vengine

0.1.0

Generated by Doxygen 1.11.0

Chapter 1

vengine

1.1 Description

ACTUALLY WORKING ON IT!

Welcome to **VEngine**, a graphics engine developed with Vulkan. This project aims to provide a robust foundation for game and application developers, focusing on the performance and flexibility offered by Vulkan.

1.2 Prerequisites

- `CMake 3.27`
- `C++20`
- `Vulkan`
- `GLM`
- `assimp` (unused ATM)

1.3 Usage

1.3.1 Build

```
$> ./build.sh build
[...]
```

> This script also handle several other commands: `clean`, `format` and `doc`.

1.3.2 Run

```
$> ./vengine
[...]
```

1.3.3 Documentation

The documentation is generated using Doxygen. You can vizualize it on GitHub Pages.

1.4 Commit Norms

Commit Type	Description
build	Changes that affect the build system or external dependencies (npm, make, etc.)
ci	Changes related to integration files and scripts or configuration (Travis, Ansible, BrowserStack, etc.)
feat	Addition of a new feature
fix	Bug fix
perf	Performance improvements
refactor	Modification that neither adds a new feature nor improves performance
style	Change that does not affect functionality or semantics (indentation, formatting, adding space, renaming a variable, etc.)
docs	Writing or updating documentation
test	Addition or modification of tests

1.5 License

1.6 Acknowledgements

Thanks to Brendan Galea.

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 myLib Namespace Reference

Classes

- class Clock

  Class for time management.
- class Random

  Class for random number generation.
- class Time

  Class used for time management.

5.2 std Namespace Reference

STL namespace.

Classes

- class **allocator**

  STL class.
- class **array**

  STL class.
- class **atomic**

  STL class.
- class **atomic_ref**

  STL class.
- class **auto_ptr**

  STL class.
- class **bad_alloc**

  STL class.
- class **bad_cast**

  STL class.
- class **bad_exception**

STL class.

- class **bad_typeid**

  STL class.
- class **basic_fstream**

  STL class.
- class **basic_ifstream**

  STL class.
- class **basic_ios**

  STL class.
- class **basic_iostream**

  STL class.
- class **basic_istream**

  STL class.
- class **basic_istringstream**

  STL class.
- class **basic_ofstream**

  STL class.
- class **basic_ostream**

  STL class.
- class **basic_ostringstream**

  STL class.
- class **basic_string**

  STL class.
- class **basic_string_view**

  STL class.
- class **basic_stringstream**

  STL class.
- class **bitset**

  STL class.
- class **complex**

  STL class.
- class **deque**

  STL class.
- class **domain_error**

  STL class.
- class **error_category**

  STL class.
- class **error_code**

  STL class.
- class **error_condition**

  STL class.
- class **exception**

  STL class.
- class **forward_list**

  STL class.
- class **fstream**

  STL class.
- struct hash< ven::Model::Vertex >
- class **ifstream**

  STL class.
- class **invalid_argument**

STL class.

- class **ios**

  STL class.
- class **ios_base**

  STL class.
- class **istream**

  STL class.
- class **istringstream**

  STL class.
- class **jthread**

  STL class.
- class **length_error**

  STL class.
- class **list**

  STL class.
- class **lock_guard**

  STL class.
- class **logic_error**

  STL class.
- class **map**

  STL class.
- class **multimap**

  STL class.
- class **multiset**

  STL class.
- class **mutex**

  STL class.
- class **ofstream**

  STL class.
- class **ostream**

  STL class.
- class **ostringstream**

  STL class.
- class **out_of_range**

  STL class.
- class **overflow_error**

  STL class.
- class **pair**

  STL class.
- class **priority_queue**

  STL class.
- class **queue**

  STL class.
- class **range_error**

  STL class.
- class **recursive_mutex**

  STL class.
- class **recursive_timed_mutex**

  STL class.
- class **runtime_error**

  STL class.

- class **set**
  STL class.
- class **shared_lock**
  STL class.
- class **shared_mutex**
  STL class.
- class **shared_ptr**
  STL class.
- class **shared_timed_mutex**
  STL class.
- class **smart_ptr**
  STL class.
- class **span**
  STL class.
- class **stack**
  STL class.
- class **string**
  STL class.
- class **string_view**
  STL class.
- class **stringstream**
  STL class.
- class **system_error**
  STL class.
- class **thread**
  STL class.
- class **timed_mutex**
  STL class.
- class **u16string**
  STL class.
- class **u16string_view**
  STL class.
- class **u32string**
  STL class.
- class **u32string_view**
  STL class.
- class **u8string**
  STL class.
- class **u8string_view**
  STL class.
- class **underflow_error**
  STL class.
- class **unique_lock**
  STL class.
- class **unique_ptr**
  STL class.
- class **unordered_map**
  STL class.
- class **unordered_multimap**
  STL class.
- class **unordered_multiset**

STL class.

- class **unordered_set**

  STL class.
- class **valarray**

  STL class.
- class **vector**

  STL class.
- class **weak_ptr**

  STL class.
- class **wfstream**

  STL class.
- class **wifstream**

  STL class.
- class **wios**

  STL class.
- class **wistream**

  STL class.
- class **wistringstream**

  STL class.
- class **wofstream**

  STL class.
- class **wostream**

  STL class.
- class **wostringstream**

  STL class.
- class **wstring**

  STL class.
- class **wstring_view**

  STL class.
- class **wstringstream**

  STL class.

5.2.1 Detailed Description

STL namespace.

5.3 ven Namespace Reference

Classes

- class Buffer

  Class for buffer.
- class Camera
- class DescriptorPool

  Class for descriptor pool.
- class DescriptorSetLayout

  Class for descriptor set layout.
- class DescriptorWriter

Class for descriptor writer.

- class Device
- class Engine
- class FrameCounter
- struct FrameInfo
- struct GlobalUbo
- class KeyboardController
- class Model
- class Object
- struct PipelineConfigInfo
- struct PointLight
- struct PointLightComponent
- class PointLightSystem

  Class for point light system.
- struct QueueFamilyIndices
- class Renderer
- class RenderSystem

  Class for render system.
- class Shaders
- struct SimplePushConstantData
- class SwapChain
- struct SwapChainSupportDetails
- struct Transform3DComponent
- class Window

Typedefs

- using return_type_t
- using id_t = unsigned int

Functions

- template<typename T , typename... Rest>
  void hashCombine (std::size_t &seed, const T &v, const Rest &... rest)

Variables

- static constexpr uint32_t DEFAULT_WIDTH = 1920
- static constexpr uint32_t DEFAULT_HEIGHT = 1080
- static constexpr std::string_view DEFAULT_TITLE = "VEngine"
- static constexpr std::string_view SHADERS_BIN_PATH = "shaders/bin/"
- static constexpr std::size_t MAX_LIGHTS = 10

5.3.1 Typedef Documentation

5.3.1.1 id_t

```
using ven::id_t = unsigned int
```

Definition at line 18 of file Object.hpp.

5.3.1.2 return_type_t

```
using ven::return_type_t
```

Initial value:

```
enum ReturnType : uint8_t {
        VEN_SUCCESS = 0,
        VEN_FAILURE = 1
    }
```

Definition at line 17 of file Constant.hpp.

5.3.2 Function Documentation

5.3.2.1 hashCombine()

```
template<typename T , typename...  Rest>
void ven::hashCombine (
             std::size_t & seed,
             const T & v,
             const Rest &...  rest)
```

Definition at line 14 of file Utils.hpp.

References hashCombine().

Referenced by hashCombine(), and std::hash< ven::Model::Vertex >::operator()().

Here is the call graph for this function: Here is the caller graph for this function:

5.3.3 Variable Documentation

5.3.3.1 DEFAULT_HEIGHT

```
uint32_t ven::DEFAULT_HEIGHT = 1080  [static], [constexpr]
```

Definition at line 12 of file Constant.hpp.

5.3.3.2 DEFAULT_TITLE

```
std::string_view ven::DEFAULT_TITLE = "VEngine"  [static], [constexpr]
```

Definition at line 14 of file Constant.hpp.

5.3.3.3 DEFAULT_WIDTH

```
uint32_t ven::DEFAULT_WIDTH = 1920  [static], [constexpr]
```

Definition at line 11 of file Constant.hpp.

5.3.3.4 MAX_LIGHTS

```
std::size_t ven::MAX_LIGHTS = 10  [static], [constexpr]
```

Definition at line 16 of file FrameInfo.hpp.

Referenced by ven::PointLightSystem::update().

5.3.3.5 SHADERS_BIN_PATH

```
std::string_view ven::SHADERS_BIN_PATH = "shaders/bin/"  [static], [constexpr]
```

Definition at line 15 of file Constant.hpp.

Referenced by ven::PointLightSystem::createPipeline(), and ven::RenderSystem::createPipeline().

Chapter 6

Class Documentation

6.1 ven::Buffer Class Reference

Class for buffer.

`#include <Buffer.hpp>`

Collaboration diagram for ven::Buffer:

Public Member Functions

- Buffer (Device &device, VkDeviceSize instanceSize, uint32_t instanceCount, VkBufferUsageFlags usage↩ Flags, VkMemoryPropertyFlags memoryPropertyFlags, VkDeviceSize minOffsetAlignment=1)
- ∼Buffer ()
- Buffer (const Buffer &)=delete
- Buffer & operator= (const Buffer &)=delete
- VkResult map (VkDeviceSize size=VK_WHOLE_SIZE, VkDeviceSize offset=0)

  Map a memory range of this buffer.
- void unmap ()

  Unmap a mapped memory range.
- void writeToBuffer (const void ∗data, VkDeviceSize size=VK_WHOLE_SIZE, VkDeviceSize offset=0) const

  Copies the specified data to the mapped buffer.
- VkResult flush (VkDeviceSize size=VK_WHOLE_SIZE, VkDeviceSize offset=0) const

  Flush a memory range of the buffer to make it visible to the device.
- VkDescriptorBufferInfo descriptorInfo (const VkDeviceSize size=VK_WHOLE_SIZE, const VkDeviceSize offset=0) const

  Create a buffer info descriptor.
- VkResult invalidate (VkDeviceSize size=VK_WHOLE_SIZE, VkDeviceSize offset=0) const

  Invalidate a memory range of the buffer to make it visible to the host.
- void writeToIndex (const void ∗data, const VkDeviceSize index) const

  Copies "instanceSize" bytes of data to the mapped buffer at an offset of index ∗ alignmentSize.
- VkResult flushIndex (const VkDeviceSize index) const

  Flush the memory range at index ∗ alignmentSize of the buffer to make it visible to the device.
- VkDescriptorBufferInfo descriptorInfoForIndex (const VkDeviceSize index) const

  Create a buffer info descriptor.
- VkResult invalidateIndex (const VkDeviceSize index) const

Invalidate a memory range of the buffer to make it visible to the host.

- VkBuffer getBuffer () const
- void * getMappedMemory () const
- uint32_t getInstanceCount () const
- VkDeviceSize getInstanceSize () const
- VkDeviceSize getAlignmentSize () const
- VkBufferUsageFlags getUsageFlags () const
- VkMemoryPropertyFlags getMemoryPropertyFlags () const
- VkDeviceSize getBufferSize () const

Static Private Member Functions

- static VkDeviceSize getAlignment (VkDeviceSize instanceSize, VkDeviceSize minOffsetAlignment)

  Returns the minimum instance size required to be compatible with devices minOffsetAlignment.

Private Attributes

- Device & m_device
- void * m_mapped = nullptr
- VkBuffer m_buffer = VK_NULL_HANDLE
- VkDeviceMemory m_memory = VK_NULL_HANDLE
- VkDeviceSize m_bufferSize
- VkDeviceSize m_instanceSize
- uint32_t m_instanceCount
- VkDeviceSize m_alignmentSize
- VkBufferUsageFlags m_usageFlags
- VkMemoryPropertyFlags m_memoryPropertyFlags

6.1.1 Detailed Description

Class for buffer.

Definition at line 17 of file Buffer.hpp.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 Buffer() [1/2]

```
ven::Buffer::Buffer (
            Device & device,
            VkDeviceSize instanceSize,
            uint32_t instanceCount,
            VkBufferUsageFlags usageFlags,
            VkMemoryPropertyFlags memoryPropertyFlags,
            VkDeviceSize minOffsetAlignment = 1)
```

Definition at line 13 of file buffer.cpp.

References ven::Device::createBuffer(), m_alignmentSize, m_buffer, m_bufferSize, m_instanceCount, m_memory, m_memoryPropertyFlags, and m_usageFlags.

Here is the call graph for this function:

6.1.2.2 ~Buffer()

```
ven::Buffer::~Buffer ()
```

Definition at line 19 of file buffer.cpp.

6.1.2.3 Buffer() [2/2]

```
ven::Buffer::Buffer (
            const Buffer & )  [delete]
```

6.1.3 Member Function Documentation

6.1.3.1 descriptorInfo()

```
VkDescriptorBufferInfo ven::Buffer::descriptorInfo (
            const VkDeviceSize size = VK_WHOLE_SIZE,
            const VkDeviceSize offset = 0) const  [inline], [nodiscard]
```

Create a buffer info descriptor.

Parameters

size	(Optional) Size of the memory range of the descriptor
offset	(Optional) Byte offset from beginning

Returns

VkDescriptorBufferInfo of specified offset and range

Definition at line 73 of file Buffer.hpp.

References m_buffer.

Referenced by descriptorInfoForIndex().

Here is the caller graph for this function:

6.1.3.2 descriptorInfoForIndex()

```
VkDescriptorBufferInfo ven::Buffer::descriptorInfoForIndex (
            const VkDeviceSize index) const  [inline], [nodiscard]
```

Create a buffer info descriptor.

Parameters

index	Specifies the region given by index ∗ alignmentSize

Returns

VkDescriptorBufferInfo for instance at index

Definition at line 112 of file Buffer.hpp.

References descriptorInfo(), and m_alignmentSize.

Here is the call graph for this function:

6.1.3.3 flush()

```
VkResult ven::Buffer::flush (
            VkDeviceSize size = VK_WHOLE_SIZE,
            VkDeviceSize offset = 0) const  [nodiscard]
```

Flush a memory range of the buffer to make it visible to the device.

Note

Only required for non-coherent memory

Parameters

size	(Optional) Size of the memory range to flush. Pass VK_WHOLE_SIZE to flush the complete buffer range.
offset	(Optional) Byte offset from beginning

Returns

VkResult of the flush call

Definition at line 53 of file buffer.cpp.

Referenced by flushIndex().

Here is the caller graph for this function:

6.1.3.4 flushIndex()

```
VkResult ven::Buffer::flushIndex (
            const VkDeviceSize index) const  [inline], [nodiscard]
```

Flush the memory range at index ∗ alignmentSize of the buffer to make it visible to the device.

Parameters

index	Used in offset calculation

Definition at line 102 of file Buffer.hpp.

References flush(), and m_alignmentSize.

Here is the call graph for this function:

6.1.3.5 getAlignment()

```
VkDeviceSize ven::Buffer::getAlignment (
            VkDeviceSize instanceSize,
            VkDeviceSize minOffsetAlignment)  [static], [private]
```

Returns the minimum instance size required to be compatible with devices minOffsetAlignment.

Parameters

instanceSize	The size of an instance
minOffsetAlignment	The minimum required alignment, in bytes, for the offset member (eg minUniformBufferOffsetAlignment)

Returns

VkResult of the buffer mapping call

Definition at line 6 of file buffer.cpp.

6.1.3.6 getAlignmentSize()

```
VkDeviceSize ven::Buffer::getAlignmentSize () const  [inline], [nodiscard]
```

Definition at line 129 of file Buffer.hpp.

References m_instanceSize.

6.1.3.7 getBuffer()

```
VkBuffer ven::Buffer::getBuffer () const  [inline], [nodiscard]
```

Definition at line 125 of file Buffer.hpp.

References m_buffer.

6.1.3.8 getBufferSize()

```
VkDeviceSize ven::Buffer::getBufferSize () const  [inline], [nodiscard]
```

Definition at line 132 of file Buffer.hpp.

References m_bufferSize.

6.1.3.9 getInstanceCount()

```
uint32_t ven::Buffer::getInstanceCount () const  [inline], [nodiscard]
```

Definition at line 127 of file Buffer.hpp.

References m_instanceCount.

6.1.3.10 getInstanceSize()

```
VkDeviceSize ven::Buffer::getInstanceSize () const  [inline], [nodiscard]
```

Definition at line 128 of file Buffer.hpp.

References m_instanceSize.

6.1.3.11 getMappedMemory()

```
void * ven::Buffer::getMappedMemory () const  [inline], [nodiscard]
```

Definition at line 126 of file Buffer.hpp.

References m_mapped.

6.1.3.12 getMemoryPropertyFlags()

```
VkMemoryPropertyFlags ven::Buffer::getMemoryPropertyFlags () const  [inline], [nodiscard]
```

Definition at line 131 of file Buffer.hpp.

References m_memoryPropertyFlags.

6.1.3.13 getUsageFlags()

```
VkBufferUsageFlags ven::Buffer::getUsageFlags () const  [inline], [nodiscard]
```

Definition at line 130 of file Buffer.hpp.

References m_usageFlags.

6.1.3.14 invalidate()

```
VkResult ven::Buffer::invalidate (
            VkDeviceSize size = VK_WHOLE_SIZE,
            VkDeviceSize offset = 0) const  [nodiscard]
```

Invalidate a memory range of the buffer to make it visible to the host.

Note

Only required for non-coherent memory

Parameters

size	(Optional) Size of the memory range to invalidate. Pass VK_WHOLE_SIZE to invalidate the complete buffer range.
offset	(Optional) Byte offset from beginning

Returns

VkResult of the invalidate call

Definition at line 63 of file buffer.cpp.

Referenced by invalidateIndex().

Here is the caller graph for this function:

6.1.3.15 invalidateIndex()

```
VkResult ven::Buffer::invalidateIndex (
            const VkDeviceSize index) const  [inline], [nodiscard]
```

Invalidate a memory range of the buffer to make it visible to the host.

Note

Only required for non-coherent memory

Parameters

index	Specifies the region to invalidate: index ∗ alignmentSize

Returns

VkResult of the invalidate call

Definition at line 123 of file Buffer.hpp.

References invalidate(), and m_alignmentSize.

Here is the call graph for this function:

6.1.3.16 map()

```
VkResult ven::Buffer::map (
            VkDeviceSize size = VK_WHOLE_SIZE,
            VkDeviceSize offset = 0)
```

Map a memory range of this buffer.

If successful, mapped points to the specified buffer range.

Parameters

size	(Optional) Size of the memory range to map. Pass VK_WHOLE_SIZE to map the complete buffer range.
offset	(Optional) Byte offset from beginning

Returns

VkResult of the buffer mapping call

Definition at line 26 of file buffer.cpp.

Referenced by ven::Model::createIndexBuffer(), and ven::Model::createVertexBuffer().

Here is the caller graph for this function:

6.1.3.17 operator=()

```
Buffer & ven::Buffer::operator= (
            const Buffer & )  [delete]
```

6.1.3.18 unmap()

```
void ven::Buffer::unmap ()
```

Unmap a mapped memory range.

Note

Does not return a result as vkUnmapMemory can't fail

Definition at line 32 of file buffer.cpp.

6.1.3.19 writeToBuffer()

```
void ven::Buffer::writeToBuffer (
            const void * data,
            VkDeviceSize size = VK_WHOLE_SIZE,
            VkDeviceSize offset = 0) const
```

Copies the specified data to the mapped buffer.

Default value writes whole buffer range

Parameters

data	Pointer to the data to copy
size	(Optional) Size of the data to copy. Pass VK_WHOLE_SIZE to flush the complete buffer range.
offset	(Optional) Byte offset from beginning of mapped region

Definition at line 40 of file buffer.cpp.

Referenced by writeToIndex().

Here is the caller graph for this function:

6.1.3.20 writeToIndex()

```
void ven::Buffer::writeToIndex (
            const void * data,
            const VkDeviceSize index) const  [inline]
```

Copies "instanceSize" bytes of data to the mapped buffer at an offset of index * alignmentSize.

Parameters

data	Pointer to the data to copy
index	Used in offset calculation

Definition at line 95 of file Buffer.hpp.

References m_alignmentSize, m_instanceSize, and writeToBuffer().

Here is the call graph for this function:

6.1.4 Member Data Documentation

6.1.4.1 m_alignmentSize

```
VkDeviceSize ven::Buffer::m_alignmentSize  [private]
```

Definition at line 154 of file Buffer.hpp.

Referenced by Buffer(), descriptorInfoForIndex(), flushIndex(), invalidateIndex(), and writeToIndex().

6.1.4.2 m_buffer

```
VkBuffer ven::Buffer::m_buffer = VK_NULL_HANDLE  [private]
```

Definition at line 148 of file Buffer.hpp.

Referenced by Buffer(), descriptorInfo(), and getBuffer().

6.1.4.3 m_bufferSize

```
VkDeviceSize ven::Buffer::m_bufferSize  [private]
```

Definition at line 151 of file Buffer.hpp.

Referenced by Buffer(), and getBufferSize().

6.1.4.4 m_device

```
Device& ven::Buffer::m_device  [private]
```

Definition at line 146 of file Buffer.hpp.

6.1.4.5 m_instanceCount

```
uint32_t ven::Buffer::m_instanceCount  [private]
```

Definition at line 153 of file Buffer.hpp.

Referenced by Buffer(), and getInstanceCount().

6.1.4.6 m_instanceSize

```
VkDeviceSize ven::Buffer::m_instanceSize  [private]
```

Definition at line 152 of file Buffer.hpp.

Referenced by getAlignmentSize(), getInstanceSize(), and writeToIndex().

6.1.4.7 m_mapped

```
void* ven::Buffer::m_mapped = nullptr  [private]
```

Definition at line 147 of file Buffer.hpp.

Referenced by getMappedMemory().

6.1.4.8 m_memory

```
VkDeviceMemory ven::Buffer::m_memory = VK_NULL_HANDLE  [private]
```

Definition at line 149 of file Buffer.hpp.

Referenced by Buffer().

6.1.4.9 m_memoryPropertyFlags

```
VkMemoryPropertyFlags ven::Buffer::m_memoryPropertyFlags  [private]
```

Definition at line 156 of file Buffer.hpp.

Referenced by Buffer(), and getMemoryPropertyFlags().

6.1.4.10 m_usageFlags

```
VkBufferUsageFlags ven::Buffer::m_usageFlags  [private]
```

Definition at line 155 of file Buffer.hpp.

Referenced by Buffer(), and getUsageFlags().

The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/Buffer.hpp
- /home/runner/work/VEngine/VEngine/src/buffer.cpp

6.2 ven::DescriptorPool::Builder Class Reference

```
#include <Descriptors.hpp>
```

Collaboration diagram for ven::DescriptorPool::Builder:

Public Member Functions

- Builder (Device &device)
- Builder & addPoolSize (VkDescriptorType descriptorType, uint32_t count)
- Builder & setPoolFlags (VkDescriptorPoolCreateFlags flags)
- Builder & setMaxSets (uint32_t count)
- std::unique_ptr< DescriptorPool > build () const

Private Attributes

- Device & m_device
- std::vector< VkDescriptorPoolSize > m_poolSizes
- uint32_t m_maxSets = 1000
- VkDescriptorPoolCreateFlags m_poolFlags = 0

6.2.1 Detailed Description

Definition at line 65 of file Descriptors.hpp.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 Builder()

```
ven::DescriptorPool::Builder::Builder (
            Device & device)  [inline], [explicit]
```

Definition at line 69 of file Descriptors.hpp.

6.2.3 Member Function Documentation

6.2.3.1 addPoolSize()

```
ven::DescriptorPool::Builder & ven::DescriptorPool::Builder::addPoolSize (
            VkDescriptorType descriptorType,
            uint32_t count)
```

Definition at line 39 of file descriptors.cpp.

Referenced by ven::Engine::Engine().

Here is the caller graph for this function:

6.2.3.2 build()

```
std::unique_ptr< DescriptorPool > ven::DescriptorPool::Builder::build () const  [inline],
[nodiscard]
```

Definition at line 74 of file Descriptors.hpp.

References m_device, m_maxSets, m_poolFlags, and m_poolSizes.

Referenced by ven::Engine::Engine().

