método que os alocadores necessitam para que outros objetos possam liberar alocações de memória de vídeo que haviam feito anteriormente.

A memória é tomada na forma de uma referência de unique_ptr e deverá nulificar esse ponteiro além de retornar o objeto Memory para a propriedade do alocador.

No PoolAllocator, este método meramente tomará a propriedade do objeto que lhe foi encaminhado e o adicionará na frente da lista linkada freeList.

◆ createAllocator (initialSize : VkDeviceSize, partitionSize : VkDeviceSize, alignment : VkDeviceSize, flags : VkMemoryPropertyFlags) : std::shared_ptr<PoolAllocator> Public

Este método estático é a forma correta e padrão de efetuar a criação de um objeto do tipo PoolAllocator.

O método criará um PoolAllocator que conforme com as propriedades passadas nos parâmetros, alocará a memória com a API Vulkan e, por fim, retornará um shared_ptr que permite compartilhar o objeto entre vários outros objetos que necessitem utilizá-lo.

Queue

Class in package 'Graphics'

A classe Queue é uma abstração sobre o objeto VkQueue da API Vulkan. Sua funcionalidade básica é submeter comandos à respectiva fila que representa e, portanto, tais objetos serão usados frequentemente em conjunto com o Escalonador (Scheduler) enquanto este decide em qual das filas é apropriado realizar as operações a ele requisitadas.

A classe Queue necessita aplicar a regra dos 5 em C++, efetuando a deleção dos seguintes métodos:

1. O construtor padrão que permite a criação de objetos resetados;
2. O construtor de cópia que permite copiar outros objetos do mesmo tipo;
3. O construtor de movimento que permite incorporar outros objetos através da std::move;
4. O operador de atribuição que permite copiar outros objetos do mesmo tipo;
5. O operador de atribuição que permite incorporar outros objetos através da std::move.

Queue
Version 1.0 Proposed

Result

Class in package 'Graphics'

A classe Result existe para permitir o retorno de valores de métodos e funções que nem sempre estarão presentes devido a erros na execução destas. Logo, objetos do tipo Result devem ser verificados pela presença de erros, caso estes não estejam presentes então deve-se proceder com a leitura do valor retornado da função, do contrário deve-se realizar os passos corretos para prosseguir com a execução sem aquele valor.

É importante notar que caso seja efetuada uma tentativa de acessar um valor em um objeto do tipo Result que esteja armazenando um erro proveniente da função ou método invocado, a aplicação levantará uma exception.

Result
Version 1.0 Proposed

ATTRIBUTES

◆ error : bool Private

O atributo que indica se há um erro ou não armazenado no objeto do tipo Result.

◆ errorCode : ErrorCode Private

O atributo que indica qual o erro que está armazenado, se tiver algum.

◆ value : T Private

O atributo que armazena o valor retornado de algum método se este retornar corretamente, deve ser ignorado se houver erro no objeto.

OPERATIONS

◆ Result (err : ErrorCode) : Private

Um construtor privado que permite criar um objeto do tipo Result a partir de um código de erro, esse construtor é utilizado pelo método error que deve ser utilizado pelos métodos que retornarem objetos do tipo Result para gerarem erros com o código apropriado.

◆ Result (val : T) : Public

Um construtor que permite criar um objeto do tipo Result a partir do valor correto a ser retornado, neste caso não haverá presença de erro e o objeto armazenado será retornado corretamente.

◆ error (err : ErrorCode) : Result Public

Um método estático que permite criar um objeto do tipo Result com um código de erro especificado. Deve ser utilizado pelos métodos para suas condições de erro.

◆ getError () : ErrorCode Public

O método cujo objetivo é retornar o código de erro que está armazenado no objeto. Útil para depuração e logging.

◆ hasError () : bool Public

O método que é utilizado para verificar se há um erro no objeto do tipo Result. O seu retorno deve ser verificado antes de se tentar extrair o objeto retornado do método e será verdadeiro se houver erro.

◆ operator= (val : const T&) : Result<T>& Public

O operador que permite a criação de objetos do tipo Result a partir do operador de atribuição sendo igualado a algum valor desejado de retorno e, portanto, sem erro.

❖ operator T () : Public

O operador que permite a conversão automática entre o tipo de retorno e o objeto Result. Se não houver erro retornará corretamente o objeto armazenado, caso contrário, irá jogar uma exception reclamando sobre a tentativa de extrair valores de uma Result com erros.

IAllocator

Interface in package 'Graphics'

A interface IAllocator tem como objetivo fornecer um conjunto de métodos simples que todos os alocadores precisam ter, sejam eles StackAllocators, PoolAllocators ou outros. Através desta interface, outros objetos poderão armazenar ou receber alocadores sem necessitarem saber qual o tipo específico que lhes foi encaminhado.

IAllocator

Version 1.0 Proposed

OPERATIONS

❖ IAllocator () : Public

Um construtor padrão da interface que serve apenas para permitir a criação correta de seus descendentes. Um objeto do tipo IAllocator não poderá ser criado devido a virtualidade pura de seus métodos.

❖ allocate (siz : VkDeviceSize) : std::unique_ptr<Memory> Public

O método que os alocadores necessitam para que outros objetos possam efetuar requisições de alocações de espaços de memória de vídeo.

A memória é retornada na forma de um unique_ptr de forma a manter garantida a exclusividade do objeto Memory e de evitar leaks de memória de vídeo.

❖ free (mem : std::unique_ptr<Memory>&) : void Public

O método que os alocadores necessitam para que outros objetos possam liberar alocações de memória de vídeo que haviam feito anteriormente.

A memória é tomada na forma de uma referência de unique_ptr e deverá nulificar esse ponteiro além de retornar o objeto Memory para a propriedade do alocador.

ErrorCode

Enumeration in package 'Graphics'

O ErrorCode é uma enumeração que contém os diversos tipos de erros que podem ser retornados de métodos e funções através de objetos do tipo Result. Os erros contidos aqui podem ocorrer em vários dos objetos espalhados através da Real Engine e, a respectiva forma de lidar com eles ou reportá-los ao usuário/desenvolvedor está contida em outro objeto que é responsável por medidas de depuração, este sendo o DebugManager.

ErrorCode

Version 1.0 Proposed