Here is the caller graph for this function:

6.2.3.3 setMaxSets()

```
ven::DescriptorPool::Builder & ven::DescriptorPool::Builder::setMaxSets (
            uint32_t count)
```

Definition at line 50 of file descriptors.cpp.

Referenced by ven::Engine::Engine().

Here is the caller graph for this function:

6.2.3.4 setPoolFlags()

```
ven::DescriptorPool::Builder & ven::DescriptorPool::Builder::setPoolFlags (
            VkDescriptorPoolCreateFlags flags)
```

Definition at line 45 of file descriptors.cpp.

6.2.4 Member Data Documentation

6.2.4.1 m_device

```
Device& ven::DescriptorPool::Builder::m_device  [private]
```

Definition at line 78 of file Descriptors.hpp.

Referenced by build().

6.2.4.2 m_maxSets

```
uint32_t ven::DescriptorPool::Builder::m_maxSets = 1000  [private]
```

Definition at line 80 of file Descriptors.hpp.

Referenced by build().

6.2.4.3 m_poolFlags

```
VkDescriptorPoolCreateFlags ven::DescriptorPool::Builder::m_poolFlags = 0  [private]
```

Definition at line 81 of file Descriptors.hpp.

Referenced by build().

6.2.4.4 m_poolSizes

```
std::vector<VkDescriptorPoolSize> ven::DescriptorPool::Builder::m_poolSizes  [private]
```

Definition at line 79 of file Descriptors.hpp.

Referenced by build().

The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors.hpp
- /home/runner/work/VEngine/VEngine/src/descriptors.cpp

6.3 ven::DescriptorSetLayout::Builder Class Reference

```
#include <Descriptors.hpp>
```

Collaboration diagram for ven::DescriptorSetLayout::Builder:

Public Member Functions

- Builder (Device &device)
- Builder & addBinding (uint32_t binding, VkDescriptorType descriptorType, VkShaderStageFlags stageFlags, uint32_t count=1)
- std::unique_ptr< DescriptorSetLayout > build () const

Private Attributes

- Device & m_device
- std::unordered_map< uint32_t, VkDescriptorSetLayoutBinding > m_bindings

6.3.1 Detailed Description

Definition at line 25 of file Descriptors.hpp.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 Builder()

```
ven::DescriptorSetLayout::Builder::Builder (
            Device & device)  [inline], [explicit]
```

Definition at line 29 of file Descriptors.hpp.

6.3.3 Member Function Documentation

6.3.3.1 addBinding()

```
ven::DescriptorSetLayout::Builder & ven::DescriptorSetLayout::Builder::addBinding (
            uint32_t binding,
            VkDescriptorType descriptorType,
            VkShaderStageFlags stageFlags,
            uint32_t count = 1)
```

Definition at line 5 of file descriptors.cpp.

References m_bindings.

Referenced by ven::Engine::mainLoop().

Here is the caller graph for this function:

6.3.3.2 build()

```
std::unique_ptr< DescriptorSetLayout > ven::DescriptorSetLayout::Builder::build () const
[inline]
```

Definition at line 32 of file Descriptors.hpp.

References m_bindings, and m_device.

Referenced by ven::Engine::mainLoop().

Here is the caller graph for this function:

6.3.4 Member Data Documentation

6.3.4.1 m_bindings

```
std::unordered_map<uint32_t, VkDescriptorSetLayoutBinding> ven::DescriptorSetLayout::Builder↩
::m_bindings  [private]
```

Definition at line 36 of file Descriptors.hpp.

Referenced by addBinding(), and build().

6.3.4.2 m_device

```
Device& ven::DescriptorSetLayout::Builder::m_device  [private]
```

Definition at line 35 of file Descriptors.hpp.

Referenced by build().

The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors.hpp
- /home/runner/work/VEngine/VEngine/src/descriptors.cpp

6.4 ven::Model::Builder Struct Reference

`#include <Model.hpp>`

Collaboration diagram for ven::Model::Builder:

Public Member Functions

- void loadModel (const std::string &filename)

Public Attributes

- std::vector< Vertex > vertices
- std::vector< uint32_t > indices

6.4.1 Detailed Description

Definition at line 34 of file Model.hpp.

6.4.2 Member Function Documentation

6.4.2.1 loadModel()

```
void ven::Model::Builder::loadModel (
            const std::string & filename)
```

Definition at line 120 of file model.cpp.

References ven::Model::Vertex::position.

Referenced by ven::Model::createModelFromFile().

Here is the caller graph for this function:

6.4.3 Member Data Documentation

6.4.3.1 indices

```
std::vector<uint32_t> ven::Model::Builder::indices
```

Definition at line 36 of file Model.hpp.

Referenced by ven::Model::Model().

6.4.3.2 vertices

```
std::vector<Vertex> ven::Model::Builder::vertices
```

Definition at line 35 of file Model.hpp.

Referenced by ven::Model::Model().

The documentation for this struct was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp
- /home/runner/work/VEngine/VEngine/src/model.cpp

6.5 ven::Camera Class Reference

```
#include <Camera.hpp>
```

Collaboration diagram for ven::Camera:

Public Member Functions

- void setOrthographicProjection (float left, float right, float top, float bottom, float near, float far)
- void setPerspectiveProjection (float fovy, float aspect, float near, float far)
- void setViewDirection (glm::vec3 position, glm::vec3 direction, glm::vec3 up=glm::vec3{0.F, -1.F, 0.F})
- void setViewTarget (glm::vec3 position, glm::vec3 target, glm::vec3 up=glm::vec3{0.F, -1.F, 0.F})
- void setViewYXZ (glm::vec3 position, glm::vec3 rotation)
- const glm::mat4 & getProjection () const
- const glm::mat4 & getView () const
- const glm::mat4 & getInverseView () const

Private Attributes

- glm::mat4 m_projectionMatrix {1.F}
- glm::mat4 m_viewMatrix {1.F}
- glm::mat4 m_inverseViewMatrix {1.F}

6.5.1 Detailed Description

Definition at line 17 of file Camera.hpp.

6.5.2 Member Function Documentation

6.5.2.1 getInverseView()

```
const glm::mat4 & ven::Camera::getInverseView () const  [inline], [nodiscard]
```

Definition at line 29 of file Camera.hpp.

References m_inverseViewMatrix.

6.5.2.2 getProjection()

```
const glm::mat4 & ven::Camera::getProjection () const  [inline], [nodiscard]
```

Definition at line 27 of file Camera.hpp.

References m_projectionMatrix.

6.5.2.3 getView()

```
const glm::mat4 & ven::Camera::getView () const  [inline], [nodiscard]
```

Definition at line 28 of file Camera.hpp.

References m_viewMatrix.

6.5.2.4 setOrthographicProjection()

```
void ven::Camera::setOrthographicProjection (
            float left,
            float right,
            float top,
            float bottom,
            float near,
            float far)
```

Definition at line 6 of file camera.cpp.

References m_projectionMatrix.

6.5.2.5 setPerspectiveProjection()

```
void ven::Camera::setPerspectiveProjection (
            float fovy,
            float aspect,
            float near,
            float far)
```

Definition at line 17 of file camera.cpp.

6.5.2.6 setViewDirection()

```
void ven::Camera::setViewDirection (
            glm::vec3 position,
            glm::vec3 direction,
            glm::vec3 up = glm::vec3{0.F, -1.F, 0.F})
```

Definition at line 29 of file camera.cpp.

6.5.2.7 setViewTarget()

```
void ven::Camera::setViewTarget (
            glm::vec3 position,
            glm::vec3 target,
            glm::vec3 up = glm::vec3{0.F, -1.F, 0.F})  [inline]
```

Definition at line 24 of file Camera.hpp.

6.5.2.8 setViewYXZ()

```
void ven::Camera::setViewYXZ (
            glm::vec3 position,
            glm::vec3 rotation)
```

Definition at line 64 of file camera.cpp.

6.5.3 Member Data Documentation

6.5.3.1 m_inverseViewMatrix

```
glm::mat4 ven::Camera::m_inverseViewMatrix {1.F}  [private]
```

Definition at line 35 of file Camera.hpp.

Referenced by getInverseView().

6.5.3.2 m_projectionMatrix

```
glm::mat4 ven::Camera::m_projectionMatrix {1.F}  [private]
```

Definition at line 33 of file Camera.hpp.

Referenced by getProjection(), and setOrthographicProjection().

6.5.3.3 m_viewMatrix

```
glm::mat4 ven::Camera::m_viewMatrix {1.F}  [private]
```

Definition at line 34 of file Camera.hpp.

Referenced by getView().

The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/Camera.hpp
- /home/runner/work/VEngine/VEngine/src/camera.cpp

6.6 myLib::Clock Class Reference

Class for time management.

`#include <Clock.hpp>`

Collaboration diagram for myLib::Clock:

Public Member Functions

- Clock ()
- ~Clock ()=default
- void restart ()

  Restart the clock.
- void pause ()

  Pause the clock.
- void resume ()

  Resume the clock.
- Time getElapsedTime () const

  Get the elapsed time since the last restart.

Private Attributes

- TimePoint m_start
- TimePoint m_pause
- bool m_paused {false}

6.6.1 Detailed Description

Class for time management.

Definition at line 23 of file Clock.hpp.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 Clock()

```
myLib::Clock::Clock () [inline]
```

Definition at line 27 of file Clock.hpp.

6.6.2.2 ~Clock()

```
myLib::Clock::~Clock () [default]
```

6.6.3 Member Function Documentation

6.6.3.1 getElapsedTime()

```
myLib::Time myLib::Clock::getElapsedTime () const  [nodiscard]
```

Get the elapsed time since the last restart.

Returns

Time The elapsed time

Definition at line 22 of file clock.cpp.

6.6.3.2 pause()

```
void myLib::Clock::pause ()
```

Pause the clock.

Definition at line 3 of file clock.cpp.

References m_pause, and m_paused.

6.6.3.3 restart()

```
void myLib::Clock::restart ()  [inline]
```

Restart the clock.

Definition at line 34 of file Clock.hpp.

References m_start.

6.6.3.4 resume()

```
void myLib::Clock::resume ()
```

Resume the clock.

Definition at line 12 of file clock.cpp.

6.6.4 Member Data Documentation

6.6.4.1 m_pause

```
TimePoint myLib::Clock::m_pause  [private]
```

Definition at line 62 of file Clock.hpp.

Referenced by pause().

6.6.4.2 m_paused

```
bool myLib::Clock::m_paused {false}  [private]
```

Definition at line 67 of file Clock.hpp.

Referenced by pause().

6.6.4.3 m_start

```
TimePoint myLib::Clock::m_start  [private]
```

Definition at line 57 of file Clock.hpp.

Referenced by restart().

The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/Clock/Clock.hpp
- /home/runner/work/VEngine/VEngine/lib/local/static/myLib/src/clock.cpp

6.7 ven::DescriptorPool Class Reference

Class for descriptor pool.

```
#include <Descriptors.hpp>
```

Collaboration diagram for ven::DescriptorPool:

Classes

- class Builder

Public Member Functions

- DescriptorPool (Device &device, uint32_t maxSets, VkDescriptorPoolCreateFlags poolFlags, const std↩ ::vector< VkDescriptorPoolSize > &poolSizes)
- ~DescriptorPool ()
- DescriptorPool (const DescriptorPool &)=delete
- DescriptorPool & operator= (const DescriptorPool &)=delete
- bool allocateDescriptor (VkDescriptorSetLayout descriptorSetLayout, VkDescriptorSet &descriptor) const
- void freeDescriptors (const std::vector< VkDescriptorSet > &descriptors) const
- void resetPool () const

Private Attributes

- Device & m_device
- VkDescriptorPool m_descriptorPool

Friends

- class DescriptorWriter

6.7.1 Detailed Description

Class for descriptor pool.

Definition at line 61 of file Descriptors.hpp.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 DescriptorPool() [1/2]

```
ven::DescriptorPool::DescriptorPool (
            Device & device,
            uint32_t maxSets,
            VkDescriptorPoolCreateFlags poolFlags,
            const std::vector< VkDescriptorPoolSize > & poolSizes)
```

Definition at line 56 of file descriptors.cpp.

References ven::Device::device(), m_descriptorPool, and m_device.

Here is the call graph for this function:

6.7.2.2 ~DescriptorPool()

```
ven::DescriptorPool::~DescriptorPool ()  [inline]
```

Definition at line 85 of file Descriptors.hpp.

References ven::Device::device(), m_descriptorPool, and m_device.

Here is the call graph for this function:

6.7.2.3 DescriptorPool() [2/2]

```
ven::DescriptorPool::DescriptorPool (
            const DescriptorPool & )  [delete]
```

6.7.3 Member Function Documentation

6.7.3.1 allocateDescriptor()

```
bool ven::DescriptorPool::allocateDescriptor (
            VkDescriptorSetLayout descriptorSetLayout,
            VkDescriptorSet & descriptor) const
```

Definition at line 71 of file descriptors.cpp.

6.7.3.2 freeDescriptors()

```
void ven::DescriptorPool::freeDescriptors (
            const std::vector< VkDescriptorSet > & descriptors) const  [inline]
```

Definition at line 91 of file Descriptors.hpp.

References ven::Device::device(), m_descriptorPool, and m_device.

Here is the call graph for this function:

6.7.3.3 operator=()

```
DescriptorPool & ven::DescriptorPool::operator= (
            const DescriptorPool & )  [delete]
```

6.7.3.4 resetPool()

```
void ven::DescriptorPool::resetPool () const  [inline]
```

Definition at line 93 of file Descriptors.hpp.

References ven::Device::device(), m_descriptorPool, and m_device.

Here is the call graph for this function:

6.7.4 Friends And Related Symbol Documentation

6.7.4.1 DescriptorWriter

```
friend class DescriptorWriter  [friend]
```

Definition at line 100 of file Descriptors.hpp.

6.7.5 Member Data Documentation

6.7.5.1 m_descriptorPool

```
VkDescriptorPool ven::DescriptorPool::m_descriptorPool  [private]
```

Definition at line 98 of file Descriptors.hpp.

Referenced by DescriptorPool(), freeDescriptors(), resetPool(), and ~DescriptorPool().

6.7.5.2 m_device

```
Device& ven::DescriptorPool::m_device  [private]
```

Definition at line 97 of file Descriptors.hpp.

Referenced by DescriptorPool(), freeDescriptors(), resetPool(), and ~DescriptorPool().

The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors.hpp
- /home/runner/work/VEngine/VEngine/src/descriptors.cpp

6.8 ven::DescriptorSetLayout Class Reference

Class for descriptor set layout.

```
#include <Descriptors.hpp>
```

Collaboration diagram for ven::DescriptorSetLayout:

Classes

- class Builder

Public Member Functions

- DescriptorSetLayout (Device &device, const std::unordered_map< uint32_t, VkDescriptorSetLayoutBinding > &bindings)
- ~DescriptorSetLayout ()
- DescriptorSetLayout (const DescriptorSetLayout &)=delete
- DescriptorSetLayout & operator= (const DescriptorSetLayout &)=delete
- VkDescriptorSetLayout getDescriptorSetLayout () const

Private Attributes

- Device & m_device
- VkDescriptorSetLayout m_descriptorSetLayout
- std::unordered_map< uint32_t, VkDescriptorSetLayoutBinding > m_bindings

Friends

- class DescriptorWriter

6.8.1 Detailed Description

Class for descriptor set layout.

Definition at line 21 of file Descriptors.hpp.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 DescriptorSetLayout() [1/2]

```
ven::DescriptorSetLayout::DescriptorSetLayout (
            Device & device,
            const std::unordered_map< uint32_t, VkDescriptorSetLayoutBinding > & bindings)
```

Definition at line 17 of file descriptors.cpp.

References ven::Device::device(), m_descriptorSetLayout, and m_device.

Here is the call graph for this function:

6.8.2.2 ~DescriptorSetLayout()

```
ven::DescriptorSetLayout::~DescriptorSetLayout ()  [inline]
```

Definition at line 40 of file Descriptors.hpp.

References ven::Device::device(), m_descriptorSetLayout, and m_device.

Here is the call graph for this function:

6.8.2.3 DescriptorSetLayout() [2/2]

```
ven::DescriptorSetLayout::DescriptorSetLayout (
            const DescriptorSetLayout & )  [delete]
```

6.8.3 Member Function Documentation

6.8.3.1 getDescriptorSetLayout()

```
VkDescriptorSetLayout ven::DescriptorSetLayout::getDescriptorSetLayout () const  [inline]
```

Definition at line 44 of file Descriptors.hpp.

References m_descriptorSetLayout.

6.8.3.2 operator=()

```
DescriptorSetLayout & ven::DescriptorSetLayout::operator= (
            const DescriptorSetLayout & )  [delete]
```

6.8.4 Friends And Related Symbol Documentation

6.8.4.1 DescriptorWriter

```
friend class DescriptorWriter  [friend]
```

Definition at line 52 of file Descriptors.hpp.

6.8.5 Member Data Documentation

6.8.5.1 m_bindings

```
std::unordered_map<uint32_t, VkDescriptorSetLayoutBinding> ven::DescriptorSetLayout::m_↩
bindings  [private]
```

Definition at line 50 of file Descriptors.hpp.

6.8.5.2 m_descriptorSetLayout

```
VkDescriptorSetLayout ven::DescriptorSetLayout::m_descriptorSetLayout  [private]
```

Definition at line 49 of file Descriptors.hpp.

Referenced by DescriptorSetLayout(), getDescriptorSetLayout(), and ~DescriptorSetLayout().

6.8.5.3 m_device

```
Device& ven::DescriptorSetLayout::m_device  [private]
```

Definition at line 48 of file Descriptors.hpp.

Referenced by DescriptorSetLayout(), and ~DescriptorSetLayout().

The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors.hpp
- /home/runner/work/VEngine/VEngine/src/descriptors.cpp

6.9 ven::DescriptorWriter Class Reference

Class for descriptor writer.

```
#include <Descriptors.hpp>
```

Collaboration diagram for ven::DescriptorWriter:

Public Member Functions

- DescriptorWriter (DescriptorSetLayout &setLayout, DescriptorPool &pool)
- DescriptorWriter & writeBuffer (uint32_t binding, const VkDescriptorBufferInfo *bufferInfo)
- DescriptorWriter & writeImage (uint32_t binding, const VkDescriptorImageInfo *imageInfo)
- bool build (VkDescriptorSet &set)
- void overwrite (const VkDescriptorSet &set)

Private Attributes

- DescriptorSetLayout & m_setLayout
- DescriptorPool & m_pool
- std::vector< VkWriteDescriptorSet > m_writes

6.9.1 Detailed Description

Class for descriptor writer.

Definition at line 109 of file Descriptors.hpp.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 DescriptorWriter()

```
ven::DescriptorWriter::DescriptorWriter (
            DescriptorSetLayout & setLayout,
            DescriptorPool & pool)  [inline]
```

Definition at line 113 of file Descriptors.hpp.

6.9.3 Member Function Documentation

6.9.3.1 build()

```
bool ven::DescriptorWriter::build (
            VkDescriptorSet & set)
```

Definition at line 122 of file descriptors.cpp.

Referenced by ven::Engine::mainLoop().

Here is the caller graph for this function:

6.9.3.2 overwrite()

```
void ven::DescriptorWriter::overwrite (
            const VkDescriptorSet & set)
```

Definition at line 131 of file descriptors.cpp.

6.9.3.3 writeBuffer()

```
ven::DescriptorWriter & ven::DescriptorWriter::writeBuffer (
            uint32_t binding,
            const VkDescriptorBufferInfo * bufferInfo)
```

Definition at line 84 of file descriptors.cpp.

Referenced by ven::Engine::mainLoop().

Here is the caller graph for this function:

6.9.3.4 writeImage()

```
ven::DescriptorWriter & ven::DescriptorWriter::writeImage (
            uint32_t binding,
            const VkDescriptorImageInfo * imageInfo)
```

Definition at line 103 of file descriptors.cpp.

6.9.4 Member Data Documentation

6.9.4.1 m_pool

```
DescriptorPool& ven::DescriptorWriter::m_pool  [private]
```

Definition at line 124 of file Descriptors.hpp.

6.9.4.2 m_setLayout

```
DescriptorSetLayout& ven::DescriptorWriter::m_setLayout  [private]
```

Definition at line 123 of file Descriptors.hpp.

6.9.4.3 m_writes

```
std::vector<VkWriteDescriptorSet> ven::DescriptorWriter::m_writes  [private]
```

Definition at line 125 of file Descriptors.hpp.

The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors.hpp
- /home/runner/work/VEngine/VEngine/src/descriptors.cpp

6.10 ven::Device Class Reference

`#include <Device.hpp>`

Collaboration diagram for ven::Device:

Public Member Functions

- Device (Window &window)
- ~Device ()
- Device (const Device &)=delete
- Device & operator= (const Device &)=delete
- Device (Device &&)=delete
- Device & operator= (Device &&)=delete
- VkCommandPool getCommandPool () const
- VkDevice device () const
- VkSurfaceKHR surface () const
- VkQueue graphicsQueue () const
- VkQueue presentQueue () const
- SwapChainSupportDetails getSwapChainSupport () const
- uint32_t findMemoryType (uint32_t typeFilter, VkMemoryPropertyFlags propertiesp) const
- QueueFamilyIndices findPhysicalQueueFamilies () const
- VkFormat findSupportedFormat (const std::vector< VkFormat > &candidates, VkImageTiling tiling, VkFormatFeatureFlags features) const
- void createBuffer (VkDeviceSize size, VkBufferUsageFlags usage, VkMemoryPropertyFlags propertiesp, VkBuffer &buffer, VkDeviceMemory &bufferMemory) const
- VkCommandBuffer beginSingleTimeCommands () const
- void endSingleTimeCommands (VkCommandBuffer commandBuffer) const
- void copyBuffer (VkBuffer srcBuffer, VkBuffer dstBuffer, VkDeviceSize size) const
- void copyBufferToImage (VkBuffer buffer, VkImage image, uint32_t width, uint32_t height, uint32_t layerCount) const
- void createImageWithInfo (const VkImageCreateInfo &imageInfo, VkMemoryPropertyFlags properties, VkImage &image, VkDeviceMemory &imageMemory) const
- VkPhysicalDevice getPhysicalDevice () const
- VkQueue getGraphicsQueue () const

Public Attributes

- const bool enableValidationLayers = true
- VkPhysicalDeviceProperties m_properties

Private Member Functions

- void createInstance ()
- void setupDebugMessenger ()
- void createSurface ()
- void pickPhysicalDevice ()
- void createLogicalDevice ()
- void createCommandPool ()
- bool isDeviceSuitable (VkPhysicalDevice device) const
- std::vector< const char * > getRequiredExtensions () const
- bool checkValidationLayerSupport () const
- QueueFamilyIndices findQueueFamilies (VkPhysicalDevice device) const
- void hasGlfwRequiredInstanceExtensions () const
- bool checkDeviceExtensionSupport (VkPhysicalDevice device) const
- SwapChainSupportDetails querySwapChainSupport (VkPhysicalDevice device) const

Static Private Member Functions

- static void populateDebugMessengerCreateInfo (VkDebugUtilsMessengerCreateInfoEXT &createInfo)

Private Attributes

- VkInstance instance
- VkDebugUtilsMessengerEXT debugMessenger
- VkPhysicalDevice physicalDevice = VK_NULL_HANDLE
- Window & m_window
- VkCommandPool commandPool
- VkDevice device_
- VkSurfaceKHR surface_
- VkQueue graphicsQueue_
- VkQueue presentQueue_
- const std::vector< const char * > validationLayers = {"VK_LAYER_KHRONOS_validation"}
- const std::vector< const char * > deviceExtensions = {VK_KHR_SWAPCHAIN_EXTENSION_NAME}

6.10.1 Detailed Description

Definition at line 29 of file Device.hpp.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 Device() [1/3]

```
ven::Device::Device (
            Window & window)  [explicit]
```

Definition at line 34 of file device.cpp.

References createCommandPool(), createInstance(), createLogicalDevice(), createSurface(), pickPhysicalDevice(), and setupDebugMessenger().

Here is the call graph for this function:

6.10.2.2 ~Device()

```
ven::Device::~Device ()
```

Definition at line 44 of file device.cpp.

References DestroyDebugUtilsMessengerEXT().

Here is the call graph for this function:

6.10.2.3 Device() [2/3]

```
ven::Device::Device (
            const Device & )  [delete]
```

6.10.2.4 Device() [3/3]

```
ven::Device::Device (
            Device && )  [delete]
```

6.10.3 Member Function Documentation

6.10.3.1 beginSingleTimeCommands()

```
VkCommandBuffer ven::Device::beginSingleTimeCommands () const  [nodiscard]
```

Definition at line 411 of file device.cpp.

6.10.3.2 checkDeviceExtensionSupport()

```
bool ven::Device::checkDeviceExtensionSupport (
            VkPhysicalDevice device) const  [private]
```

Definition at line 289 of file device.cpp.

6.10.3.3 checkValidationLayerSupport()

```
bool ven::Device::checkValidationLayerSupport () const  [nodiscard], [private]
```

Definition at line 225 of file device.cpp.

6.10.3.4 copyBuffer()

```
void ven::Device::copyBuffer (
            VkBuffer srcBuffer,
            VkBuffer dstBuffer,
            VkDeviceSize size) const
```

Definition at line 445 of file device.cpp.

6.10.3.5 copyBufferToImage()

```
void ven::Device::copyBufferToImage (
            VkBuffer buffer,
            VkImage image,
            uint32_t width,
            uint32_t height,
            uint32_t layerCount) const
```

Definition at line 458 of file device.cpp.

6.10.3.6 createBuffer()

```
void ven::Device::createBuffer (
            VkDeviceSize size,
            VkBufferUsageFlags usage,
            VkMemoryPropertyFlags propertiesp,
            VkBuffer & buffer,
            VkDeviceMemory & bufferMemory) const
```

Definition at line 384 of file device.cpp.

Referenced by ven::Buffer::Buffer().

Here is the caller graph for this function:

6.10.3.7 createCommandPool()

```
void ven::Device::createCommandPool ()  [private]
```

Definition at line 171 of file device.cpp.

References ven::QueueFamilyIndices::graphicsFamily.

Referenced by Device().

Here is the caller graph for this function:

6.10.3.8 createImageWithInfo()

```
void ven::Device::createImageWithInfo (
            const VkImageCreateInfo & imageInfo,
            VkMemoryPropertyFlags properties,
            VkImage & image,
            VkDeviceMemory & imageMemory) const
```

Definition at line 479 of file device.cpp.

6.10.3.9 createInstance()

```
void ven::Device::createInstance ()  [private]
```

Definition at line 57 of file device.cpp.

Referenced by Device().

Here is the caller graph for this function:

6.10.3.10 createLogicalDevice()

```
void ven::Device::createLogicalDevice ()  [private]
```

Definition at line 124 of file device.cpp.

References ven::QueueFamilyIndices::graphicsFamily, and ven::QueueFamilyIndices::presentFamily.

Referenced by Device().

Here is the caller graph for this function:

6.10.3.11 createSurface()

```
void ven::Device::createSurface ()  [inline], [private]
```

Definition at line 76 of file Device.hpp.

References ven::Window::createWindowSurface(), instance, m_window, and surface_.

Referenced by Device().

Here is the call graph for this function: Here is the caller graph for this function:

6.10.3.12 device()

```
VkDevice ven::Device::device () const  [inline], [nodiscard]
```

Definition at line 48 of file Device.hpp.

References device_.

Referenced by ven::Renderer::createCommandBuffers(), ven::DescriptorPool::DescriptorPool(), ven::DescriptorSetLayout::Descriptor
ven::DescriptorPool::freeDescriptors(), ven::DescriptorPool::resetPool(), ven::DescriptorPool::~DescriptorPool(),
ven::DescriptorSetLayout::~DescriptorSetLayout(), ven::PointLightSystem::~PointLightSystem(), ven::RenderSystem::~RenderSyst
ven::Shaders::~Shaders(), and ven::SwapChain::~SwapChain().

Here is the caller graph for this function:

6.10.3.13 endSingleTimeCommands()

```
void ven::Device::endSingleTimeCommands (
            VkCommandBuffer commandBuffer) const
```

Definition at line 430 of file device.cpp.

6.10.3.14 findMemoryType()

```
uint32_t ven::Device::findMemoryType (
            uint32_t typeFilter,
            VkMemoryPropertyFlags propertiesp) const  [nodiscard]
```

Definition at line 369 of file device.cpp.

6.10.3.15 findPhysicalQueueFamilies()

```
QueueFamilyIndices ven::Device::findPhysicalQueueFamilies () const  [inline], [nodiscard]
```

Definition at line 55 of file Device.hpp.

References findQueueFamilies(), and physicalDevice.

Here is the call graph for this function:

6.10.3.16 findQueueFamilies()

```
ven::QueueFamilyIndices ven::Device::findQueueFamilies (
            VkPhysicalDevice device) const  [private]
```

Definition at line 305 of file device.cpp.

References ven::QueueFamilyIndices::graphicsFamily, ven::QueueFamilyIndices::graphicsFamilyHasValue, ven::QueueFamilyIndices::isComplete(), ven::QueueFamilyIndices::presentFamily, and ven::QueueFamilyIndices::presentFamilyHasV

Referenced by findPhysicalQueueFamilies().

Here is the call graph for this function: Here is the caller graph for this function:

6.10.3.17 findSupportedFormat()

```
VkFormat ven::Device::findSupportedFormat (
            const std::vector< VkFormat > & candidates,
            VkImageTiling tiling,
            VkFormatFeatureFlags features) const  [nodiscard]
```

Definition at line 355 of file device.cpp.

6.10.3.18 getCommandPool()

```
VkCommandPool ven::Device::getCommandPool () const  [inline], [nodiscard]
```

Definition at line 47 of file Device.hpp.

References commandPool.

Referenced by ven::Renderer::createCommandBuffers().

Here is the caller graph for this function:

6.10.3.19 getGraphicsQueue()

```
VkQueue ven::Device::getGraphicsQueue () const  [inline], [nodiscard]
```

Definition at line 70 of file Device.hpp.

References graphicsQueue_.

6.10.3.20 getPhysicalDevice()

```
VkPhysicalDevice ven::Device::getPhysicalDevice () const  [inline], [nodiscard]
```

Definition at line 69 of file Device.hpp.

References physicalDevice.

6.10.3.21 getRequiredExtensions()

```
std::vector< const char * > ven::Device::getRequiredExtensions () const  [nodiscard], [private]
```

Definition at line 250 of file device.cpp.

6.10.3.22 getSwapChainSupport()

```
SwapChainSupportDetails ven::Device::getSwapChainSupport () const  [inline], [nodiscard]
```

Definition at line 53 of file Device.hpp.

References physicalDevice, and querySwapChainSupport().

Here is the call graph for this function:

6.10.3.23 graphicsQueue()

```
VkQueue ven::Device::graphicsQueue () const  [inline], [nodiscard]
```

Definition at line 50 of file Device.hpp.

References graphicsQueue_.

6.10.3.24 hasGlfwRequiredInstanceExtensions()

```
void ven::Device::hasGlfwRequiredInstanceExtensions () const  [private]
```

Definition at line 265 of file device.cpp.

6.10.3.25 isDeviceSuitable()

```
bool ven::Device::isDeviceSuitable (
            VkPhysicalDevice device) const  [private]
```

Definition at line 185 of file device.cpp.

References ven::SwapChainSupportDetails::formats, ven::QueueFamilyIndices::isComplete(), and ven::SwapChainSupportDetails::pr

Here is the call graph for this function:

6.10.3.26 operator=() [1/2]

```
Device & ven::Device::operator= (
            const Device & )  [delete]
```

6.10.3.27 operator=() [2/2]

```
Device & ven::Device::operator= (
            Device && )  [delete]
```

6.10.3.28 pickPhysicalDevice()

```
void ven::Device::pickPhysicalDevice ()  [private]
```

Definition at line 98 of file device.cpp.

Referenced by Device().

Here is the caller graph for this function:

6.10.3.29 populateDebugMessengerCreateInfo()

```
void ven::Device::populateDebugMessengerCreateInfo (
            VkDebugUtilsMessengerCreateInfoEXT & createInfo)  [static], [private]
```

Definition at line 202 of file device.cpp.

References debugCallback().

Here is the call graph for this function:

6.10.3.30 presentQueue()

```
VkQueue ven::Device::presentQueue () const  [inline], [nodiscard]
```

Definition at line 51 of file Device.hpp.

References presentQueue_.

6.10.3.31 querySwapChainSupport()

```
ven::SwapChainSupportDetails ven::Device::querySwapChainSupport (
            VkPhysicalDevice device) const  [private]
```

Definition at line 334 of file device.cpp.

References ven::SwapChainSupportDetails::capabilities, ven::SwapChainSupportDetails::formats, and ven::SwapChainSupportDetail

Referenced by getSwapChainSupport().

Here is the caller graph for this function:

6.10.3.32 setupDebugMessenger()

```
void ven::Device::setupDebugMessenger ()  [private]
```

Definition at line 215 of file device.cpp.

References CreateDebugUtilsMessengerEXT().

Referenced by Device().

Here is the call graph for this function: Here is the caller graph for this function:

6.10.3.33 surface()

```
VkSurfaceKHR ven::Device::surface () const  [inline], [nodiscard]
```

Definition at line 49 of file Device.hpp.

References surface_.

6.10.4 Member Data Documentation

6.10.4.1 commandPool

```
VkCommandPool ven::Device::commandPool  [private]
```

Definition at line 95 of file Device.hpp.

Referenced by getCommandPool().

6.10.4.2 debugMessenger

```
VkDebugUtilsMessengerEXT ven::Device::debugMessenger  [private]
```

Definition at line 92 of file Device.hpp.

6.10.4.3 device_

```
VkDevice ven::Device::device_  [private]
```

Definition at line 97 of file Device.hpp.

Referenced by device().

6.10.4.4 deviceExtensions

```
const std::vector<const char *> ven::Device::deviceExtensions = {VK_KHR_SWAPCHAIN_EXTENSION_↩
NAME}  [private]
```

Definition at line 103 of file Device.hpp.

6.10.4.5 enableValidationLayers

```
const bool ven::Device::enableValidationLayers = true
```

Definition at line 36 of file Device.hpp.

6.10.4.6 graphicsQueue_

```
VkQueue ven::Device::graphicsQueue_  [private]
```

Definition at line 99 of file Device.hpp.

Referenced by getGraphicsQueue(), and graphicsQueue().

6.10.4.7 instance

```
VkInstance ven::Device::instance  [private]
```

Definition at line 91 of file Device.hpp.

Referenced by createSurface().

6.10.4.8 m_properties

```
VkPhysicalDeviceProperties ven::Device::m_properties
```

Definition at line 67 of file Device.hpp.

6.10.4.9 m_window

```
Window& ven::Device::m_window  [private]
```

Definition at line 94 of file Device.hpp.

Referenced by createSurface().

6.10.4.10 physicalDevice

```
VkPhysicalDevice ven::Device::physicalDevice = VK_NULL_HANDLE  [private]
```

Definition at line 93 of file Device.hpp.

Referenced by findPhysicalQueueFamilies(), getPhysicalDevice(), and getSwapChainSupport().

6.10.4.11 presentQueue_

```
VkQueue ven::Device::presentQueue_  [private]
```

Definition at line 100 of file Device.hpp.

Referenced by presentQueue().

6.10.4.12 surface_

```
VkSurfaceKHR ven::Device::surface_  [private]
```

Definition at line 98 of file Device.hpp.

Referenced by createSurface(), and surface().

6.10.4.13 validationLayers

```
const std::vector<const char *> ven::Device::validationLayers = {"VK_LAYER_KHRONOS_validation"}
[private]
```

Definition at line 102 of file Device.hpp.

The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp
- /home/runner/work/VEngine/VEngine/src/device.cpp

6.11 ven::Engine Class Reference

```
#include <Engine.hpp>
```

Collaboration diagram for ven::Engine:

Public Member Functions

- Engine (uint32_t=DEFAULT_WIDTH, uint32_t=DEFAULT_HEIGHT, const std::string &title=DEFAULT_↩ TITLE.data())
- ~Engine ()=default
- Engine (const Engine &)=delete
- Engine operator= (const Engine &)=delete
- Window & getWindow ()
- void mainLoop ()

Private Member Functions

- void loadObjects ()
- void createInstance ()
- void createSurface ()

Private Attributes

- Window m_window
- Device m_device {m_window}
- Renderer m_renderer {m_window, m_device}
- std::unique_ptr< DescriptorPool > m_globalPool
- Object::Map m_objects
- VkInstance m_instance {nullptr}
- VkSurfaceKHR m_surface {nullptr}

6.11.1 Detailed Description

Definition at line 20 of file Engine.hpp.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 Engine() [1/2]

```
ven::Engine::Engine (
            uint32_t width = DEFAULT_WIDTH,
            uint32_t height = DEFAULT_HEIGHT,
            const std::string & title = DEFAULT_TITLE.data())  [explicit]
```

Definition at line 59 of file engine.cpp.

References ven::DescriptorPool::Builder::addPoolSize(), ven::DescriptorPool::Builder::build(), createInstance(), createSurface(), loadObjects(), m_device, m_globalPool, ven::SwapChain::MAX_FRAMES_IN_FLIGHT, and ven::DescriptorPool::Builder::setMaxSets().

Here is the call graph for this function:

6.11.2.2 ~Engine()

```
ven::Engine::~Engine ()  [default]
```

6.11.2.3 Engine() [2/2]

```
ven::Engine::Engine (
            const Engine & )  [delete]
```

6.11.3 Member Function Documentation

6.11.3.1 createInstance()

```
void ven::Engine::createInstance ()  [private]
```

Definition at line 133 of file engine.cpp.

Referenced by Engine().

Here is the caller graph for this function:

6.11.3.2 createSurface()

```
void ven::Engine::createSurface ()  [inline], [private]
```

Definition at line 49 of file Engine.hpp.

References ven::Window::getGLFWindow(), m_instance, m_surface, and m_window.

Referenced by Engine().

Here is the call graph for this function: Here is the caller graph for this function:

6.11.3.3 getWindow()

```
Window & ven::Engine::getWindow ()  [inline]
```

Definition at line 30 of file Engine.hpp.

References m_window.

6.11.3.4 loadObjects()

```
void ven::Engine::loadObjects ()  [private]
```

Definition at line 16 of file engine.cpp.

References ven::Object::color, ven::Model::createModelFromFile(), ven::Object::createObject(), ven::Object::getId(), m_device, m_objects, ven::Object::makePointLight(), ven::Object::model, ven::Transform3DComponent::scale, ven::Object::transform3D, and ven::Transform3DComponent::translation.

Referenced by Engine().

Here is the call graph for this function: Here is the caller graph for this function:

6.11.3.5 mainLoop()

```
void ven::Engine::mainLoop ()
```

Definition at line 67 of file engine.cpp.

References ven::DescriptorSetLayout::Builder::addBinding(), ven::DescriptorSetLayout::Builder::build(), ven::DescriptorWriter::build(), ven::Object::createObject(), ven::Renderer::endSwapChainRenderPass(), ven::SwapChain::MAX_FRAMES_IN_FLIGHT, ven::GlobalUbo::projection, ven::PointLightSystem::render(), ven::RenderSystem::renderObjects(), ven::Transform3DComponent::rota ven::Object::transform3D, ven::Transform3DComponent::translation, ven::PointLightSystem::update(), and ven::DescriptorWriter::writeBuffer().

Referenced by main().

Here is the call graph for this function: Here is the caller graph for this function:

6.11.3.6 operator=()

```
Engine ven::Engine::operator= (
            const Engine & )  [delete]
```

6.11.4 Member Data Documentation

6.11.4.1 m_device

```
Device ven::Engine::m_device {m_window}  [private]
```

Definition at line 39 of file Engine.hpp.

Referenced by Engine(), and loadObjects().

6.11.4.2 m_globalPool

```
std::unique_ptr<DescriptorPool> ven::Engine::m_globalPool  [private]
```

Definition at line 42 of file Engine.hpp.

Referenced by Engine().

6.11.4.3 m_instance

```
VkInstance ven::Engine::m_instance {nullptr}  [private]
```

Definition at line 45 of file Engine.hpp.

Referenced by createSurface().

6.11.4.4 m_objects

```
Object::Map ven::Engine::m_objects  [private]
```

Definition at line 43 of file Engine.hpp.

Referenced by loadObjects().

6.11.4.5 m_renderer

```
Renderer ven::Engine::m_renderer {m_window, m_device}  [private]
```

Definition at line 40 of file Engine.hpp.

6.11.4.6 m_surface

```
VkSurfaceKHR ven::Engine::m_surface {nullptr}  [private]
```

Definition at line 46 of file Engine.hpp.

Referenced by createSurface().

6.11.4.7 m_window

```
Window ven::Engine::m_window  [private]
```

Definition at line 38 of file Engine.hpp.

Referenced by createSurface(), and getWindow().

The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/Engine.hpp
- /home/runner/work/VEngine/VEngine/src/engine.cpp

6.12 ven::FrameCounter Class Reference

```
#include <FrameCounter.hpp>
```

Collaboration diagram for ven::FrameCounter:

Public Member Functions

- FrameCounter ()=default
- ~FrameCounter ()=default
- void update (const float deltaTime)
- float getFps () const
- float getFrameTime () const

Private Attributes

- float m_fps {0.F}
- float m_frameTime {0.F}
- float m_frameCounter {0.F}
- float m_timeCounter {0.F}

6.12.1 Detailed Description

Definition at line 13 of file FrameCounter.hpp.

6.12.2 Constructor & Destructor Documentation

6.12.2.1 FrameCounter()

```
ven::FrameCounter::FrameCounter ()  [default]
```

6.12.2.2 ~FrameCounter()

```
ven::FrameCounter::~FrameCounter ()  [default]
```

6.12.3 Member Function Documentation

6.12.3.1 getFps()

```
float ven::FrameCounter::getFps () const  [inline], [nodiscard]
```

Definition at line 33 of file FrameCounter.hpp.

References m_fps.

6.12.3.2 getFrameTime()

```
float ven::FrameCounter::getFrameTime () const  [inline], [nodiscard]
```

Definition at line 34 of file FrameCounter.hpp.

References m_frameTime.

6.12.3.3 update()

```
void ven::FrameCounter::update (
            const float deltaTime)  [inline]
```

Definition at line 20 of file FrameCounter.hpp.

References m_fps, m_frameCounter, m_frameTime, and m_timeCounter.

6.12.4 Member Data Documentation

6.12.4.1 m_fps

```
float ven::FrameCounter::m_fps {0.F}  [private]
```

Definition at line 38 of file FrameCounter.hpp.

Referenced by getFps(), and update().

6.12.4.2 m_frameCounter

```
float ven::FrameCounter::m_frameCounter {0.F}  [private]
```

Definition at line 40 of file FrameCounter.hpp.

Referenced by update().

6.12.4.3 m_frameTime

```
float ven::FrameCounter::m_frameTime {0.F}  [private]
```

Definition at line 39 of file FrameCounter.hpp.

Referenced by getFrameTime(), and update().

6.12.4.4 m_timeCounter

```
float ven::FrameCounter::m_timeCounter {0.F}  [private]
```

Definition at line 41 of file FrameCounter.hpp.

Referenced by update().

The documentation for this class was generated from the following file:

- /home/runner/work/VEngine/VEngine/include/VEngine/FrameCounter.hpp

6.13 ven::FrameInfo Struct Reference

```
#include <FrameInfo.hpp>
```

Collaboration diagram for ven::FrameInfo:

Public Attributes

- int frameIndex
- float frameTime
- VkCommandBuffer commandBuffer
- Camera & camera
- VkDescriptorSet globalDescriptorSet
- Object::Map & objects

6.13.1 Detailed Description

Definition at line 34 of file FrameInfo.hpp.

6.13.2 Member Data Documentation

6.13.2.1 camera

```
Camera& ven::FrameInfo::camera
```

Definition at line 39 of file FrameInfo.hpp.

6.13.2.2 commandBuffer

```
VkCommandBuffer ven::FrameInfo::commandBuffer
```

Definition at line 38 of file FrameInfo.hpp.

Referenced by ven::PointLightSystem::render(), and ven::RenderSystem::renderObjects().

6.13.2.3 frameIndex

```
int ven::FrameInfo::frameIndex
```

Definition at line 36 of file FrameInfo.hpp.

6.13.2.4 frameTime

```
float ven::FrameInfo::frameTime
```

Definition at line 37 of file FrameInfo.hpp.

Referenced by ven::PointLightSystem::update().

6.13.2.5 globalDescriptorSet

```
VkDescriptorSet ven::FrameInfo::globalDescriptorSet
```

Definition at line 40 of file FrameInfo.hpp.

Referenced by ven::PointLightSystem::render(), and ven::RenderSystem::renderObjects().

6.13.2.6 objects

```
Object::Map& ven::FrameInfo::objects
```

Definition at line 41 of file FrameInfo.hpp.

Referenced by ven::PointLightSystem::render(), ven::RenderSystem::renderObjects(), and ven::PointLightSystem::update().

The documentation for this struct was generated from the following file:

- /home/runner/work/VEngine/VEngine/include/VEngine/FrameInfo.hpp

6.14 ven::GlobalUbo Struct Reference

`#include <FrameInfo.hpp>`

Collaboration diagram for ven::GlobalUbo:

Public Attributes

- glm::mat4 projection {1.F}
- glm::mat4 view {1.F}
- glm::mat4 inverseView {1.F}
- glm::vec4 ambientLightColor {1.F, 1.F, 1.F, .02F}
- std::array< PointLight, MAX_LIGHTS > pointLights
- int numLights

6.14.1 Detailed Description

Definition at line 24 of file FrameInfo.hpp.

6.14.2 Member Data Documentation

6.14.2.1 ambientLightColor

```
glm::vec4 ven::GlobalUbo::ambientLightColor {1.F, 1.F, 1.F, .02F}
```

Definition at line 29 of file FrameInfo.hpp.

6.14.2.2 inverseView

```
glm::mat4 ven::GlobalUbo::inverseView {1.F}
```

Definition at line 28 of file FrameInfo.hpp.

6.14.2.3 numLights

```
int ven::GlobalUbo::numLights
```

Definition at line 31 of file FrameInfo.hpp.

Referenced by ven::PointLightSystem::update().

6.14.2.4 pointLights

```
std::array<PointLight, MAX_LIGHTS> ven::GlobalUbo::pointLights
```

Definition at line 30 of file FrameInfo.hpp.

Referenced by ven::PointLightSystem::update().

6.14.2.5 projection

```
glm::mat4 ven::GlobalUbo::projection {1.F}
```

Definition at line 26 of file FrameInfo.hpp.

Referenced by ven::Engine::mainLoop().

6.14.2.6 view

```
glm::mat4 ven::GlobalUbo::view {1.F}
```

Definition at line 27 of file FrameInfo.hpp.

The documentation for this struct was generated from the following file:

- /home/runner/work/VEngine/VEngine/include/VEngine/FrameInfo.hpp

6.15 std::hash< ven::Model::Vertex > Struct Reference

Collaboration diagram for std::hash< ven::Model::Vertex >:

Public Member Functions

- size_t operator() (ven::Model::Vertex const &vertex) const

6.15.1 Detailed Description

Definition at line 16 of file model.cpp.

6.15.2 Member Function Documentation

6.15.2.1 operator()()

```
size_t std::hash< ven::Model::Vertex >::operator() (
            ven::Model::Vertex const & vertex) const  [inline]
```

Definition at line 17 of file model.cpp.

References ven::Model::Vertex::color, ven::hashCombine(), ven::Model::Vertex::normal, ven::Model::Vertex::position, and ven::Model::Vertex::uv.

Here is the call graph for this function:

The documentation for this struct was generated from the following file:

- /home/runner/work/VEngine/VEngine/src/model.cpp

6.16 ven::KeyboardController Class Reference

`#include <KeyboardController.hpp>`

Collaboration diagram for ven::KeyboardController:

Classes

- struct KeyMappings

Public Member Functions

- void moveInPlaneXZ (GLFWwindow *window, float dt, Object &object) const

Public Attributes

- KeyMappings m_keys {}
- float m_moveSpeed {3.F}
- float m_lookSpeed {1.5F}

6.16.1 Detailed Description

Definition at line 14 of file KeyboardController.hpp.

6.16.2 Member Function Documentation

6.16.2.1 moveInPlaneXZ()

```
void ven::KeyboardController::moveInPlaneXZ (
            GLFWwindow * window,
            float dt,
            Object & object) const
```

Definition at line 5 of file keyboardController.cpp.

References ven::KeyboardController::KeyMappings::lookDown, ven::KeyboardController::KeyMappings::lookLeft, ven::KeyboardController::KeyMappings::lookRight, ven::KeyboardController::KeyMappings::lookUp, m_keys, m_lookSpeed, m_moveSpeed, ven::KeyboardController::KeyMappings::moveBackward, ven::KeyboardController::KeyMappings::mov ven::KeyboardController::KeyMappings::moveForward, ven::KeyboardController::KeyMappings::moveLeft, ven::KeyboardController::K and ven::KeyboardController::KeyMappings::moveUp.

6.16.3 Member Data Documentation

6.16.3.1 m_keys

```
KeyMappings ven::KeyboardController::m_keys {}
```

Definition at line 33 of file KeyboardController.hpp.

Referenced by moveInPlaneXZ().

6.16.3.2 m_lookSpeed

```
float ven::KeyboardController::m_lookSpeed {1.5F}
```

Definition at line 35 of file KeyboardController.hpp.

Referenced by moveInPlaneXZ().

6.16.3.3 m_moveSpeed

```
float ven::KeyboardController::m_moveSpeed {3.F}
```

Definition at line 34 of file KeyboardController.hpp.

Referenced by moveInPlaneXZ().

The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/KeyboardController.hpp
- /home/runner/work/VEngine/VEngine/src/keyboardController.cpp

6.17 ven::KeyboardController::KeyMappings Struct Reference

```
#include <KeyboardController.hpp>
```

Collaboration diagram for ven::KeyboardController::KeyMappings:

Public Attributes

- int moveLeft = GLFW_KEY_A
- int moveRight = GLFW_KEY_D
- int moveForward = GLFW_KEY_W
- int moveBackward = GLFW_KEY_S
- int moveUp = GLFW_KEY_SPACE
- int moveDown = GLFW_KEY_LEFT_SHIFT
- int lookLeft = GLFW_KEY_LEFT
- int lookRight = GLFW_KEY_RIGHT
- int lookUp = GLFW_KEY_UP
- int lookDown = GLFW_KEY_DOWN

6.17.1 Detailed Description

Definition at line 18 of file KeyboardController.hpp.

6.17.2 Member Data Documentation

6.17.2.1 lookDown

```
int ven::KeyboardController::KeyMappings::lookDown = GLFW_KEY_DOWN
```

Definition at line 28 of file KeyboardController.hpp.

Referenced by ven::KeyboardController::moveInPlaneXZ().

6.17.2.2 lookLeft

```
int ven::KeyboardController::KeyMappings::lookLeft = GLFW_KEY_LEFT
```

Definition at line 25 of file KeyboardController.hpp.

Referenced by ven::KeyboardController::moveInPlaneXZ().

6.17.2.3 lookRight

```
int ven::KeyboardController::KeyMappings::lookRight = GLFW_KEY_RIGHT
```

Definition at line 26 of file KeyboardController.hpp.

Referenced by ven::KeyboardController::moveInPlaneXZ().

6.17.2.4 lookUp

```
int ven::KeyboardController::KeyMappings::lookUp = GLFW_KEY_UP
```

Definition at line 27 of file KeyboardController.hpp.

Referenced by ven::KeyboardController::moveInPlaneXZ().

6.17.2.5 moveBackward

```
int ven::KeyboardController::KeyMappings::moveBackward = GLFW_KEY_S
```

Definition at line 22 of file KeyboardController.hpp.

Referenced by ven::KeyboardController::moveInPlaneXZ().

6.17.2.6 moveDown

```
int ven::KeyboardController::KeyMappings::moveDown = GLFW_KEY_LEFT_SHIFT
```

Definition at line 24 of file KeyboardController.hpp.

Referenced by ven::KeyboardController::moveInPlaneXZ().

6.17.2.7 moveForward

```
int ven::KeyboardController::KeyMappings::moveForward = GLFW_KEY_W
```

Definition at line 21 of file KeyboardController.hpp.

Referenced by ven::KeyboardController::moveInPlaneXZ().

6.17.2.8 moveLeft

```
int ven::KeyboardController::KeyMappings::moveLeft = GLFW_KEY_A
```

Definition at line 19 of file KeyboardController.hpp.

Referenced by ven::KeyboardController::moveInPlaneXZ().

6.17.2.9 moveRight

```
int ven::KeyboardController::KeyMappings::moveRight = GLFW_KEY_D
```

Definition at line 20 of file KeyboardController.hpp.

Referenced by ven::KeyboardController::moveInPlaneXZ().

6.17.2.10 moveUp

```
int ven::KeyboardController::KeyMappings::moveUp = GLFW_KEY_SPACE
```

Definition at line 23 of file KeyboardController.hpp.

Referenced by ven::KeyboardController::moveInPlaneXZ().

The documentation for this struct was generated from the following file:

- /home/runner/work/VEngine/VEngine/include/VEngine/KeyboardController.hpp

6.18 ven::Model Class Reference

```
#include <Model.hpp>
```

Collaboration diagram for ven::Model:

Classes

- struct Builder
- struct Vertex

Public Member Functions

- Model (Device &device, const Builder &builder)
- ~Model ()
- Model (const Model &)=delete
- void operator= (const Model &)=delete
- void bind (VkCommandBuffer commandBuffer) const
- void draw (VkCommandBuffer commandBuffer) const

Static Public Member Functions

- static std::unique_ptr< Model > createModelFromFile (Device &device, const std::string &filename)

Private Member Functions

- void createVertexBuffer (const std::vector< Vertex > &vertices)
- void createIndexBuffer (const std::vector< uint32_t > &indices)

Private Attributes

- Device & m_device
- std::unique_ptr< Buffer > m_vertexBuffer
- uint32_t m_vertexCount
- bool m_hasIndexBuffer {false}
- std::unique_ptr< Buffer > m_indexBuffer
- uint32_t m_indexCount

6.18.1 Detailed Description

Definition at line 16 of file Model.hpp.

6.18.2 Constructor & Destructor Documentation

6.18.2.1 Model() [1/2]

```
ven::Model::Model (
            Device & device,
            const Builder & builder)
```

Definition at line 25 of file model.cpp.

References createIndexBuffer(), createVertexBuffer(), ven::Model::Builder::indices, and ven::Model::Builder::vertices.

Here is the call graph for this function:

6.18.2.2 ~Model()

```
ven::Model::~Model ()  [default]
```

6.18.2.3 Model() [2/2]

```
ven::Model::Model (
            const Model & )  [delete]
```

6.18.3 Member Function Documentation

6.18.3.1 bind()

```
void ven::Model::bind (
            VkCommandBuffer commandBuffer) const
```

Definition at line 81 of file model.cpp.

6.18.3.2 createIndexBuffer()

```
void ven::Model::createIndexBuffer (
            const std::vector< uint32_t > & indices)  [private]
```

Definition at line 50 of file model.cpp.

References ven::Buffer::map().

Referenced by Model().

Here is the call graph for this function: Here is the caller graph for this function:

6.18.3.3 createModelFromFile()

```
std::unique_ptr< ven::Model > ven::Model::createModelFromFile (
            Device & device,
            const std::string & filename)  [static]
```

Definition at line 92 of file model.cpp.

References ven::Model::Builder::loadModel().

Referenced by ven::Engine::loadObjects().

Here is the call graph for this function: Here is the caller graph for this function:

6.18.3.4 createVertexBuffer()

```
void ven::Model::createVertexBuffer (
            const std::vector< Vertex > & vertices)  [private]
```

Definition at line 33 of file model.cpp.

References ven::Buffer::map().

Referenced by Model().

Here is the call graph for this function: Here is the caller graph for this function:

6.18.3.5 draw()

```
void ven::Model::draw (
            VkCommandBuffer commandBuffer) const
```

Definition at line 72 of file model.cpp.

6.18.3.6 operator=()

```
void ven::Model::operator= (
            const Model & )  [delete]
```

6.18.4 Member Data Documentation

6.18.4.1 m_device

```
Device& ven::Model::m_device  [private]
```

Definition at line 57 of file Model.hpp.

6.18.4.2 m_hasIndexBuffer

```
bool ven::Model::m_hasIndexBuffer {false}  [private]
```

Definition at line 61 of file Model.hpp.

6.18.4.3 m_indexBuffer

```
std::unique_ptr<Buffer> ven::Model::m_indexBuffer  [private]
```

Definition at line 62 of file Model.hpp.

6.18.4.4 m_indexCount

```
uint32_t ven::Model::m_indexCount  [private]
```

Definition at line 63 of file Model.hpp.

6.18.4.5 m_vertexBuffer

```
std::unique_ptr<Buffer> ven::Model::m_vertexBuffer  [private]
```

Definition at line 58 of file Model.hpp.

6.18.4.6 m_vertexCount

```
uint32_t ven::Model::m_vertexCount  [private]
```

Definition at line 59 of file Model.hpp.

The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp
- /home/runner/work/VEngine/VEngine/src/model.cpp

6.19 ven::Object Class Reference

```
#include <Object.hpp>
```

Collaboration diagram for ven::Object:

Public Types

- using Map = std::unordered_map<id_t, Object>

Public Member Functions

- ~Object ()=default
- Object (const Object &)=delete
- Object & operator= (const Object &)=delete
- Object (Object &&)=default
- Object & operator= (Object &&)=default
- id_t getId () const

Static Public Member Functions

- static Object createObject ()
- static Object makePointLight (float intensity=10.F, float radius=0.1F, glm::vec3 color=glm::vec3(1.F))

Public Attributes

- std::shared_ptr< Model > model {}
- glm::vec3 color {}
- Transform3DComponent transform3D {}
- std::unique_ptr< PointLightComponent > pointLight = nullptr

Private Member Functions

- Object (const id_t objId)

Private Attributes

- id_t m_objId

6.19.1 Detailed Description

Definition at line 33 of file Object.hpp.

6.19.2 Member Typedef Documentation

6.19.2.1 Map

```
using ven::Object::Map = std::unordered_map<id_t, Object>
```

Definition at line 37 of file Object.hpp.

6.19.3 Constructor & Destructor Documentation

6.19.3.1 ~Object()

```
ven::Object::~Object ()  [default]
```

6.19.3.2 Object() [1/3]

```
ven::Object::Object (
            const Object & )  [delete]
```

Referenced by createObject().

Here is the caller graph for this function:

6.19.3.3 Object() [2/3]

```
ven::Object::Object (
            Object && )  [default]
```

6.19.3.4 Object() [3/3]

```
ven::Object::Object (
            const id_t objId)  [inline], [explicit], [private]
```

Definition at line 60 of file Object.hpp.

6.19.4 Member Function Documentation

6.19.4.1 createObject()

```
static Object ven::Object::createObject ()  [inline], [static]
```

Definition at line 40 of file Object.hpp.

References Object().

Referenced by ven::Engine::loadObjects(), ven::Engine::mainLoop(), and makePointLight().

Here is the call graph for this function: Here is the caller graph for this function:

6.19.4.2 getId()

```
id_t ven::Object::getId () const  [inline], [nodiscard]
```

Definition at line 51 of file Object.hpp.

References m_objId.

Referenced by ven::Engine::loadObjects().

Here is the caller graph for this function:

6.19.4.3 makePointLight()

```
ven::Object ven::Object::makePointLight (
            float intensity = 10.F,
            float radius = 0.1F,
            glm::vec3 color = glm::vec3(1.F))  [static]
```

Definition at line 67 of file object.cpp.

References color, createObject(), pointLight, ven::Transform3DComponent::scale, and transform3D.

Referenced by ven::Engine::loadObjects().

Here is the call graph for this function: Here is the caller graph for this function:

6.19.4.4 operator=() [1/2]

```
Object & ven::Object::operator= (
            const Object & )  [delete]
```

6.19.4.5 operator=() [2/2]

```
Object & ven::Object::operator= (
            Object && )  [default]
```

6.19.5 Member Data Documentation

6.19.5.1 color

```
glm::vec3 ven::Object::color {}
```

Definition at line 54 of file Object.hpp.

Referenced by ven::Engine::loadObjects(), and makePointLight().

6.19.5.2 m_objId

```
id_t ven::Object::m_objId  [private]
```

Definition at line 62 of file Object.hpp.

Referenced by getId().

6.19.5.3 model

```
std::shared_ptr<Model> ven::Object::model {}
```

Definition at line 53 of file Object.hpp.

Referenced by ven::Engine::loadObjects().

6.19.5.4 pointLight

```
std::unique_ptr<PointLightComponent> ven::Object::pointLight = nullptr
```

Definition at line 57 of file Object.hpp.

Referenced by makePointLight().

6.19.5.5 transform3D

```
Transform3DComponent ven::Object::transform3D {}
```

Definition at line 55 of file Object.hpp.

Referenced by ven::Engine::loadObjects(), ven::Engine::mainLoop(), and makePointLight().

The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/Object.hpp
- /home/runner/work/VEngine/VEngine/src/object.cpp

6.20 ven::PipelineConfigInfo Struct Reference

`#include <Shaders.hpp>`

Collaboration diagram for ven::PipelineConfigInfo:

Public Member Functions

- PipelineConfigInfo ()=default
- PipelineConfigInfo (const PipelineConfigInfo &)=delete
- PipelineConfigInfo & operator= (const PipelineConfigInfo &)=delete

Public Attributes

- std::vector< VkVertexInputBindingDescription > bindingDescriptions
- std::vector< VkVertexInputAttributeDescription > attributeDescriptions
- VkPipelineInputAssemblyStateCreateInfo inputAssemblyInfo {}
- VkPipelineRasterizationStateCreateInfo rasterizationInfo {}
- VkPipelineMultisampleStateCreateInfo multisampleInfo {}
- VkPipelineColorBlendAttachmentState colorBlendAttachment {}
- VkPipelineColorBlendStateCreateInfo colorBlendInfo {}
- VkPipelineDepthStencilStateCreateInfo depthStencilInfo {}
- std::vector< VkDynamicState > dynamicStateEnables
- VkPipelineDynamicStateCreateInfo dynamicStateInfo {}
- VkPipelineLayout pipelineLayout = nullptr
- VkRenderPass renderPass = nullptr
- uint32_t subpass = 0

6.20.1 Detailed Description

Definition at line 19 of file Shaders.hpp.

6.20.2 Constructor & Destructor Documentation

6.20.2.1 PipelineConfigInfo() [1/2]

```
ven::PipelineConfigInfo::PipelineConfigInfo () [default]
```

6.20.2.2 PipelineConfigInfo() [2/2]

```
ven::PipelineConfigInfo::PipelineConfigInfo (
            const PipelineConfigInfo & ) [delete]
```

6.20.3 Member Function Documentation

6.20.3.1 operator=()

```
PipelineConfigInfo & ven::PipelineConfigInfo::operator= (
            const PipelineConfigInfo & )  [delete]
```

6.20.4 Member Data Documentation

6.20.4.1 attributeDescriptions

```
std::vector<VkVertexInputAttributeDescription> ven::PipelineConfigInfo::attributeDescriptions
```

Definition at line 25 of file Shaders.hpp.

Referenced by ven::Shaders::createGraphicsPipeline(), and ven::Shaders::defaultPipelineConfigInfo().

6.20.4.2 bindingDescriptions

```
std::vector<VkVertexInputBindingDescription> ven::PipelineConfigInfo::bindingDescriptions
```

Definition at line 24 of file Shaders.hpp.

Referenced by ven::Shaders::createGraphicsPipeline(), and ven::Shaders::defaultPipelineConfigInfo().

6.20.4.3 colorBlendAttachment

```
VkPipelineColorBlendAttachmentState ven::PipelineConfigInfo::colorBlendAttachment {}
```

Definition at line 29 of file Shaders.hpp.

Referenced by ven::Shaders::defaultPipelineConfigInfo().

6.20.4.4 colorBlendInfo

```
VkPipelineColorBlendStateCreateInfo ven::PipelineConfigInfo::colorBlendInfo {}
```

Definition at line 30 of file Shaders.hpp.

Referenced by ven::Shaders::createGraphicsPipeline(), and ven::Shaders::defaultPipelineConfigInfo().

6.20.4.5 depthStencilInfo

```
VkPipelineDepthStencilStateCreateInfo ven::PipelineConfigInfo::depthStencilInfo {}
```

Definition at line 31 of file Shaders.hpp.

Referenced by ven::Shaders::createGraphicsPipeline(), and ven::Shaders::defaultPipelineConfigInfo().

6.20.4.6 dynamicStateEnables

```
std::vector<VkDynamicState> ven::PipelineConfigInfo::dynamicStateEnables
```

Definition at line 32 of file Shaders.hpp.

Referenced by ven::Shaders::defaultPipelineConfigInfo().

6.20.4.7 dynamicStateInfo

```
VkPipelineDynamicStateCreateInfo ven::PipelineConfigInfo::dynamicStateInfo {}
```

Definition at line 33 of file Shaders.hpp.

Referenced by ven::Shaders::createGraphicsPipeline(), and ven::Shaders::defaultPipelineConfigInfo().

6.20.4.8 inputAssemblyInfo

```
VkPipelineInputAssemblyStateCreateInfo ven::PipelineConfigInfo::inputAssemblyInfo {}
```

Definition at line 26 of file Shaders.hpp.

Referenced by ven::Shaders::createGraphicsPipeline(), and ven::Shaders::defaultPipelineConfigInfo().

6.20.4.9 multisampleInfo

```
VkPipelineMultisampleStateCreateInfo ven::PipelineConfigInfo::multisampleInfo {}
```

Definition at line 28 of file Shaders.hpp.

Referenced by ven::Shaders::createGraphicsPipeline(), and ven::Shaders::defaultPipelineConfigInfo().

6.20.4.10 pipelineLayout

```
VkPipelineLayout ven::PipelineConfigInfo::pipelineLayout = nullptr
```

Definition at line 34 of file Shaders.hpp.

Referenced by ven::Shaders::createGraphicsPipeline().

6.20.4.11 rasterizationInfo

```
VkPipelineRasterizationStateCreateInfo ven::PipelineConfigInfo::rasterizationInfo {}
```

Definition at line 27 of file Shaders.hpp.

Referenced by ven::Shaders::createGraphicsPipeline(), and ven::Shaders::defaultPipelineConfigInfo().

6.20.4.12 renderPass

```
VkRenderPass ven::PipelineConfigInfo::renderPass = nullptr
```

Definition at line 35 of file Shaders.hpp.

Referenced by ven::Shaders::createGraphicsPipeline().

6.20.4.13 subpass

```
uint32_t ven::PipelineConfigInfo::subpass = 0
```

Definition at line 36 of file Shaders.hpp.

Referenced by ven::Shaders::createGraphicsPipeline().

The documentation for this struct was generated from the following file:

- /home/runner/work/VEngine/VEngine/include/VEngine/Shaders.hpp

6.21 ven::PointLight Struct Reference

```
#include <FrameInfo.hpp>
```

Collaboration diagram for ven::PointLight:

Public Attributes

- glm::vec4 position {}
- glm::vec4 color {}

6.21.1 Detailed Description

Definition at line 18 of file FrameInfo.hpp.

6.21.2 Member Data Documentation

6.21.2.1 color

```
glm::vec4 ven::PointLight::color {}
```

Definition at line 21 of file FrameInfo.hpp.

6.21.2.2 position

```
glm::vec4 ven::PointLight::position {}
```

Definition at line 20 of file FrameInfo.hpp.

The documentation for this struct was generated from the following file:

- /home/runner/work/VEngine/VEngine/include/VEngine/FrameInfo.hpp

6.22 ven::PointLightComponent Struct Reference

```
#include <Object.hpp>
```

Collaboration diagram for ven::PointLightComponent:

Public Attributes

- float lightIntensity = 1.0F

6.22.1 Detailed Description

Definition at line 29 of file Object.hpp.

6.22.2 Member Data Documentation

6.22.2.1 lightIntensity

```
float ven::PointLightComponent::lightIntensity = 1.0F
```

Definition at line 30 of file Object.hpp.

The documentation for this struct was generated from the following file:

- /home/runner/work/VEngine/VEngine/include/VEngine/Object.hpp

6.23 PointLightPushConstants Struct Reference

Collaboration diagram for PointLightPushConstants:

Public Attributes

- glm::vec4 position {}
- glm::vec4 color {}
- float radius

6.23.1 Detailed Description

Definition at line 9 of file pointLightSystem.cpp.

6.23.2 Member Data Documentation

6.23.2.1 color

```
glm::vec4 PointLightPushConstants::color {}
```

Definition at line 11 of file pointLightSystem.cpp.

6.23.2.2 position

```
glm::vec4 PointLightPushConstants::position {}
```

Definition at line 10 of file pointLightSystem.cpp.

Referenced by ven::PointLightSystem::render().

6.23.2.3 radius

```
float PointLightPushConstants::radius
```

Definition at line 12 of file pointLightSystem.cpp.

The documentation for this struct was generated from the following file:

- /home/runner/work/VEngine/VEngine/src/system/pointLightSystem.cpp

6.24 ven::PointLightSystem Class Reference

Class for point light system.

```
#include <PointLightSystem.hpp>
```

Collaboration diagram for ven::PointLightSystem:

Public Member Functions

- PointLightSystem (Device &device, VkRenderPass renderPass, VkDescriptorSetLayout globalSetLayout)
- ~PointLightSystem ()
- PointLightSystem (const PointLightSystem &)=delete
- PointLightSystem & operator= (const PointLightSystem &)=delete
- void render (const FrameInfo &frameInfo) const

Static Public Member Functions

- static void update (const FrameInfo &frameInfo, GlobalUbo &ubo)

Private Member Functions

- void createPipelineLayout (VkDescriptorSetLayout globalSetLayout)
- void createPipeline (VkRenderPass renderPass)

Private Attributes

- Device & m_device
- std::unique_ptr< Shaders > m_shaders
- VkPipelineLayout m_pipelineLayout {nullptr}

6.24.1 Detailed Description

Class for point light system.

Definition at line 22 of file PointLightSystem.hpp.

6.24.2 Constructor & Destructor Documentation

6.24.2.1 PointLightSystem() [1/2]

```
ven::PointLightSystem::PointLightSystem (
            Device & device,
            VkRenderPass renderPass,
            VkDescriptorSetLayout globalSetLayout)  [explicit]
```

Definition at line 15 of file pointLightSystem.cpp.

References createPipeline(), and createPipelineLayout().

Here is the call graph for this function:

6.24.2.2 ~PointLightSystem()

```
ven::PointLightSystem::~PointLightSystem ()  [inline]
```

Definition at line 27 of file PointLightSystem.hpp.

References ven::Device::device(), m_device, and m_pipelineLayout.

Here is the call graph for this function:

6.24.2.3 PointLightSystem() [2/2]

```
ven::PointLightSystem::PointLightSystem (
            const PointLightSystem & )  [delete]
```

6.24.3 Member Function Documentation

6.24.3.1 createPipeline()

```
void ven::PointLightSystem::createPipeline (
            VkRenderPass renderPass)  [private]
```

Definition at line 42 of file pointLightSystem.cpp.

References ven::Shaders::defaultPipelineConfigInfo(), and ven::SHADERS_BIN_PATH.

Referenced by PointLightSystem().

Here is the call graph for this function: Here is the caller graph for this function:

6.24.3.2 createPipelineLayout()

```
void ven::PointLightSystem::createPipelineLayout (
            VkDescriptorSetLayout globalSetLayout)  [private]
```

Definition at line 21 of file pointLightSystem.cpp.

Referenced by PointLightSystem().

Here is the caller graph for this function:

6.24.3.3 operator=()

```
PointLightSystem & ven::PointLightSystem::operator= (
            const PointLightSystem & )  [delete]
```

6.24.3.4 render()

```
void ven::PointLightSystem::render (
            const FrameInfo & frameInfo) const
```

Definition at line 53 of file pointLightSystem.cpp.

References ven::FrameInfo::commandBuffer, ven::FrameInfo::globalDescriptorSet, ven::FrameInfo::objects, and PointLightPushConstants::position.

Referenced by ven::Engine::mainLoop().

Here is the caller graph for this function:

6.24.3.5 update()

```
void ven::PointLightSystem::update (
            const FrameInfo & frameInfo,
            GlobalUbo & ubo)  [static]
```

Definition at line 73 of file pointLightSystem.cpp.

References ven::FrameInfo::frameTime, ven::MAX_LIGHTS, ven::GlobalUbo::numLights, ven::FrameInfo::objects, and ven::GlobalUbo::pointLights.

Referenced by ven::Engine::mainLoop().

Here is the caller graph for this function:

6.24.4 Member Data Documentation

6.24.4.1 m_device

```
Device& ven::PointLightSystem::m_device  [private]
```

Definition at line 40 of file PointLightSystem.hpp.

Referenced by ~PointLightSystem().

6.24.4.2 m_pipelineLayout

```
VkPipelineLayout ven::PointLightSystem::m_pipelineLayout {nullptr}  [private]
```

Definition at line 43 of file PointLightSystem.hpp.

Referenced by ~PointLightSystem().

6.24.4.3 m_shaders

```
std::unique_ptr<Shaders> ven::PointLightSystem::m_shaders  [private]
```

Definition at line 42 of file PointLightSystem.hpp.

The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/System/PointLightSystem.hpp
- /home/runner/work/VEngine/VEngine/src/system/pointLightSystem.cpp

6.25 ven::QueueFamilyIndices Struct Reference

```
#include <Device.hpp>
```

Collaboration diagram for ven::QueueFamilyIndices:

Public Member Functions

- bool isComplete () const

Public Attributes

- uint32_t graphicsFamily {}
- uint32_t presentFamily {}
- bool graphicsFamilyHasValue = false
- bool presentFamilyHasValue = false

6.25.1 Detailed Description

Definition at line 21 of file Device.hpp.

6.25.2 Member Function Documentation

6.25.2.1 isComplete()

```
bool ven::QueueFamilyIndices::isComplete () const  [inline], [nodiscard]
```

Definition at line 26 of file Device.hpp.

References graphicsFamilyHasValue, and presentFamilyHasValue.

Referenced by ven::Device::findQueueFamilies(), and ven::Device::isDeviceSuitable().

Here is the caller graph for this function:

6.25.3 Member Data Documentation

6.25.3.1 graphicsFamily

```
uint32_t ven::QueueFamilyIndices::graphicsFamily {}
```

Definition at line 22 of file Device.hpp.

Referenced by ven::Device::createCommandPool(), ven::Device::createLogicalDevice(), ven::SwapChain::createSwapChain(), and ven::Device::findQueueFamilies().

6.25.3.2 graphicsFamilyHasValue

```
bool ven::QueueFamilyIndices::graphicsFamilyHasValue = false
```

Definition at line 24 of file Device.hpp.

Referenced by ven::Device::findQueueFamilies(), and isComplete().

6.25.3.3 presentFamily

```
uint32_t ven::QueueFamilyIndices::presentFamily {}
```

Definition at line 23 of file Device.hpp.

Referenced by ven::Device::createLogicalDevice(), ven::SwapChain::createSwapChain(), and ven::Device::findQueueFamilies().

6.25.3.4 presentFamilyHasValue

```
bool ven::QueueFamilyIndices::presentFamilyHasValue = false
```

Definition at line 25 of file Device.hpp.

Referenced by ven::Device::findQueueFamilies(), and isComplete().

The documentation for this struct was generated from the following file:

- /home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp

6.26 myLib::Random Class Reference

Class for random number generation.

`#include <Random.hpp>`

Collaboration diagram for myLib::Random:

Static Public Member Functions

- static int randomInt (int min, int max)
  Generate a random integer between min and max.
- static int randomInt ()
- static float randomFloat (float min, float max)
- static float randomFloat ()

6.26.1 Detailed Description

Class for random number generation.

Definition at line 17 of file Random.hpp.

6.26.2 Member Function Documentation

6.26.2.1 randomFloat() [1/2]

```
static float myLib::Random::randomFloat ()  [inline], [static]
```

Definition at line 36 of file Random.hpp.

References randomFloat().

Referenced by randomFloat().

Here is the call graph for this function: Here is the caller graph for this function:

6.26.2.2 randomFloat() [2/2]

```
float myLib::Random::randomFloat (
            float min,
            float max)  [static]
```

Parameters

min	The minimum value
max	The maximum value

Returns

float The random float

Definition at line 10 of file random.cpp.

6.26.2.3 randomInt() [1/2]

```
static int myLib::Random::randomInt ()  [inline], [static]
```

Definition at line 28 of file Random.hpp.

References randomInt().

Referenced by randomInt().

Here is the call graph for this function: Here is the caller graph for this function:

6.26.2.4 randomInt() [2/2]

```
int myLib::Random::randomInt (
            int min,
            int max)  [static]
```

Generate a random integer between min and max.

Parameters

min	The minimum value
max	The maximum value

Returns

int The random integer

Definition at line 3 of file random.cpp.

The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/Random.hpp
- /home/runner/work/VEngine/VEngine/lib/local/static/myLib/src/random.cpp

6.27 ven::Renderer Class Reference

`#include <Renderer.hpp>`

Collaboration diagram for ven::Renderer:

Public Member Functions

- Renderer (Window &window, Device &device)
- ~Renderer ()
- Renderer (const Renderer &)=delete
- Renderer & operator= (const Renderer &)=delete
- VkRenderPass getSwapChainRenderPass () const
- float getAspectRatio () const
- bool isFrameInProgress () const
- VkCommandBuffer getCurrentCommandBuffer () const
- int getFrameIndex () const
- VkCommandBuffer beginFrame ()
- void endFrame ()
- void beginSwapChainRenderPass (VkCommandBuffer commandBuffer) const

Static Public Member Functions

- static void endSwapChainRenderPass (VkCommandBuffer commandBuffer)

Private Member Functions

- void createCommandBuffers ()
- void freeCommandBuffers ()
- void recreateSwapChain ()

Private Attributes

- Window & m_window
- Device & m_device
- std::unique_ptr< SwapChain > m_swapChain
- std::vector< VkCommandBuffer > m_commandBuffers
- uint32_t m_currentImageIndex {0}
- int m_currentFrameIndex {0}
- bool m_isFrameStarted {false}

6.27.1 Detailed Description

Definition at line 20 of file Renderer.hpp.

6.27.2 Constructor & Destructor Documentation

6.27.2.1 Renderer() [1/2]

```
ven::Renderer::Renderer (
            Window & window,
            Device & device)  [inline]
```

Definition at line 24 of file Renderer.hpp.

References createCommandBuffers(), and recreateSwapChain().

Here is the call graph for this function:

6.27.2.2 ~Renderer()

```
ven::Renderer::~Renderer ()  [inline]
```

Definition at line 25 of file Renderer.hpp.

References freeCommandBuffers().

Here is the call graph for this function:

6.27.2.3 Renderer() [2/2]

```
ven::Renderer::Renderer (
            const Renderer & )  [delete]
```

6.27.3 Member Function Documentation

6.27.3.1 beginFrame()

```
VkCommandBuffer ven::Renderer::beginFrame ()
```

Definition at line 43 of file renderer.cpp.

6.27.3.2 beginSwapChainRenderPass()

```
void ven::Renderer::beginSwapChainRenderPass (
            VkCommandBuffer commandBuffer) const
```

Definition at line 90 of file renderer.cpp.

6.27.3.3 createCommandBuffers()

```
void ven::Renderer::createCommandBuffers ()  [private]
```

Definition at line 3 of file renderer.cpp.

References ven::Device::device(), ven::Device::getCommandPool(), m_commandBuffers, m_device, and ven::SwapChain::MAX_FRAMES_IN_FLIGHT.

Referenced by Renderer().

Here is the call graph for this function: Here is the caller graph for this function:

6.27.3.4 endFrame()

```
void ven::Renderer::endFrame ()
```

Definition at line 69 of file renderer.cpp.

References ven::SwapChain::MAX_FRAMES_IN_FLIGHT.

6.27.3.5 endSwapChainRenderPass()

```
void ven::Renderer::endSwapChainRenderPass (
            VkCommandBuffer commandBuffer)  [static]
```

Definition at line 123 of file renderer.cpp.

Referenced by ven::Engine::mainLoop().

Here is the caller graph for this function:

6.27.3.6 freeCommandBuffers()

```
void ven::Renderer::freeCommandBuffers ()  [private]
```

Definition at line 17 of file renderer.cpp.

Referenced by ~Renderer().

Here is the caller graph for this function:

6.27.3.7 getAspectRatio()

```
float ven::Renderer::getAspectRatio () const  [inline], [nodiscard]
```

Definition at line 31 of file Renderer.hpp.

References m_swapChain.

6.27.3.8 getCurrentCommandBuffer()

```
VkCommandBuffer ven::Renderer::getCurrentCommandBuffer () const  [inline], [nodiscard]
```

Definition at line 33 of file Renderer.hpp.

References isFrameInProgress(), m_commandBuffers, and m_currentFrameIndex.

Here is the call graph for this function:

6.27.3.9 getFrameIndex()

```
int ven::Renderer::getFrameIndex () const  [inline], [nodiscard]
```

Definition at line 35 of file Renderer.hpp.

References isFrameInProgress(), and m_currentFrameIndex.

Here is the call graph for this function:

6.27.3.10 getSwapChainRenderPass()

```
VkRenderPass ven::Renderer::getSwapChainRenderPass () const  [inline], [nodiscard]
```

Definition at line 30 of file Renderer.hpp.

References m_swapChain.

6.27.3.11 isFrameInProgress()

```
bool ven::Renderer::isFrameInProgress () const  [inline], [nodiscard]
```

Definition at line 32 of file Renderer.hpp.

References m_isFrameStarted.

Referenced by getCurrentCommandBuffer(), and getFrameIndex().

Here is the caller graph for this function:

6.27.3.12 operator=()

```
Renderer & ven::Renderer::operator= (
            const Renderer & )  [delete]
```

6.27.3.13 recreateSwapChain()

```
void ven::Renderer::recreateSwapChain ()  [private]
```

Definition at line 23 of file renderer.cpp.

Referenced by Renderer().

Here is the caller graph for this function:

6.27.4 Member Data Documentation

6.27.4.1 m_commandBuffers

```
std::vector<VkCommandBuffer> ven::Renderer::m_commandBuffers  [private]
```

Definition at line 51 of file Renderer.hpp.

Referenced by createCommandBuffers(), and getCurrentCommandBuffer().

6.27.4.2 m_currentFrameIndex

```
int ven::Renderer::m_currentFrameIndex {0}  [private]
```

Definition at line 54 of file Renderer.hpp.

Referenced by getCurrentCommandBuffer(), and getFrameIndex().

6.27.4.3 m_currentImageIndex

```
uint32_t ven::Renderer::m_currentImageIndex {0}  [private]
```

Definition at line 53 of file Renderer.hpp.

6.27.4.4 m_device

```
Device& ven::Renderer::m_device  [private]
```

Definition at line 49 of file Renderer.hpp.

Referenced by createCommandBuffers().

6.27.4.5 m_isFrameStarted

```
bool ven::Renderer::m_isFrameStarted {false}  [private]
```

Definition at line 55 of file Renderer.hpp.

Referenced by isFrameInProgress().

6.27.4.6 m_swapChain

```
std::unique_ptr<SwapChain> ven::Renderer::m_swapChain  [private]
```

Definition at line 50 of file Renderer.hpp.

Referenced by getAspectRatio(), and getSwapChainRenderPass().

6.27.4.7 m_window

```
Window& ven::Renderer::m_window  [private]
```

Definition at line 48 of file Renderer.hpp.

The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/Renderer.hpp
- /home/runner/work/VEngine/VEngine/src/renderer.cpp

6.28 ven::RenderSystem Class Reference

Class for render system.

```
#include <RenderSystem.hpp>
```

Collaboration diagram for ven::RenderSystem:

Public Member Functions

- RenderSystem (Device &device, VkRenderPass renderPass, VkDescriptorSetLayout globalSetLayout)
- ~RenderSystem ()
- RenderSystem (const RenderSystem &)=delete
- RenderSystem & operator= (const RenderSystem &)=delete
- void renderObjects (const FrameInfo &frameInfo) const

Private Member Functions

- void createPipelineLayout (VkDescriptorSetLayout globalSetLayout)
- void createPipeline (VkRenderPass renderPass)

Private Attributes

- Device & m_device
- std::unique_ptr< Shaders > m_shaders
- VkPipelineLayout m_pipelineLayout {nullptr}

6.28.1 Detailed Description

Class for render system.

Definition at line 28 of file RenderSystem.hpp.

6.28.2 Constructor & Destructor Documentation

6.28.2.1 RenderSystem() [1/2]

```
ven::RenderSystem::RenderSystem (
            Device & device,
            VkRenderPass renderPass,
            VkDescriptorSetLayout globalSetLayout)  [explicit]
```

Definition at line 5 of file renderSystem.cpp.

References createPipeline(), and createPipelineLayout().

Here is the call graph for this function:

6.28.2.2 ~RenderSystem()

```
ven::RenderSystem::~RenderSystem ()  [inline]
```

Definition at line 33 of file RenderSystem.hpp.

References ven::Device::device(), m_device, and m_pipelineLayout.

Here is the call graph for this function:

6.28.2.3 RenderSystem() [2/2]

```
ven::RenderSystem::RenderSystem (
            const RenderSystem & )  [delete]
```

6.28.3 Member Function Documentation

6.28.3.1 createPipeline()

```
void ven::RenderSystem::createPipeline (
            VkRenderPass renderPass)  [private]
```

Definition at line 32 of file renderSystem.cpp.

References ven::Shaders::defaultPipelineConfigInfo(), and ven::SHADERS_BIN_PATH.

Referenced by RenderSystem().

Here is the call graph for this function: Here is the caller graph for this function:

6.28.3.2 createPipelineLayout()

```
void ven::RenderSystem::createPipelineLayout (
            VkDescriptorSetLayout globalSetLayout)  [private]
```

Definition at line 11 of file renderSystem.cpp.

Referenced by RenderSystem().

Here is the caller graph for this function:

6.28.3.3 operator=()

```
RenderSystem & ven::RenderSystem::operator= (
            const RenderSystem & )  [delete]
```

6.28.3.4 renderObjects()

```
void ven::RenderSystem::renderObjects (
            const FrameInfo & frameInfo) const
```

Definition at line 41 of file renderSystem.cpp.

References ven::FrameInfo::commandBuffer, ven::FrameInfo::globalDescriptorSet, ven::SimplePushConstantData::modelMatrix, and ven::FrameInfo::objects.

Referenced by ven::Engine::mainLoop().

Here is the caller graph for this function:

6.28.4 Member Data Documentation

6.28.4.1 m_device

```
Device& ven::RenderSystem::m_device  [private]
```

Definition at line 45 of file RenderSystem.hpp.

Referenced by ~RenderSystem().

6.28.4.2 m_pipelineLayout

```
VkPipelineLayout ven::RenderSystem::m_pipelineLayout {nullptr}  [private]
```

Definition at line 48 of file RenderSystem.hpp.

Referenced by ~RenderSystem().

6.28.4.3 m_shaders

```
std::unique_ptr<Shaders> ven::RenderSystem::m_shaders  [private]
```

Definition at line 47 of file RenderSystem.hpp.

The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/System/RenderSystem.hpp
- /home/runner/work/VEngine/VEngine/src/system/renderSystem.cpp

6.29 ven::Shaders Class Reference

```
#include <Shaders.hpp>
```

Collaboration diagram for ven::Shaders:

Public Member Functions

- Shaders (Device &device, const std::string &vertFilepath, const std::string &fragFilepath, const PipelineConfigInfo &configInfo)
- ~Shaders ()
- Shaders (const Shaders &)=delete
- Shaders & operator= (const Shaders &)=delete
- void bind (const VkCommandBuffer commandBuffer) const

Static Public Member Functions

- static void defaultPipelineConfigInfo (PipelineConfigInfo &configInfo)

Private Member Functions

- void createGraphicsPipeline (const std::string &vertFilepath, const std::string &fragFilepath, const PipelineConfigInfo &configInfo)
- void createShaderModule (const std::vector< char > &code, VkShaderModule *shaderModule) const

Static Private Member Functions

- static std::vector< char > readFile (const std::string &filename)

Private Attributes

- Device & m_device
- VkPipeline m_graphicsPipeline {nullptr}
- VkShaderModule m_vertShaderModule {nullptr}
- VkShaderModule m_fragShaderModule {nullptr}

6.29.1 Detailed Description

Definition at line 39 of file Shaders.hpp.

6.29.2 Constructor & Destructor Documentation

6.29.2.1 Shaders() [1/2]

```
ven::Shaders::Shaders (
            Device & device,
            const std::string & vertFilepath,
            const std::string & fragFilepath,
            const PipelineConfigInfo & configInfo)  [inline]
```

Definition at line 43 of file Shaders.hpp.

References createGraphicsPipeline().

Here is the call graph for this function:

6.29.2.2 ~Shaders()

```
ven::Shaders::~Shaders ()
```

Definition at line 6 of file shaders.cpp.

References ven::Device::device(), m_device, m_fragShaderModule, m_graphicsPipeline, and m_vertShaderModule.

Here is the call graph for this function:

6.29.2.3 Shaders() [2/2]

```
ven::Shaders::Shaders (
            const Shaders & )  [delete]
```

6.29.3 Member Function Documentation

6.29.3.1 bind()

```
void ven::Shaders::bind (
            const VkCommandBuffer commandBuffer) const  [inline]
```

Definition at line 50 of file Shaders.hpp.

References m_graphicsPipeline.

6.29.3.2 createGraphicsPipeline()

```
void ven::Shaders::createGraphicsPipeline (
            const std::string & vertFilepath,
            const std::string & fragFilepath,
            const PipelineConfigInfo & configInfo)  [private]
```

Definition at line 31 of file shaders.cpp.

References ven::PipelineConfigInfo::attributeDescriptions, ven::PipelineConfigInfo::bindingDescriptions, ven::PipelineConfigInfo::color
ven::PipelineConfigInfo::depthStencilInfo, ven::PipelineConfigInfo::dynamicStateInfo, ven::PipelineConfigInfo::inputAssemblyInfo,
ven::PipelineConfigInfo::multisampleInfo, ven::PipelineConfigInfo::pipelineLayout, ven::PipelineConfigInfo::rasterizationInfo,
ven::PipelineConfigInfo::renderPass, and ven::PipelineConfigInfo::subpass.

Referenced by Shaders().

Here is the caller graph for this function:

6.29.3.3 createShaderModule()

```
void ven::Shaders::createShaderModule (
            const std::vector< char > & code,
            VkShaderModule * shaderModule) const  [private]
```

Definition at line 100 of file shaders.cpp.

6.29.3.4 defaultPipelineConfigInfo()

```
void ven::Shaders::defaultPipelineConfigInfo (
            PipelineConfigInfo & configInfo)  [static]
```

Definition at line 112 of file shaders.cpp.

References ven::PipelineConfigInfo::attributeDescriptions, ven::PipelineConfigInfo::bindingDescriptions, ven::PipelineConfigInfo::color
ven::PipelineConfigInfo::colorBlendInfo, ven::PipelineConfigInfo::depthStencilInfo, ven::PipelineConfigInfo::dynamicStateEnables,
ven::PipelineConfigInfo::dynamicStateInfo, ven::Model::Vertex::getAttributeDescriptions(), ven::Model::Vertex::getBindingDescriptions
ven::PipelineConfigInfo::inputAssemblyInfo, ven::PipelineConfigInfo::multisampleInfo, and ven::PipelineConfigInfo::rasterizationInfo.

Referenced by ven::PointLightSystem::createPipeline(), and ven::RenderSystem::createPipeline().

Here is the call graph for this function: Here is the caller graph for this function:

6.29.3.5 operator=()

```
Shaders & ven::Shaders::operator= (
            const Shaders & )  [delete]
```

6.29.3.6 readFile()

```
std::vector< char > ven::Shaders::readFile (
            const std::string & filename)  [static], [private]
```

Definition at line 13 of file shaders.cpp.

6.29.4 Member Data Documentation

6.29.4.1 m_device

```
Device& ven::Shaders::m_device  [private]
```

Definition at line 58 of file Shaders.hpp.

Referenced by ~Shaders().

6.29.4.2 m_fragShaderModule

```
VkShaderModule ven::Shaders::m_fragShaderModule {nullptr}  [private]
```

Definition at line 61 of file Shaders.hpp.

Referenced by ~Shaders().

6.29.4.3 m_graphicsPipeline

```
VkPipeline ven::Shaders::m_graphicsPipeline {nullptr}  [private]
```

Definition at line 59 of file Shaders.hpp.

Referenced by bind(), and ~Shaders().

6.29.4.4 m_vertShaderModule

```
VkShaderModule ven::Shaders::m_vertShaderModule {nullptr}  [private]
```

Definition at line 60 of file Shaders.hpp.

Referenced by ~Shaders().

The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/Shaders.hpp
- /home/runner/work/VEngine/VEngine/src/shaders.cpp

6.30 ven::SimplePushConstantData Struct Reference

```
#include <RenderSystem.hpp>
```

Collaboration diagram for ven::SimplePushConstantData:

Public Attributes

- glm::mat4 modelMatrix {1.F}
- glm::mat4 normalMatrix {1.F}

6.30.1 Detailed Description

Definition at line 19 of file RenderSystem.hpp.

6.30.2 Member Data Documentation

6.30.2.1 modelMatrix

```
glm::mat4 ven::SimplePushConstantData::modelMatrix {1.F}
```

Definition at line 20 of file RenderSystem.hpp.

Referenced by ven::RenderSystem::renderObjects().

6.30.2.2 normalMatrix

```
glm::mat4 ven::SimplePushConstantData::normalMatrix {1.F}
```

Definition at line 21 of file RenderSystem.hpp.

The documentation for this struct was generated from the following file:

- /home/runner/work/VEngine/VEngine/include/VEngine/System/RenderSystem.hpp

6.31 ven::SwapChain Class Reference

```
#include <SwapChain.hpp>
```

Collaboration diagram for ven::SwapChain:

Public Member Functions

- SwapChain (Device &deviceRef, const VkExtent2D windowExtentRef)
- SwapChain (Device &deviceRef, const VkExtent2D windowExtentRef, std::shared_ptr< SwapChain > previous)
- ~SwapChain ()
- SwapChain (const SwapChain &)=delete
- SwapChain & operator= (const SwapChain &)=delete
- VkFramebuffer getFrameBuffer (const unsigned long index) const
- VkRenderPass getRenderPass () const
- VkImageView getImageView (const int index) const
- size_t imageCount () const
- VkFormat getSwapChainImageFormat () const
- VkExtent2D getSwapChainExtent () const
- uint32_t width () const
- uint32_t height () const
- float extentAspectRatio () const
- VkFormat findDepthFormat () const
- VkResult acquireNextImage (uint32_t *imageIndex) const
- VkResult submitCommandBuffers (const VkCommandBuffer *buffers, const uint32_t *imageIndex)
- bool compareSwapFormats (const SwapChain &swapChainp) const

Static Public Attributes

- static constexpr int MAX_FRAMES_IN_FLIGHT = 2

Private Member Functions

- void init ()
- void createSwapChain ()
- void createImageViews ()
- void createDepthResources ()
- void createRenderPass ()
- void createFramebuffers ()
- void createSyncObjects ()
- VkExtent2D chooseSwapExtent (const VkSurfaceCapabilitiesKHR &capabilities) const

Static Private Member Functions

- static VkSurfaceFormatKHR chooseSwapSurfaceFormat (const std::vector< VkSurfaceFormatKHR > &availableFormats)
- static VkPresentModeKHR chooseSwapPresentMode (const std::vector< VkPresentModeKHR > &availablePresentModes)

Private Attributes

- VkFormat swapChainImageFormat {}
- VkFormat swapChainDepthFormat {}
- VkExtent2D m_swapChainExtent {}
- std::vector< VkFramebuffer > swapChainFramebuffers
- VkRenderPass renderPass {}
- std::vector< VkImage > depthImages
- std::vector< VkDeviceMemory > depthImageMemorys
- std::vector< VkImageView > depthImageViews
- std::vector< VkImage > swapChainImages
- std::vector< VkImageView > swapChainImageViews
- Device & device
- VkExtent2D windowExtent
- VkSwapchainKHR swapChain {}
- std::shared_ptr< SwapChain > oldSwapChain
- std::vector< VkSemaphore > imageAvailableSemaphores
- std::vector< VkSemaphore > renderFinishedSemaphores
- std::vector< VkFence > inFlightFences
- std::vector< VkFence > imagesInFlight
- size_t currentFrame = 0

6.31.1 Detailed Description

Definition at line 16 of file SwapChain.hpp.

6.31.2 Constructor & Destructor Documentation

6.31.2.1 SwapChain() [1/3]

```
ven::SwapChain::SwapChain (
            Device & deviceRef,
            const VkExtent2D windowExtentRef)  [inline]
```

Definition at line 22 of file SwapChain.hpp.

References init().

Here is the call graph for this function:

6.31.2.2 SwapChain() [2/3]

```
ven::SwapChain::SwapChain (
            Device & deviceRef,
            const VkExtent2D windowExtentRef,
            std::shared_ptr< SwapChain > previous)  [inline]
```

Definition at line 23 of file SwapChain.hpp.

References init(), and oldSwapChain.

Here is the call graph for this function:

6.31.2.3 ~SwapChain()

```
ven::SwapChain::~SwapChain ()
```

Definition at line 7 of file swapChain.cpp.

References depthImageMemorys, depthImages, depthImageViews, ven::Device::device(), device, imageAvailableSemaphores, inFlightFences, MAX_FRAMES_IN_FLIGHT, renderFinishedSemaphores, renderPass, swapChain, swapChainFramebuffers, and swapChainImageViews.

Here is the call graph for this function:

6.31.2.4 SwapChain() [3/3]

```
ven::SwapChain::SwapChain (
            const SwapChain & )  [delete]
```

6.31.3 Member Function Documentation

6.31.3.1 acquireNextImage()

```
VkResult ven::SwapChain::acquireNextImage (
            uint32_t * imageIndex) const
```

Definition at line 49 of file swapChain.cpp.

6.31.3.2 chooseSwapExtent()

```
VkExtent2D ven::SwapChain::chooseSwapExtent (
            const VkSurfaceCapabilitiesKHR & capabilities) const  [private]
```

Definition at line 366 of file swapChain.cpp.

6.31.3.3 chooseSwapPresentMode()

```
VkPresentModeKHR ven::SwapChain::chooseSwapPresentMode (
            const std::vector< VkPresentModeKHR > & availablePresentModes)  [static], [private]
```

Definition at line 346 of file swapChain.cpp.

6.31.3.4 chooseSwapSurfaceFormat()

```
VkSurfaceFormatKHR ven::SwapChain::chooseSwapSurfaceFormat (
            const std::vector< VkSurfaceFormatKHR > & availableFormats)  [static], [private]
```

Definition at line 335 of file swapChain.cpp.

6.31.3.5 compareSwapFormats()

```
bool ven::SwapChain::compareSwapFormats (
            const SwapChain & swapChainp) const  [inline], [nodiscard]
```

Definition at line 44 of file SwapChain.hpp.

References swapChainDepthFormat, and swapChainImageFormat.

6.31.3.6 createDepthResources()

```
void ven::SwapChain::createDepthResources ()  [private]
```

Definition at line 266 of file swapChain.cpp.

6.31.3.7 createFramebuffers()

```
void ven::SwapChain::createFramebuffers ()  [private]
```

Definition at line 244 of file swapChain.cpp.

6.31.3.8 createImageViews()

```
void ven::SwapChain::createImageViews ()  [private]
```

Definition at line 164 of file swapChain.cpp.

6.31.3.9 createRenderPass()

```
void ven::SwapChain::createRenderPass ()  [private]
```

Definition at line 185 of file swapChain.cpp.

6.31.3.10 createSwapChain()

```
void ven::SwapChain::createSwapChain ()  [private]
```

Definition at line 103 of file swapChain.cpp.

References ven::SwapChainSupportDetails::capabilities, ven::SwapChainSupportDetails::formats, ven::QueueFamilyIndices::graphics
ven::QueueFamilyIndices::presentFamily, and ven::SwapChainSupportDetails::presentModes.

6.31.3.11 createSyncObjects()

```
void ven::SwapChain::createSyncObjects ()  [private]
```

Definition at line 312 of file swapChain.cpp.

6.31.3.12 extentAspectRatio()

```
float ven::SwapChain::extentAspectRatio () const  [inline], [nodiscard]
```

Definition at line 38 of file SwapChain.hpp.

References m_swapChainExtent.

6.31.3.13 findDepthFormat()

```
VkFormat ven::SwapChain::findDepthFormat () const
```

Definition at line 378 of file swapChain.cpp.

6.31.3.14 getFrameBuffer()

```
VkFramebuffer ven::SwapChain::getFrameBuffer (
            const unsigned long index) const  [inline], [nodiscard]
```

Definition at line 29 of file SwapChain.hpp.

References swapChainFramebuffers.

6.31.3.15 getImageView()

```
VkImageView ven::SwapChain::getImageView (
            const int index) const  [inline], [nodiscard]
```

Definition at line 31 of file SwapChain.hpp.

References swapChainImageViews.

6.31.3.16 getRenderPass()

```
VkRenderPass ven::SwapChain::getRenderPass () const  [inline], [nodiscard]
```

Definition at line 30 of file SwapChain.hpp.

References renderPass.

6.31.3.17 getSwapChainExtent()

```
VkExtent2D ven::SwapChain::getSwapChainExtent () const  [inline], [nodiscard]
```

Definition at line 34 of file SwapChain.hpp.

References m_swapChainExtent.

6.31.3.18 getSwapChainImageFormat()

```
VkFormat ven::SwapChain::getSwapChainImageFormat () const  [inline], [nodiscard]
```

Definition at line 33 of file SwapChain.hpp.

References swapChainImageFormat.

6.31.3.19 height()

```
uint32_t ven::SwapChain::height () const  [inline], [nodiscard]
```

Definition at line 36 of file SwapChain.hpp.

References m_swapChainExtent.

6.31.3.20 imageCount()

```
size_t ven::SwapChain::imageCount () const  [inline], [nodiscard]
```

Definition at line 32 of file SwapChain.hpp.

References swapChainImages.

6.31.3.21 init()

```
void ven::SwapChain::init ()  [private]
```

Definition at line 39 of file swapChain.cpp.

Referenced by SwapChain(), and SwapChain().

Here is the caller graph for this function:

6.31.3.22 operator=()

```
SwapChain & ven::SwapChain::operator= (
            const SwapChain & )  [delete]
```

6.31.3.23 submitCommandBuffers()

```
VkResult ven::SwapChain::submitCommandBuffers (
            const VkCommandBuffer * buffers,
            const uint32_t * imageIndex)
```

Definition at line 56 of file swapChain.cpp.

6.31.3.24 width()

```
uint32_t ven::SwapChain::width () const  [inline], [nodiscard]
```

Definition at line 35 of file SwapChain.hpp.

References m_swapChainExtent.

6.31.4 Member Data Documentation

6.31.4.1 currentFrame

```
size_t ven::SwapChain::currentFrame = 0  [private]
```

Definition at line 85 of file SwapChain.hpp.

6.31.4.2 depthImageMemorys

```
std::vector<VkDeviceMemory> ven::SwapChain::depthImageMemorys  [private]
```

Definition at line 70 of file SwapChain.hpp.

Referenced by ~SwapChain().

6.31.4.3 depthImages

```
std::vector<VkImage> ven::SwapChain::depthImages  [private]
```

Definition at line 69 of file SwapChain.hpp.

Referenced by ~SwapChain().

6.31.4.4 depthImageViews

```
std::vector<VkImageView> ven::SwapChain::depthImageViews  [private]
```

Definition at line 71 of file SwapChain.hpp.

Referenced by ~SwapChain().

6.31.4.5 device

```
Device& ven::SwapChain::device  [private]
```

Definition at line 75 of file SwapChain.hpp.

Referenced by ~SwapChain().

6.31.4.6 imageAvailableSemaphores

```
std::vector<VkSemaphore> ven::SwapChain::imageAvailableSemaphores  [private]
```

Definition at line 81 of file SwapChain.hpp.

Referenced by ~SwapChain().

6.31.4.7 imagesInFlight

```
std::vector<VkFence> ven::SwapChain::imagesInFlight  [private]
```

Definition at line 84 of file SwapChain.hpp.

6.31.4.8 inFlightFences

```
std::vector<VkFence> ven::SwapChain::inFlightFences  [private]
```

Definition at line 83 of file SwapChain.hpp.

Referenced by ~SwapChain().

6.31.4.9 m_swapChainExtent

```
VkExtent2D ven::SwapChain::m_swapChainExtent {}  [private]
```

Definition at line 64 of file SwapChain.hpp.

Referenced by extentAspectRatio(), getSwapChainExtent(), height(), and width().

6.31.4.10 MAX_FRAMES_IN_FLIGHT

```
int ven::SwapChain::MAX_FRAMES_IN_FLIGHT = 2  [static], [constexpr]
```

Definition at line 20 of file SwapChain.hpp.

Referenced by ven::Renderer::createCommandBuffers(), ven::Renderer::endFrame(), ven::Engine::Engine(), ven::Engine::mainLoop(), and ~SwapChain().

6.31.4.11 oldSwapChain

```
std::shared_ptr<SwapChain> ven::SwapChain::oldSwapChain  [private]
```

Definition at line 79 of file SwapChain.hpp.

Referenced by SwapChain().

6.31.4.12 renderFinishedSemaphores

```
std::vector<VkSemaphore> ven::SwapChain::renderFinishedSemaphores  [private]
```

Definition at line 82 of file SwapChain.hpp.

Referenced by ~SwapChain().

6.31.4.13 renderPass

```
VkRenderPass ven::SwapChain::renderPass {}  [private]
```

Definition at line 67 of file SwapChain.hpp.

Referenced by getRenderPass(), and ~SwapChain().

6.31.4.14 swapChain

```
VkSwapchainKHR ven::SwapChain::swapChain {}  [private]
```

Definition at line 78 of file SwapChain.hpp.

Referenced by ~SwapChain().

6.31.4.15 swapChainDepthFormat

```
VkFormat ven::SwapChain::swapChainDepthFormat {}  [private]
```

Definition at line 63 of file SwapChain.hpp.

Referenced by compareSwapFormats().

6.31.4.16 swapChainFramebuffers

```
std::vector<VkFramebuffer> ven::SwapChain::swapChainFramebuffers  [private]
```

Definition at line 66 of file SwapChain.hpp.

Referenced by getFrameBuffer(), and ~SwapChain().

6.31.4.17 swapChainImageFormat

```
VkFormat ven::SwapChain::swapChainImageFormat {}  [private]
```

Definition at line 62 of file SwapChain.hpp.

Referenced by compareSwapFormats(), and getSwapChainImageFormat().

6.31.4.18 swapChainImages

```
std::vector<VkImage> ven::SwapChain::swapChainImages  [private]
```

Definition at line 72 of file SwapChain.hpp.

Referenced by imageCount().

6.31.4.19 swapChainImageViews

```
std::vector<VkImageView> ven::SwapChain::swapChainImageViews  [private]
```

Definition at line 73 of file SwapChain.hpp.

Referenced by getImageView(), and ~SwapChain().

6.31.4.20 windowExtent

```
VkExtent2D ven::SwapChain::windowExtent  [private]
```

Definition at line 76 of file SwapChain.hpp.

The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/SwapChain.hpp
- /home/runner/work/VEngine/VEngine/src/swapChain.cpp

6.32 ven::SwapChainSupportDetails Struct Reference

`#include <Device.hpp>`

Collaboration diagram for ven::SwapChainSupportDetails:

Public Attributes

- VkSurfaceCapabilitiesKHR capabilities
- std::vector< VkSurfaceFormatKHR > formats
- std::vector< VkPresentModeKHR > presentModes

6.32.1 Detailed Description

Definition at line 15 of file Device.hpp.

6.32.2 Member Data Documentation

6.32.2.1 capabilities

`VkSurfaceCapabilitiesKHR ven::SwapChainSupportDetails::capabilities`

Definition at line 16 of file Device.hpp.

Referenced by ven::SwapChain::createSwapChain(), and ven::Device::querySwapChainSupport().

6.32.2.2 formats

`std::vector<VkSurfaceFormatKHR> ven::SwapChainSupportDetails::formats`

Definition at line 17 of file Device.hpp.

Referenced by ven::SwapChain::createSwapChain(), ven::Device::isDeviceSuitable(), and ven::Device::querySwapChainSupport().

6.32.2.3 presentModes

`std::vector<VkPresentModeKHR> ven::SwapChainSupportDetails::presentModes`

Definition at line 18 of file Device.hpp.

Referenced by ven::SwapChain::createSwapChain(), ven::Device::isDeviceSuitable(), and ven::Device::querySwapChainSupport().

The documentation for this struct was generated from the following file:

- /home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp

6.33 myLib::Time Class Reference

Class used for time management.

`#include <Time.hpp>`

Collaboration diagram for myLib::Time:

Public Member Functions

- Time (const double seconds)
  Construct a new Time object.
- int asSeconds () const
  Transform the time to seconds.
- int asMilliseconds () const
  Transform the time to milliseconds.
- int asMicroseconds () const
  Transform the time to microseconds.

Private Attributes

- double m_seconds {0.0F}

6.33.1 Detailed Description

Class used for time management.

Definition at line 15 of file Time.hpp.

6.33.2 Constructor & Destructor Documentation

6.33.2.1 Time()

```
myLib::Time::Time (
            const double seconds)  [inline], [explicit]
```

Construct a new Time object.

Definition at line 22 of file Time.hpp.

6.33.3 Member Function Documentation

6.33.3.1 asMicroseconds()

```
int myLib::Time::asMicroseconds () const  [inline], [nodiscard]
```

Transform the time to microseconds.

Returns

int The time in microseconds

Definition at line 40 of file Time.hpp.

References m_seconds.

6.33.3.2 asMilliseconds()

```
int myLib::Time::asMilliseconds () const  [inline], [nodiscard]
```

Transform the time to milliseconds.

Returns

int The time in milliseconds

Definition at line 34 of file Time.hpp.

References m_seconds.

6.33.3.3 asSeconds()

```
int myLib::Time::asSeconds () const  [inline], [nodiscard]
```

Transform the time to seconds.

Returns

int The time in seconds

Definition at line 28 of file Time.hpp.

References m_seconds.

6.33.4 Member Data Documentation

6.33.4.1 m_seconds

```
double myLib::Time::m_seconds {0.0F}  [private]
```

Definition at line 47 of file Time.hpp.

Referenced by asMicroseconds(), asMilliseconds(), and asSeconds().

The documentation for this class was generated from the following file:

- /home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/Clock/Time.hpp

6.34 ven::Transform3DComponent Struct Reference

```
#include <Object.hpp>
```

Collaboration diagram for ven::Transform3DComponent:

Public Member Functions

- glm::mat4 mat4 () const
- glm::mat3 normalMatrix () const

Public Attributes

- glm::vec3 translation {}
- glm::vec3 scale {1.F, 1.F, 1.F}
- glm::vec3 rotation {}

6.34.1 Detailed Description

Definition at line 20 of file Object.hpp.

6.34.2 Member Function Documentation

6.34.2.1 mat4()

```
glm::mat4 ven::Transform3DComponent::mat4 () const  [nodiscard]
```

Definition at line 3 of file object.cpp.

References rotation, scale, and translation.

6.34.2.2 normalMatrix()

```
glm::mat3 ven::Transform3DComponent::normalMatrix () const  [nodiscard]
```

Definition at line 38 of file object.cpp.

6.34.3 Member Data Documentation

6.34.3.1 rotation

```
glm::vec3 ven::Transform3DComponent::rotation {}
```

Definition at line 23 of file Object.hpp.

Referenced by ven::Engine::mainLoop(), and mat4().

6.34.3.2 scale

```
glm::vec3 ven::Transform3DComponent::scale {1.F, 1.F, 1.F}
```

Definition at line 22 of file Object.hpp.

Referenced by ven::Engine::loadObjects(), ven::Object::makePointLight(), and mat4().

6.34.3.3 translation

```
glm::vec3 ven::Transform3DComponent::translation {}
```

Definition at line 21 of file Object.hpp.

Referenced by ven::Engine::loadObjects(), ven::Engine::mainLoop(), and mat4().

The documentation for this struct was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/Object.hpp
- /home/runner/work/VEngine/VEngine/src/object.cpp

6.35 ven::Model::Vertex Struct Reference

```
#include <Model.hpp>
```

Collaboration diagram for ven::Model::Vertex:

Public Member Functions

- bool operator== (const Vertex &other) const

Static Public Member Functions

- static std::vector< VkVertexInputBindingDescription > getBindingDescriptions ()
- static std::vector< VkVertexInputAttributeDescription > getAttributeDescriptions ()

Public Attributes

- glm::vec3 position {}
- glm::vec3 color {}
- glm::vec3 normal {}
- glm::vec2 uv {}

6.35.1 Detailed Description

Definition at line 20 of file Model.hpp.

6.35.2 Member Function Documentation

6.35.2.1 getAttributeDescriptions()

```
std::vector< VkVertexInputAttributeDescription > ven::Model::Vertex::getAttributeDescriptions
() [static]
```

Definition at line 108 of file model.cpp.

Referenced by ven::Shaders::defaultPipelineConfigInfo().

Here is the caller graph for this function:

6.35.2.2 getBindingDescriptions()

```
std::vector< VkVertexInputBindingDescription > ven::Model::Vertex::getBindingDescriptions ()
[static]
```

Definition at line 99 of file model.cpp.

Referenced by ven::Shaders::defaultPipelineConfigInfo().

Here is the caller graph for this function:

6.35.2.3 operator==()

```
bool ven::Model::Vertex::operator== (
            const Vertex & other) const  [inline]
```

Definition at line 29 of file Model.hpp.

References color, normal, position, and uv.

6.35.3 Member Data Documentation

6.35.3.1 color

```
glm::vec3 ven::Model::Vertex::color {}
```

Definition at line 22 of file Model.hpp.

Referenced by std::hash< ven::Model::Vertex >::operator()(), and operator==().

6.35.3.2 normal

```
glm::vec3 ven::Model::Vertex::normal {}
```

Definition at line 23 of file Model.hpp.

Referenced by std::hash< ven::Model::Vertex >::operator()(), and operator==().

6.35.3.3 position

```
glm::vec3 ven::Model::Vertex::position {}
```

Definition at line 21 of file Model.hpp.

Referenced by ven::Model::Builder::loadModel(), std::hash< ven::Model::Vertex >::operator()(), and operator==().

6.35.3.4 uv

```
glm::vec2 ven::Model::Vertex::uv {}
```

Definition at line 24 of file Model.hpp.

Referenced by std::hash< ven::Model::Vertex >::operator()(), and operator==().

The documentation for this struct was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp
- /home/runner/work/VEngine/VEngine/src/model.cpp

6.36 ven::Window Class Reference

```
#include <Window.hpp>
```

Collaboration diagram for ven::Window:

Public Member Functions

- Window (const uint32_t width, const uint32_t height, const std::string &title)
- ~Window ()
- GLFWwindow * createWindow (uint32_t width, uint32_t height, const std::string &title)
- void createWindowSurface (VkInstance instance, VkSurfaceKHR *surface) const
- GLFWwindow * getGLFWindow () const
- VkExtent2D getExtent () const
- bool wasWindowResized () const
- void resetWindowResizedFlag ()

Static Private Member Functions

- static void framebufferResizeCallback (GLFWwindow *window, int width, int height)

Private Attributes

- GLFWwindow * m_window {nullptr}
- uint32_t m_width
- uint32_t m_height
- bool m_framebufferResized = false

6.36.1 Detailed Description

Definition at line 17 of file Window.hpp.

6.36.2 Constructor & Destructor Documentation

6.36.2.1 Window()

```
ven::Window::Window (
            const uint32_t width,
            const uint32_t height,
            const std::string & title)  [inline]
```

Definition at line 21 of file Window.hpp.

6.36.2.2 ~Window()

```
ven::Window::~Window ()  [inline]
```

Definition at line 22 of file Window.hpp.

References m_window.

6.36.3 Member Function Documentation

6.36.3.1 createWindow()

```
GLFWwindow * ven::Window::createWindow (
            uint32_t width,
            uint32_t height,
            const std::string & title)  [nodiscard]
```

Definition at line 5 of file window.cpp.

References framebufferResizeCallback().

Here is the call graph for this function:

6.36.3.2 createWindowSurface()

```
void ven::Window::createWindowSurface (
            VkInstance instance,
            VkSurfaceKHR * surface) const
```

Definition at line 24 of file window.cpp.

Referenced by ven::Device::createSurface().

Here is the caller graph for this function:

6.36.3.3 framebufferResizeCallback()

```
void ven::Window::framebufferResizeCallback (
            GLFWwindow * window,
            int width,
            int height)  [static], [private]
```

Definition at line 31 of file window.cpp.

References m_framebufferResized.

Referenced by createWindow().

Here is the caller graph for this function:

6.36.3.4 getExtent()

```
VkExtent2D ven::Window::getExtent () const  [inline], [nodiscard]
```

Definition at line 29 of file Window.hpp.

References m_height, and m_width.

6.36.3.5 getGLFWindow()

```
GLFWwindow * ven::Window::getGLFWindow () const  [inline], [nodiscard]
```

Definition at line 27 of file Window.hpp.

References m_window.

Referenced by ven::Engine::createSurface().

Here is the caller graph for this function:

6.36.3.6 resetWindowResizedFlag()

```
void ven::Window::resetWindowResizedFlag ()  [inline]
```

Definition at line 31 of file Window.hpp.

References m_framebufferResized.

6.36.3.7 wasWindowResized()

```
bool ven::Window::wasWindowResized () const  [inline], [nodiscard]
```

Definition at line 30 of file Window.hpp.

References m_framebufferResized.

6.36.4 Member Data Documentation

6.36.4.1 m_framebufferResized

```
bool ven::Window::m_framebufferResized = false  [private]
```

Definition at line 41 of file Window.hpp.

Referenced by framebufferResizeCallback(), resetWindowResizedFlag(), and wasWindowResized().

6.36.4.2 m_height

```
uint32_t ven::Window::m_height  [private]
```

Definition at line 39 of file Window.hpp.

Referenced by getExtent().

6.36.4.3 m_width

```
uint32_t ven::Window::m_width  [private]
```

Definition at line 38 of file Window.hpp.

Referenced by getExtent().

6.36.4.4 m_window

```
GLFWwindow* ven::Window::m_window {nullptr}  [private]
```

Definition at line 37 of file Window.hpp.

Referenced by getGLFWindow(), and ~Window().

The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/Window.hpp
- /home/runner/work/VEngine/VEngine/src/window.cpp

Chapter 7

File Documentation

7.1 /home/runner/work/VEngine/VEngine/include/VEngine/Buffer.hpp File Reference

This file contains the Buffer class.

`#include "VEngine/Device.hpp"`
Include dependency graph for Buffer.hpp:

7.2 Buffer.hpp

Go to the documentation of this file.

```
00001 ///
00002 /// @file Buffer.hpp
00003 /// @brief This file contains the Buffer class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include "VEngine/Device.hpp"
00010
00011 namespace ven {
00012
00013     ///
00014     /// @class Buffer
00015     /// @brief Class for buffer
00016     /// @namespace ven
00017     class Buffer {
00018
00019         public:
00020
00021             Buffer(Device& device, VkDeviceSize instanceSize, uint32_t instanceCount,
      VkBufferUsageFlags usageFlags, VkMemoryPropertyFlags memoryPropertyFlags, VkDeviceSize
      minOffsetAlignment = 1);
00022             ~Buffer();
00023
00024             Buffer(const Buffer&) = delete;
00025             Buffer& operator=(const Buffer&) = delete;
00026
00027             ///
00028             /// @brief Map a memory range of this buffer. If successful, mapped points to the
      specified buffer range.
00029             ///
00030             /// @param size (Optional) Size of the memory range to map. Pass VK_WHOLE_SIZE to map the
      complete buffer range.
00031             /// @param offset (Optional) Byte offset from beginning
00032             ///
00033             /// @return VkResult of the buffer mapping call
00034             ///
```

```
00035             VkResult map(VkDeviceSize size = VK_WHOLE_SIZE, VkDeviceSize offset = 0);
00036
00037             ///
00038             /// @brief Unmap a mapped memory range
00039             ///
00040             /// @note Does not return a result as vkUnmapMemory can't fail
00041             ///
00042             void unmap();
00043
00044             ///
00045             /// @brief Copies the specified data to the mapped buffer. Default value writes whole
      buffer range
00046             ///
00047             /// @param data Pointer to the data to copy
00048             /// @param size (Optional) Size of the data to copy. Pass VK_WHOLE_SIZE to flush the
      complete buffer range.
00049             /// @param offset (Optional) Byte offset from beginning of mapped region
00050             ///
00051             void writeToBuffer(const void* data, VkDeviceSize size = VK_WHOLE_SIZE, VkDeviceSize
      offset = 0) const;
00052
00053             ///
00054             /// @brief Flush a memory range of the buffer to make it visible to the device
00055             ///
00056             ///  @note Only required for non-coherent memory
00057             ///
00058             /// @param size (Optional) Size of the memory range to flush. Pass VK_WHOLE_SIZE to flush
      the complete buffer range.
00059             /// @param offset (Optional) Byte offset from beginning
00060             ///
00061             /// @return VkResult of the flush call
00062             ///
00063             [[nodiscard]] VkResult flush(VkDeviceSize size = VK_WHOLE_SIZE, VkDeviceSize offset = 0)
      const;
00064
00065             ///
00066             /// @brief Create a buffer info descriptor
00067             ///
00068             /// @param size (Optional) Size of the memory range of the descriptor
00069             /// @param offset (Optional) Byte offset from beginning
00070             ///
00071             /// @return VkDescriptorBufferInfo of specified offset and range
00072             ///
00073             [[nodiscard]] VkDescriptorBufferInfo descriptorInfo(const VkDeviceSize size =
      VK_WHOLE_SIZE, const VkDeviceSize offset = 0) const { return VkDescriptorBufferInfo{m_buffer, offset,
      size, }; }
00074
00075             ///
00076             /// @brief Invalidate a memory range of the buffer to make it visible to the host
00077             ///
00078             /// @note Only required for non-coherent memory
00079             ///
00080             /// @param size (Optional) Size of the memory range to invalidate. Pass VK_WHOLE_SIZE to
      invalidate
00081             /// the complete buffer range.
00082             /// @param offset (Optional) Byte offset from beginning
00083             ///
00084             /// @return VkResult of the invalidate call
00085             ///
00086             [[nodiscard]] VkResult invalidate(VkDeviceSize size = VK_WHOLE_SIZE, VkDeviceSize offset =
      0) const;
00087
00088             ///
00089             /// Copies "instanceSize" bytes of data to the mapped buffer at an offset of index *
      alignmentSize
00090             ///
00091             /// @param data Pointer to the data to copy
00092             /// @param index Used in offset calculation
00093             ///
00094             ///
00095             void writeToIndex(const void* data, const VkDeviceSize index) const { writeToBuffer(data,
      m_instanceSize, index * m_alignmentSize); }
00096
00097             ///
00098             ///  Flush the memory range at index * alignmentSize of the buffer to make it visible to
      the device
00099             ///
00100             /// @param index Used in offset calculation
00101             ///
00102             [[nodiscard]] VkResult flushIndex(const VkDeviceSize index) const { return
      flush(m_alignmentSize, index * m_alignmentSize); }
00103
00104             ///
00105             ///
00106             /// Create a buffer info descriptor
00107             ///
00108             /// @param index Specifies the region given by index * alignmentSize
```

```
00109             ///
00110             /// @return VkDescriptorBufferInfo for instance at index
00111             ///
00112             [[nodiscard]] VkDescriptorBufferInfo descriptorInfoForIndex(const VkDeviceSize index)
      const { return descriptorInfo(m_alignmentSize, index * m_alignmentSize); }
00113
00114             ///
00115             /// Invalidate a memory range of the buffer to make it visible to the host
00116             ///
00117             /// @note Only required for non-coherent memory
00118             ///
00119             /// @param index Specifies the region to invalidate: index * alignmentSize
00120             ///
00121             /// @return VkResult of the invalidate call
00122             ///
00123             [[nodiscard]] VkResult invalidateIndex(const VkDeviceSize index) const { return
      invalidate(m_alignmentSize, index * m_alignmentSize); }
00124
00125             [[nodiscard]] VkBuffer getBuffer() const { return m_buffer; }
00126             [[nodiscard]] void* getMappedMemory() const { return m_mapped; }
00127             [[nodiscard]] uint32_t getInstanceCount() const { return m_instanceCount; }
00128             [[nodiscard]] VkDeviceSize getInstanceSize() const { return m_instanceSize; }
00129             [[nodiscard]] VkDeviceSize getAlignmentSize() const { return m_instanceSize; }
00130             [[nodiscard]] VkBufferUsageFlags getUsageFlags() const { return m_usageFlags; }
00131             [[nodiscard]] VkMemoryPropertyFlags getMemoryPropertyFlags() const { return
      m_memoryPropertyFlags; }
00132             [[nodiscard]] VkDeviceSize getBufferSize() const { return m_bufferSize; }
00133
00134         private:
00135             ///
00136             /// Returns the minimum instance size required to be compatible with devices
      minOffsetAlignment
00137             ///
00138             /// @param instanceSize The size of an instance
00139             /// @param minOffsetAlignment The minimum required alignment, in bytes, for the offset
      member (eg
00140             /// minUniformBufferOffsetAlignment)
00141             ///
00142             /// @return VkResult of the buffer mapping call
00143             ///
00144             static VkDeviceSize getAlignment(VkDeviceSize instanceSize, VkDeviceSize
      minOffsetAlignment);
00145
00146             Device& m_device;
00147             void* m_mapped = nullptr;
00148             VkBuffer m_buffer = VK_NULL_HANDLE;
00149             VkDeviceMemory m_memory = VK_NULL_HANDLE;
00150
00151             VkDeviceSize m_bufferSize;
00152             VkDeviceSize m_instanceSize;
00153             uint32_t m_instanceCount;
00154             VkDeviceSize m_alignmentSize;
00155             VkBufferUsageFlags m_usageFlags;
00156             VkMemoryPropertyFlags m_memoryPropertyFlags;
00157
00158     }; // class Buffer
00159
00160 } // namespace ven
```

7.3 /home/runner/work/VEngine/VEngine/include/VEngine/Camera.hpp File Reference

This file contains the Camera class.

`#include <glm/glm.hpp>`
Include dependency graph for Camera.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class ven::Camera

Namespaces

- namespace ven

Macros

- #define GLM_FORCE_RADIANS
- #define GLM_FORCE_DEPTH_ZERO_TO_ONE

7.3.1 Detailed Description

This file contains the Camera class.

This file contains the KeyboardController class.

Definition in file Camera.hpp.

7.3.2 Macro Definition Documentation

7.3.2.1 GLM_FORCE_DEPTH_ZERO_TO_ONE

```
#define GLM_FORCE_DEPTH_ZERO_TO_ONE
```

Definition at line 10 of file Camera.hpp.

7.3.2.2 GLM_FORCE_RADIANS

```
#define GLM_FORCE_RADIANS
```

Definition at line 9 of file Camera.hpp.

7.4 Camera.hpp

Go to the documentation of this file.

```
00001 ///
00002 /// @file Camera.hpp
00003 /// @brief This file contains the Camera class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #define GLM_FORCE_RADIANS
00010 #define GLM_FORCE_DEPTH_ZERO_TO_ONE
00011 #include <glm/glm.hpp>
00012
00013 namespace ven {
00014
00015 ///
00016
00017     class Camera {
00018
00019         public:
00020
00021             void setOrthographicProjection(float left, float right, float top, float bottom, float
      near, float far);
00022             void setPerspectiveProjection(float fovy, float aspect, float near, float far);
00023             void setViewDirection(glm::vec3 position, glm::vec3 direction, glm::vec3 up =
      glm::vec3{0.F, -1.F, 0.F});
00024             void setViewTarget(glm::vec3 position, glm::vec3 target, glm::vec3 up = glm::vec3{0.F,
      -1.F, 0.F}) { setViewDirection(position, target - position, up); }
00025             void setViewYXZ(glm::vec3 position, glm::vec3 rotation);
00026
00027             [[nodiscard]] const glm::mat4& getProjection() const { return m_projectionMatrix; }
00028             [[nodiscard]] const glm::mat4& getView() const { return m_viewMatrix; }
00029             [[nodiscard]] const glm::mat4& getInverseView() const { return m_inverseViewMatrix; }
00030
00031         private:
00032
00033             glm::mat4 m_projectionMatrix{1.F};
00034             glm::mat4 m_viewMatrix{1.F};
00035             glm::mat4 m_inverseViewMatrix{1.F};
00036
00037     }; // class Camera
00038
00039 } // namespace ven
```

7.5 /home/runner/work/VEngine/VEngine/include/VEngine/Constant.hpp File Reference

This file contains the constant values used in the project.

This graph shows which files directly or indirectly include this file:

Namespaces

- namespace ven

Typedefs

- using ven::return_type_t

Variables

- static constexpr uint32_t ven::DEFAULT_WIDTH = 1920
- static constexpr uint32_t ven::DEFAULT_HEIGHT = 1080
- static constexpr std::string_view ven::DEFAULT_TITLE = "VEngine"
- static constexpr std::string_view ven::SHADERS_BIN_PATH = "shaders/bin/"

7.5.1 Detailed Description

This file contains the constant values used in the project.

Definition in file Constant.hpp.

7.6 Constant.hpp

Go to the documentation of this file.

```
00001 ///
00002 /// @file Constant.hpp
00003 /// @brief This file contains the constant values used in the project
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 namespace ven {
00010
00011     static constexpr uint32_t DEFAULT_WIDTH = 1920;
00012     static constexpr uint32_t DEFAULT_HEIGHT = 1080;
00013
00014     static constexpr std::string_view DEFAULT_TITLE = "VEngine";
00015     static constexpr std::string_view SHADERS_BIN_PATH = "shaders/bin/";
00016
00017     using return_type_t = enum ReturnType : uint8_t {
00018         VEN_SUCCESS = 0,
00019         VEN_FAILURE = 1
00020     };
00021
00022 } // namespace ven
```

7.7 /home/runner/work/VEngine/VEngine/include/VEngine/↩ Descriptors.hpp File Reference

This file contains the Descriptors class.

```
#include <memory>
#include <unordered_map>
#include "VEngine/Device.hpp"
```

Include dependency graph for Descriptors.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class ven::DescriptorSetLayout

  Class for descriptor set layout.
- class ven::DescriptorSetLayout::Builder
- class ven::DescriptorPool

  Class for descriptor pool.
- class ven::DescriptorPool::Builder
- class ven::DescriptorWriter

  Class for descriptor writer.

Namespaces

- namespace ven

7.7.1 Detailed Description

This file contains the Descriptors class.

Definition in file Descriptors.hpp.

7.8 Descriptors.hpp

Go to the documentation of this file.

```
00001 ///
00002 /// @file Descriptors.hpp
00003 /// @brief This file contains the Descriptors class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <memory>
00010 #include <unordered_map>
00011
00012 #include "VEngine/Device.hpp"
00013
00014 namespace ven {
00015
00016 ///
00017     /// @class DescriptorSetLayout
00018     /// @brief Class for descriptor set layout
00019     /// @namespace ven
00020     ///
00021     class DescriptorSetLayout {
00022
00023         public:
```

```
00024
00025             class Builder {
00026
00027                 public:
00028
00029                     explicit Builder(Device &device) : m_device{device} {}
00030
00031                     Builder &addBinding(uint32_t binding, VkDescriptorType descriptorType,
      VkShaderStageFlags stageFlags, uint32_t count = 1);
00032                     std::unique_ptr<DescriptorSetLayout> build() const { return
      std::make_unique<DescriptorSetLayout>(m_device, m_bindings); }
00033
00034                 private:
00035                     Device &m_device;
00036                     std::unordered_map<uint32_t, VkDescriptorSetLayoutBinding> m_bindings;
00037             };
00038
00039             DescriptorSetLayout(Device &device, const std::unordered_map<uint32_t,
      VkDescriptorSetLayoutBinding>& bindings);
00040             ~DescriptorSetLayout() { vkDestroyDescriptorSetLayout(m_device.device(),
      m_descriptorSetLayout, nullptr); }
00041             DescriptorSetLayout(const DescriptorSetLayout &) = delete;
00042             DescriptorSetLayout &operator=(const DescriptorSetLayout &) = delete;
00043
00044             VkDescriptorSetLayout getDescriptorSetLayout() const { return m_descriptorSetLayout; }
00045
00046         private:
00047
00048             Device &m_device;
00049             VkDescriptorSetLayout m_descriptorSetLayout;
00050             std::unordered_map<uint32_t, VkDescriptorSetLayoutBinding> m_bindings;
00051
00052             friend class DescriptorWriter;
00053
00054     }; // class DescriptorSetLayout
00055
00056     ///
00057     /// @class DescriptorPool
00058     /// @brief Class for descriptor pool
00059     /// @namespace ven
00060     ///
00061     class DescriptorPool {
00062
00063         public:
00064
00065             class Builder {
00066
00067                 public:
00068
00069                     explicit Builder(Device &device) : m_device{device} {}
00070
00071                     Builder &addPoolSize(VkDescriptorType descriptorType, uint32_t count);
00072                     Builder &setPoolFlags(VkDescriptorPoolCreateFlags flags);
00073                     Builder &setMaxSets(uint32_t count);
00074                     [[nodiscard]] std::unique_ptr<DescriptorPool> build() const { return
      std::make_unique<DescriptorPool>(m_device, m_maxSets, m_poolFlags, m_poolSizes); }
00075
00076                 private:
00077
00078                     Device &m_device;
00079                     std::vector<VkDescriptorPoolSize> m_poolSizes;
00080                     uint32_t m_maxSets = 1000;
00081                     VkDescriptorPoolCreateFlags m_poolFlags = 0;
00082             };
00083
00084             DescriptorPool(Device &device, uint32_t maxSets, VkDescriptorPoolCreateFlags poolFlags,
      const std::vector<VkDescriptorPoolSize> &poolSizes);
00085             ~DescriptorPool() { vkDestroyDescriptorPool(m_device.device(), m_descriptorPool, nullptr);
      }
00086             DescriptorPool(const DescriptorPool &) = delete;
00087             DescriptorPool &operator=(const DescriptorPool &) = delete;
00088
00089             bool allocateDescriptor(VkDescriptorSetLayout descriptorSetLayout, VkDescriptorSet
      &descriptor) const;
00090
00091             void freeDescriptors(const std::vector<VkDescriptorSet> &descriptors) const {
      vkFreeDescriptorSets(m_device.device(), m_descriptorPool, static_cast<uint32_t>(descriptors.size()),
      descriptors.data()); }
00092
00093             void resetPool() const { vkResetDescriptorPool(m_device.device(), m_descriptorPool, 0); }
00094
00095         private:
00096
00097             Device &m_device;
00098             VkDescriptorPool m_descriptorPool;
00099
00100             friend class DescriptorWriter;
```

```
00101
00102     }; // class DescriptorPool
00103
00104     ///
00105     /// @class DescriptorWriter
00106     /// @brief Class for descriptor writer
00107     /// @namespace ven
00108     ///
00109     class DescriptorWriter {
00110
00111         public:
00112
00113             DescriptorWriter(DescriptorSetLayout &setLayout, DescriptorPool &pool) :
      m_setLayout{setLayout}, m_pool{pool} {}
00114
00115             DescriptorWriter &writeBuffer(uint32_t binding, const VkDescriptorBufferInfo *bufferInfo);
00116             DescriptorWriter &writeImage(uint32_t binding, const VkDescriptorImageInfo *imageInfo);
00117
00118             bool build(VkDescriptorSet &set);
00119             void overwrite(const VkDescriptorSet &set);
00120
00121         private:
00122
00123             DescriptorSetLayout &m_setLayout;
00124             DescriptorPool &m_pool;
00125             std::vector<VkWriteDescriptorSet> m_writes;
00126
00127     }; // class DescriptorWriter
00128
00129 } // namespace ven
```

7.9 /home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp File Reference

This file contains the Device class.

```
#include <vector>
#include "VEngine/Window.hpp"
```

Include dependency graph for Device.hpp: This graph shows which files directly or indirectly include this file:

Classes

- struct ven::SwapChainSupportDetails
- struct ven::QueueFamilyIndices
- class ven::Device

Namespaces

- namespace ven

7.9.1 Detailed Description

This file contains the Device class.

Definition in file Device.hpp.

7.10 Device.hpp

Go to the documentation of this file.

```
00001 ///
00002 /// @file Device.hpp
00003 /// @brief This file contains the Device class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <vector>
00010
00011 #include "VEngine/Window.hpp"
00012
00013 namespace ven {
00014
00015     struct SwapChainSupportDetails {
00016         VkSurfaceCapabilitiesKHR capabilities;
00017         std::vector<VkSurfaceFormatKHR> formats;
00018         std::vector<VkPresentModeKHR> presentModes;
00019     };
00020
00021     struct QueueFamilyIndices {
00022         uint32_t graphicsFamily{};
00023         uint32_t presentFamily{};
00024         bool graphicsFamilyHasValue = false;
00025         bool presentFamilyHasValue = false;
00026         [[nodiscard]] bool isComplete() const { return graphicsFamilyHasValue &&
      presentFamilyHasValue; }
00027     };
00028
00029     class Device {
00030
00031         public:
00032
00033             #ifdef NDEBUG
00034                 const bool enableValidationLayers = false;
00035             #else
00036                 const bool enableValidationLayers = true;
00037             #endif
00038
00039             explicit Device(Window &window);
00040             ~Device();
00041
00042             Device(const Device &) = delete;
00043             Device& operator=(const Device &) = delete;
00044             Device(Device &&) = delete;
00045             Device &operator=(Device &&) = delete;
00046
00047             [[nodiscard]] VkCommandPool getCommandPool() const { return commandPool; }
00048             [[nodiscard]] VkDevice device() const { return device_; }
00049             [[nodiscard]] VkSurfaceKHR surface() const { return surface_; }
00050             [[nodiscard]] VkQueue graphicsQueue() const { return graphicsQueue_; }
00051             [[nodiscard]] VkQueue presentQueue() const { return presentQueue_; }
00052
00053         [[nodiscard]] SwapChainSupportDetails getSwapChainSupport() const { return
      querySwapChainSupport(physicalDevice); }
00054         [[nodiscard]] uint32_t findMemoryType(uint32_t typeFilter, VkMemoryPropertyFlags propertiesp)
      const;
00055         [[nodiscard]] QueueFamilyIndices findPhysicalQueueFamilies() const { return
      findQueueFamilies(physicalDevice); }
00056         [[nodiscard]] VkFormat findSupportedFormat(const std::vector<VkFormat> &candidates,
      VkImageTiling tiling, VkFormatFeatureFlags features) const;
00057
00058             // Buffer Helper Functions
00059             void createBuffer(VkDeviceSize size, VkBufferUsageFlags usage, VkMemoryPropertyFlags
      propertiesp, VkBuffer &buffer, VkDeviceMemory &bufferMemory) const;
00060             [[nodiscard]] VkCommandBuffer beginSingleTimeCommands() const;
00061             void endSingleTimeCommands(VkCommandBuffer commandBuffer) const;
00062             void copyBuffer(VkBuffer srcBuffer, VkBuffer dstBuffer, VkDeviceSize size) const;
00063             void copyBufferToImage(VkBuffer buffer, VkImage image, uint32_t width, uint32_t height,
      uint32_t layerCount) const;
00064
00065             void createImageWithInfo(const VkImageCreateInfo &imageInfo, VkMemoryPropertyFlags
      properties, VkImage &image, VkDeviceMemory &imageMemory) const;
00066
00067             VkPhysicalDeviceProperties m_properties;
00068
00069             [[nodiscard]] VkPhysicalDevice getPhysicalDevice() const { return physicalDevice; }
00070             [[nodiscard]] VkQueue getGraphicsQueue() const { return graphicsQueue_; }
00071
00072         private:
00073
00074             void createInstance();
```

```
00075             void setupDebugMessenger();
00076             void createSurface() { m_window.createWindowSurface(instance, &surface_); };
00077             void pickPhysicalDevice();
00078             void createLogicalDevice();
00079             void createCommandPool();
00080 
00081             // helper functions
00082             bool isDeviceSuitable(VkPhysicalDevice device) const;
00083             [[nodiscard]] std::vector<const char *> getRequiredExtensions() const;
00084             [[nodiscard]] bool checkValidationLayerSupport() const;
00085             QueueFamilyIndices findQueueFamilies(VkPhysicalDevice device) const;
00086             static void populateDebugMessengerCreateInfo(VkDebugUtilsMessengerCreateInfoEXT
      &createInfo);
00087             void hasGlfwRequiredInstanceExtensions() const;
00088             bool checkDeviceExtensionSupport(VkPhysicalDevice device) const;
00089             SwapChainSupportDetails querySwapChainSupport(VkPhysicalDevice device) const;
00090 
00091             VkInstance instance;
00092             VkDebugUtilsMessengerEXT debugMessenger;
00093             VkPhysicalDevice physicalDevice = VK_NULL_HANDLE;
00094             Window &m_window;
00095             VkCommandPool commandPool;
00096 
00097             VkDevice device_;
00098             VkSurfaceKHR surface_;
00099             VkQueue graphicsQueue_;
00100             VkQueue presentQueue_;
00101 
00102             const std::vector<const char *> validationLayers = {"VK_LAYER_KHRONOS_validation"};
00103             const std::vector<const char *> deviceExtensions = {VK_KHR_SWAPCHAIN_EXTENSION_NAME};
00104 
00105     }; // class Device
00106 
00107 }  // namespace ven
```

7.11 /home/runner/work/VEngine/VEngine/include/VEngine/Engine.hpp File Reference

This file contains the Engine class.

```
#include <vulkan/vulkan.h>
#include "VEngine/Window.hpp"
#include "VEngine/Constant.hpp"
#include "VEngine/Device.hpp"
#include "VEngine/Object.hpp"
#include "VEngine/Renderer.hpp"
#include "VEngine/Descriptors.hpp"
```

Include dependency graph for Engine.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class ven::Engine

Namespaces

- namespace ven

7.11.1 Detailed Description

This file contains the Engine class.

Definition in file Engine.hpp.

7.12 Engine.hpp

Go to the documentation of this file.

```
00001 ///
00002 /// @file Engine.hpp
00003 /// @brief This file contains the Engine class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <vulkan/vulkan.h>
00010
00011 #include "VEngine/Window.hpp"
00012 #include "VEngine/Constant.hpp"
00013 #include "VEngine/Device.hpp"
00014 #include "VEngine/Object.hpp"
00015 #include "VEngine/Renderer.hpp"
00016 #include "VEngine/Descriptors.hpp"
00017
00018 namespace ven {
00019
00020     class Engine {
00021
00022     public:
00023
00024         explicit Engine(uint32_t = DEFAULT_WIDTH, uint32_t = DEFAULT_HEIGHT, const std::string &title
      = DEFAULT_TITLE.data());
00025         ~Engine() = default;
00026
00027         Engine(const Engine &) = delete;
00028         Engine operator=(const Engine &) = delete;
00029
00030         Window &getWindow() { return m_window; };
00031
00032         void mainLoop();
00033
00034     private:
00035
00036         void loadObjects();
00037
00038         Window m_window;
00039         Device m_device{m_window};
00040         Renderer m_renderer{m_window, m_device};
00041
00042         std::unique_ptr<DescriptorPool> m_globalPool;
00043         Object::Map m_objects;
00044
00045         VkInstance m_instance{nullptr};
00046         VkSurfaceKHR m_surface{nullptr};
00047
00048         void createInstance();
00049         void createSurface() { if (glfwCreateWindowSurface(m_instance, m_window.getGLFWindow(),
      nullptr, &m_surface) != VK_SUCCESS) { throw std::runtime_error("Failed to create window surface"); } }
00050
00051     }; // class Engine
00052
00053 } // namespace ven
```

7.13 /home/runner/work/VEngine/VEngine/include/VEngine/FrameCounter.hpp File Reference

This file contains the FrameCounter class.

`#include <iostream>`
Include dependency graph for FrameCounter.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class ven::FrameCounter

Namespaces

- namespace ven

7.13.1 Detailed Description

This file contains the FrameCounter class.

Definition in file FrameCounter.hpp.

7.14 FrameCounter.hpp

Go to the documentation of this file.

```
00001 ///
00002 /// @file FrameCounter.hpp
00003 /// @brief This file contains the FrameCounter class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <iostream>
00010
00011 namespace ven {
00012
00013     class FrameCounter {
00014
00015         public:
00016
00017             FrameCounter() = default;
00018             ~FrameCounter() = default;
00019
00020             void update(const float deltaTime) {
00021                 m_frameCounter += 1.F;
00022                 m_timeCounter += deltaTime;
00023
00024                 if (m_timeCounter >= 1.F) {
00025                     std::cout « "FPS: " « m_frameCounter « '\n';
00026                     m_fps = m_frameCounter;
00027                     m_frameTime = 1000.F / m_fps;
00028                     m_frameCounter = 0.F;
00029                     m_timeCounter = 0.F;
00030                 }
00031             }
00032
00033             [[nodiscard]] float getFps() const { return m_fps; }
00034             [[nodiscard]] float getFrameTime() const { return m_frameTime; }
00035
00036         private:
00037
00038             float m_fps{0.F};
00039             float m_frameTime{0.F};
00040             float m_frameCounter{0.F};
00041             float m_timeCounter{0.F};
00042
00043     }; // class FrameCounter
00044
00045 } // namespace ven
```

7.15 /home/runner/work/VEngine/VEngine/include/VEngine/FrameInfo.hpp File Reference

This file contains the FrameInfo class.

```
#include <vulkan/vulkan.h>
#include "VEngine/Camera.hpp"
#include "VEngine/Object.hpp"
```

Include dependency graph for FrameInfo.hpp: This graph shows which files directly or indirectly include this file:

Classes

- struct ven::PointLight
- struct ven::GlobalUbo
- struct ven::FrameInfo

Namespaces

- namespace ven

Variables

- static constexpr std::size_t ven::MAX_LIGHTS = 10

7.15.1 Detailed Description

This file contains the FrameInfo class.

Definition in file FrameInfo.hpp.

7.16 FrameInfo.hpp

Go to the documentation of this file.

```
00001 ///
00002 /// @file FrameInfo.hpp
00003 /// @brief This file contains the FrameInfo class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <vulkan/vulkan.h>
00010
00011 #include "VEngine/Camera.hpp"
00012 #include "VEngine/Object.hpp"
00013
00014 namespace ven {
00015
00016 static constexpr std::size_t MAX_LIGHTS = 10;
00017
00018     struct PointLight
00019     {
00020         glm::vec4 position{};
00021         glm::vec4 color{};
00022     };
00023
00024     struct GlobalUbo
00025     {
00026         glm::mat4 projection{1.F};
00027         glm::mat4 view{1.F};
00028         glm::mat4 inverseView{1.F};
00029         glm::vec4 ambientLightColor{1.F, 1.F, 1.F, .02F};
00030         std::array<PointLight, MAX_LIGHTS> pointLights;
00031         int numLights;
00032     };
00033
00034     struct FrameInfo
00035     {
00036         int frameIndex;
00037         float frameTime;
00038         VkCommandBuffer commandBuffer;
00039         Camera &camera;
00040         VkDescriptorSet globalDescriptorSet;
00041         Object::Map &objects;
00042     };
00043
00044 } // namespace ven
```

7.17 /home/runner/work/VEngine/VEngine/include/VEngine/KeyboardController.hpp File Reference

```
#include "VEngine/Window.hpp"
#include "VEngine/Object.hpp"
```

Include dependency graph for KeyboardController.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class ven::KeyboardController
- struct ven::KeyboardController::KeyMappings

Namespaces

- namespace ven

7.18 KeyboardController.hpp

Go to the documentation of this file.

```
00001 ///
00002 /// @file Camera.hpp
00003 /// @brief This file contains the KeyboardController class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include "VEngine/Window.hpp"
00010 #include "VEngine/Object.hpp"
00011
00012 namespace ven {
00013
00014     class KeyboardController {
00015
00016         public:
00017
00018             struct KeyMappings {
00019                 int moveLeft = GLFW_KEY_A;
00020                 int moveRight = GLFW_KEY_D;
00021                 int moveForward = GLFW_KEY_W;
00022                 int moveBackward = GLFW_KEY_S;
00023                 int moveUp = GLFW_KEY_SPACE;
00024                 int moveDown = GLFW_KEY_LEFT_SHIFT;
00025                 int lookLeft = GLFW_KEY_LEFT;
00026                 int lookRight = GLFW_KEY_RIGHT;
00027                 int lookUp = GLFW_KEY_UP;
00028                 int lookDown = GLFW_KEY_DOWN;
00029             };
00030
00031             void moveInPlaneXZ(GLFWwindow* window, float dt, Object& object) const;
00032
00033             KeyMappings m_keys{};
00034             float m_moveSpeed{3.F};
00035             float m_lookSpeed{1.5F};
00036
00037     }; // class KeyboardController
00038
00039 } // namespace ven
```

7.19 /home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp File Reference

This file contains the Model class.

```
#include <memory>
#include "VEngine/Device.hpp"
#include "VEngine/Buffer.hpp"
```

Include dependency graph for Model.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class ven::Model
- struct ven::Model::Vertex
- struct ven::Model::Builder

Namespaces

- namespace ven

7.19.1 Detailed Description

This file contains the Model class.

Definition in file Model.hpp.

7.20 Model.hpp

Go to the documentation of this file.

```
00001 ///
00002 /// @file Model.hpp
00003 /// @brief This file contains the Model class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <memory>
00010
00011 #include "VEngine/Device.hpp"
00012 #include "VEngine/Buffer.hpp"
00013
00014 namespace ven {
00015
00016     class Model {
00017
00018         public:
00019
00020             struct Vertex {
00021                 glm::vec3 position{};
00022                 glm::vec3 color{};
00023                 glm::vec3 normal{};
00024                 glm::vec2 uv{};
00025
00026                 static std::vector<VkVertexInputBindingDescription> getBindingDescriptions();
00027                 static std::vector<VkVertexInputAttributeDescription> getAttributeDescriptions();
00028
00029                 bool operator==(const Vertex& other) const {
00030                     return position == other.position && color == other.color && normal ==
      other.normal && uv == other.uv;
00031                 }
```

```
00032             };
00033
00034             struct Builder {
00035                 std::vector<Vertex> vertices;
00036                 std::vector<uint32_t> indices;
00037
00038                 void loadModel(const std::string &filename);
00039             };
00040
00041             Model(Device &device, const Builder &builder);
00042             ~Model();
00043
00044             Model(const Model&) = delete;
00045             void operator=(const Model&) = delete;
00046
00047             static std::unique_ptr<Model> createModelFromFile(Device &device, const std::string
      &filename);
00048
00049             void bind(VkCommandBuffer commandBuffer) const;
00050             void draw(VkCommandBuffer commandBuffer) const;
00051
00052         private:
00053
00054             void createVertexBuffer(const std::vector<Vertex>& vertices);
00055             void createIndexBuffer(const std::vector<uint32_t>& indices);
00056
00057             Device& m_device;
00058             std::unique_ptr<Buffer> m_vertexBuffer;
00059             uint32_t m_vertexCount;
00060
00061             bool m_hasIndexBuffer{false};
00062             std::unique_ptr<Buffer> m_indexBuffer;
00063             uint32_t m_indexCount;
00064
00065     }; // class Model
00066
00067 } // namespace ven
```

7.21 /home/runner/work/VEngine/VEngine/include/VEngine/Object.hpp File Reference

This file contains the Object class.

```
#include <memory>
#include <unordered_map>
#include <glm/gtc/matrix_transform.hpp>
#include "VEngine/Model.hpp"
```

Include dependency graph for Object.hpp: This graph shows which files directly or indirectly include this file:

Classes

- struct ven::Transform3DComponent
- struct ven::PointLightComponent
- class ven::Object

Namespaces

- namespace ven

Typedefs

- using ven::id_t = unsigned int

7.21.1 Detailed Description

This file contains the Object class.

Definition in file Object.hpp.

7.22 Object.hpp

Go to the documentation of this file.

```
00001 ///
00002 /// @file Object.hpp
00003 /// @brief This file contains the Object class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <memory>
00010 #include <unordered_map>
00011
00012 #include <glm/gtc/matrix_transform.hpp>
00013
00014 #include "VEngine/Model.hpp"
00015
00016 namespace ven {
00017
00018     using id_t = unsigned int;
00019
00020     struct Transform3DComponent {
00021         glm::vec3 translation{};
00022         glm::vec3 scale{1.F, 1.F, 1.F};
00023         glm::vec3 rotation{};
00024
00025         [[nodiscard]] glm::mat4 mat4() const;
00026         [[nodiscard]] glm::mat3 normalMatrix() const;
00027     };
00028
00029     struct PointLightComponent {
00030         float lightIntensity = 1.0F;
00031     };
00032
00033     class Object {
00034
00035         public:
00036
00037          using Map = std::unordered_map<id_t, Object>;
00038
00039
00040             static Object createObject() { static id_t objId = 0; return Object(objId++); }
00041
00042             ~Object() = default;
00043
00044             static Object makePointLight(float intensity = 10.F, float radius = 0.1F, glm::vec3 color
      = glm::vec3(1.F));
00045
00046             Object(const Object&) = delete;
00047             Object& operator=(const Object&) = delete;
00048             Object(Object&&) = default;
00049             Object& operator=(Object&&) = default;
00050
00051             [[nodiscard]] id_t getId() const { return m_objId; }
00052
00053             std::shared_ptr<Model> model{};
00054             glm::vec3 color{};
00055             Transform3DComponent transform3D{};
00056
00057             std::unique_ptr<PointLightComponent> pointLight = nullptr;
00058
00059     private:
00060             explicit Object(const id_t objId) : m_objId(objId) {}
00061
00062             id_t m_objId;
00063
00064     }; // class Object
00065
00066 } // namespace ven
```

7.23 /home/runner/work/VEngine/VEngine/include/VEngine/↩ Renderer.hpp File Reference

This file contains the Renderer class.

```
#include <memory>
#include <cassert>
#include <vulkan/vulkan.h>
#include "VEngine/Window.hpp"
#include "VEngine/Device.hpp"
#include "VEngine/SwapChain.hpp"
```

Include dependency graph for Renderer.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class ven::Renderer

Namespaces

- namespace ven

7.23.1 Detailed Description

This file contains the Renderer class.

Definition in file Renderer.hpp.

7.24 Renderer.hpp

Go to the documentation of this file.

```
00001 ///
00002 /// @file Renderer.hpp
00003 /// @brief This file contains the Renderer class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <memory>
00010 #include <cassert>
00011
00012 #include <vulkan/vulkan.h>
00013
00014 #include "VEngine/Window.hpp"
00015 #include "VEngine/Device.hpp"
00016 #include "VEngine/SwapChain.hpp"
00017
00018 namespace ven {
00019
00020     class Renderer {
00021
00022     public:
00023
00024         Renderer(Window &window, Device &device) : m_window{window}, m_device{device} {
      recreateSwapChain(); createCommandBuffers(); }
00025         ~Renderer() { freeCommandBuffers(); }
00026
00027         Renderer(const Renderer &) = delete;
00028         Renderer& operator=(const Renderer &) = delete;
00029
```

```
00030         [[nodiscard]] VkRenderPass getSwapChainRenderPass() const { return
      m_swapChain->getRenderPass(); }
00031         [[nodiscard]] float getAspectRatio() const { return m_swapChain->extentAspectRatio(); }
00032         [[nodiscard]] bool isFrameInProgress() const { return m_isFrameStarted; }
00033         [[nodiscard]] VkCommandBuffer getCurrentCommandBuffer() const { assert(isFrameInProgress() &&
      "cannot get command m_buffer when frame not in progress"); return
      m_commandBuffers[static_cast<unsigned long>(m_currentFrameIndex)]; }
00034
00035         [[nodiscard]] int getFrameIndex() const { assert(isFrameInProgress() && "cannot get frame
      index when frame not in progress"); return m_currentFrameIndex; }
00036
00037         VkCommandBuffer beginFrame();
00038         void endFrame();
00039         void beginSwapChainRenderPass(VkCommandBuffer commandBuffer) const;
00040         static void endSwapChainRenderPass(VkCommandBuffer commandBuffer);
00041
00042     private:
00043
00044         void createCommandBuffers();
00045         void freeCommandBuffers();
00046         void recreateSwapChain();
00047
00048         Window &m_window;
00049         Device &m_device;
00050         std::unique_ptr<SwapChain> m_swapChain;
00051         std::vector<VkCommandBuffer> m_commandBuffers;
00052
00053         uint32_t m_currentImageIndex{0};
00054         int m_currentFrameIndex{0};
00055         bool m_isFrameStarted{false};
00056
00057     }; // class Renderer
00058
00059 } // namespace ven
```

7.25 /home/runner/work/VEngine/VEngine/include/VEngine/Shaders.hpp File Reference

This file contains the Shader class.

```
#include <string>
#include <vulkan/vulkan.h>
#include <glm/glm.hpp>
#include "VEngine/Device.hpp"
#include "VEngine/Model.hpp"
```

Include dependency graph for Shaders.hpp: This graph shows which files directly or indirectly include this file:

Classes

- struct ven::PipelineConfigInfo
- class ven::Shaders

Namespaces

- namespace ven

7.25.1 Detailed Description

This file contains the Shader class.

Definition in file Shaders.hpp.

7.26 Shaders.hpp

Go to the documentation of this file.

```
00001 ///
00002 /// @file Shaders.hpp
00003 /// @brief This file contains the Shader class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <string>
00010
00011 #include <vulkan/vulkan.h>
00012 #include <glm/glm.hpp>
00013
00014 #include "VEngine/Device.hpp"
00015 #include "VEngine/Model.hpp"
00016
00017 namespace ven {
00018
00019     struct PipelineConfigInfo {
00020         PipelineConfigInfo() = default;
00021         PipelineConfigInfo(const PipelineConfigInfo&) = delete;
00022         PipelineConfigInfo& operator=(const PipelineConfigInfo&) = delete;
00023
00024         std::vector<VkVertexInputBindingDescription> bindingDescriptions;
00025         std::vector<VkVertexInputAttributeDescription> attributeDescriptions;
00026         VkPipelineInputAssemblyStateCreateInfo inputAssemblyInfo{};
00027         VkPipelineRasterizationStateCreateInfo rasterizationInfo{};
00028         VkPipelineMultisampleStateCreateInfo multisampleInfo{};
00029         VkPipelineColorBlendAttachmentState colorBlendAttachment{};
00030         VkPipelineColorBlendStateCreateInfo colorBlendInfo{};
00031         VkPipelineDepthStencilStateCreateInfo depthStencilInfo{};
00032         std::vector<VkDynamicState> dynamicStateEnables;
00033         VkPipelineDynamicStateCreateInfo dynamicStateInfo{};
00034         VkPipelineLayout pipelineLayout = nullptr;
00035         VkRenderPass renderPass = nullptr;
00036         uint32_t subpass = 0;
00037     };
00038
00039     class Shaders {
00040
00041         public:
00042
00043             Shaders(Device &device, const std::string& vertFilepath, const std::string& fragFilepath,
      const PipelineConfigInfo& configInfo) : m_device{device} { createGraphicsPipeline(vertFilepath,
      fragFilepath, configInfo); };
00044             ~Shaders();
00045
00046             Shaders(const Shaders&) = delete;
00047             Shaders& operator=(const Shaders&) = delete;
00048
00049             static void defaultPipelineConfigInfo(PipelineConfigInfo& configInfo);
00050             void bind(const VkCommandBuffer commandBuffer) const { vkCmdBindPipeline(commandBuffer,
      VK_PIPELINE_BIND_POINT_GRAPHICS, m_graphicsPipeline); }
00051
00052         private:
00053
00054             static std::vector<char> readFile(const std::string &filename);
00055             void createGraphicsPipeline(const std::string& vertFilepath, const std::string&
      fragFilepath, const PipelineConfigInfo& configInfo);
00056             void createShaderModule(const std::vector<char>& code, VkShaderModule* shaderModule)
      const;
00057
00058             Device& m_device;
00059             VkPipeline m_graphicsPipeline{nullptr};
00060             VkShaderModule m_vertShaderModule{nullptr};
00061             VkShaderModule m_fragShaderModule{nullptr};
00062
00063     }; // class Shaders
00064
00065 } // namespace ven
```

7.27 /home/runner/work/VEngine/VEngine/include/VEngine/Swap↩ Chain.hpp File Reference

This file contains the Shader class.

```
#include <vulkan/vulkan.h>
#include <memory>
#include "VEngine/Device.hpp"
```

Include dependency graph for SwapChain.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class ven::SwapChain

Namespaces

- namespace ven

7.27.1 Detailed Description

This file contains the Shader class.

Definition in file SwapChain.hpp.

7.28 SwapChain.hpp

Go to the documentation of this file.

```
00001 ///
00002 /// @file SwapChain.hpp
00003 /// @brief This file contains the Shader class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <vulkan/vulkan.h>
00010 #include <memory>
00011
00012 #include "VEngine/Device.hpp"
00013
00014 namespace ven {
00015
00016     class SwapChain {
00017
00018         public:
00019
00020             static constexpr int MAX_FRAMES_IN_FLIGHT = 2;
00021
00022             SwapChain(Device &deviceRef, const VkExtent2D windowExtentRef) : device{deviceRef},
      windowExtent{windowExtentRef} { init(); }
00023             SwapChain(Device &deviceRef, const VkExtent2D windowExtentRef, std::shared_ptr<SwapChain>
      previous) : device{deviceRef}, windowExtent{windowExtentRef}, oldSwapChain{std::move(previous)} {
      init(); oldSwapChain = nullptr; }
00024             ~SwapChain();
00025
00026             SwapChain(const SwapChain &) = delete;
00027             SwapChain& operator=(const SwapChain &) = delete;
00028
00029             [[nodiscard]] VkFramebuffer getFrameBuffer(const unsigned long index) const { return
      swapChainFramebuffers[index]; }
00030             [[nodiscard]] VkRenderPass getRenderPass() const { return renderPass; }
00031             [[nodiscard]] VkImageView getImageView(const int index) const { return
      swapChainImageViews[static_cast<unsigned long>(index)]; }
00032             [[nodiscard]] size_t imageCount() const { return swapChainImages.size(); }
00033             [[nodiscard]] VkFormat getSwapChainImageFormat() const { return swapChainImageFormat; }
00034             [[nodiscard]] VkExtent2D getSwapChainExtent() const { return m_swapChainExtent; }
00035             [[nodiscard]] uint32_t width() const { return m_swapChainExtent.width; }
00036             [[nodiscard]] uint32_t height() const { return m_swapChainExtent.height; }
00037
00038             [[nodiscard]] float extentAspectRatio() const { return
      static_cast<float>(m_swapChainExtent.width) / static_cast<float>(m_swapChainExtent.height); }
```

```
00039             VkFormat findDepthFormat() const;
00040
00041             VkResult acquireNextImage(uint32_t *imageIndex) const;
00042             VkResult submitCommandBuffers(const VkCommandBuffer *buffers, const uint32_t *imageIndex);
00043
00044             [[nodiscard]] bool compareSwapFormats(const SwapChain &swapChainp) const {
00045                 return swapChainImageFormat == swapChainp.swapChainImageFormat && swapChainDepthFormat
      == swapChainp.swapChainDepthFormat;
00046             }
00047
00048         private:
00049
00050             void init();
00051             void createSwapChain();
00052             void createImageViews();
00053             void createDepthResources();
00054             void createRenderPass();
00055             void createFramebuffers();
00056             void createSyncObjects();
00057
00058             static VkSurfaceFormatKHR chooseSwapSurfaceFormat(const std::vector<VkSurfaceFormatKHR>
      &availableFormats);
00059             static VkPresentModeKHR chooseSwapPresentMode(const std::vector<VkPresentModeKHR>
      &availablePresentModes);
00060             VkExtent2D chooseSwapExtent(const VkSurfaceCapabilitiesKHR &capabilities) const;
00061
00062             VkFormat swapChainImageFormat{};
00063             VkFormat swapChainDepthFormat{};
00064             VkExtent2D m_swapChainExtent{};
00065
00066             std::vector<VkFramebuffer> swapChainFramebuffers;
00067             VkRenderPass renderPass{};
00068
00069             std::vector<VkImage> depthImages;
00070             std::vector<VkDeviceMemory> depthImageMemorys;
00071             std::vector<VkImageView> depthImageViews;
00072             std::vector<VkImage> swapChainImages;
00073             std::vector<VkImageView> swapChainImageViews;
00074
00075             Device &device;
00076             VkExtent2D windowExtent;
00077
00078             VkSwapchainKHR swapChain{};
00079             std::shared_ptr<SwapChain> oldSwapChain;
00080
00081             std::vector<VkSemaphore> imageAvailableSemaphores;
00082             std::vector<VkSemaphore> renderFinishedSemaphores;
00083             std::vector<VkFence> inFlightFences;
00084             std::vector<VkFence> imagesInFlight;
00085             size_t currentFrame = 0;
00086
00087     }; // class SwapChain
00088
00089 }  // namespace ven
```

7.29 /home/runner/work/VEngine/VEngine/include/VEngine/System/↩ PointLightSystem.hpp File Reference

This file contains the PointLightSystem class.

```
#include <memory>
#include "VEngine/Device.hpp"
#include "VEngine/Shaders.hpp"
#include "VEngine/FrameInfo.hpp"
```

Include dependency graph for PointLightSystem.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class ven::PointLightSystem

  Class for point light system.

Namespaces

- namespace ven

7.29.1 Detailed Description

This file contains the PointLightSystem class.

Definition in file PointLightSystem.hpp.

7.30 PointLightSystem.hpp

Go to the documentation of this file.

```
00001 ///
00002 /// @file PointLightSystem.hpp
00003 /// @brief This file contains the PointLightSystem class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <memory>
00010
00011 #include "VEngine/Device.hpp"
00012 #include "VEngine/Shaders.hpp"
00013 #include "VEngine/FrameInfo.hpp"
00014
00015 namespace ven {
00016
00017     ///
00018     /// @class PointLightSystem
00019     /// @brief Class for point light system
00020     /// @namespace ven
00021     ///
00022     class PointLightSystem {
00023
00024         public:
00025
00026             explicit PointLightSystem(Device& device, VkRenderPass renderPass, VkDescriptorSetLayout
      globalSetLayout);
00027             ~PointLightSystem() { vkDestroyPipelineLayout(m_device.device(), m_pipelineLayout,
      nullptr); }
00028
00029             PointLightSystem(const PointLightSystem&) = delete;
00030             PointLightSystem& operator=(const PointLightSystem&) = delete;
00031
00032             static void update(const FrameInfo &frameInfo, GlobalUbo &ubo);
00033             void render(const FrameInfo &frameInfo) const;
00034
00035         private:
00036
00037             void createPipelineLayout(VkDescriptorSetLayout globalSetLayout);
00038             void createPipeline(VkRenderPass renderPass);
00039
00040             Device &m_device;
00041
00042             std::unique_ptr<Shaders> m_shaders;
00043             VkPipelineLayout m_pipelineLayout{nullptr};
00044
00045     }; // class PointLightSystem
00046
00047 } // namespace ven
```

7.31 /home/runner/work/VEngine/VEngine/include/VEngine/System/↩ RenderSystem.hpp File Reference

This file contains the RenderSystem class.

```
#include <memory>
#include <vulkan/vulkan.h>
#include "VEngine/Device.hpp"
#include "VEngine/Shaders.hpp"
#include "VEngine/FrameInfo.hpp"
```

Include dependency graph for RenderSystem.hpp: This graph shows which files directly or indirectly include this file:

Classes

- struct ven::SimplePushConstantData
- class ven::RenderSystem

  Class for render system.

Namespaces

- namespace ven

7.31.1 Detailed Description

This file contains the RenderSystem class.

Definition in file RenderSystem.hpp.

7.32 RenderSystem.hpp

Go to the documentation of this file.

```
00001 ///
00002 /// @file RenderSystem.hpp
00003 /// @brief This file contains the RenderSystem class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <memory>
00010
00011 #include <vulkan/vulkan.h>
00012
00013 #include "VEngine/Device.hpp"
00014 #include "VEngine/Shaders.hpp"
00015 #include "VEngine/FrameInfo.hpp"
00016
00017 namespace ven {
00018
00019     struct SimplePushConstantData {
00020         glm::mat4 modelMatrix{1.F};
00021         glm::mat4 normalMatrix{1.F};
00022     };
00023
00024     ///
00025     /// @class RenderSystem
00026     /// @brief Class for render system
00027     /// @namespace ven
00028     class RenderSystem {
00029
00030     public:
00031
00032         explicit RenderSystem(Device& device, VkRenderPass renderPass, VkDescriptorSetLayout
    globalSetLayout);
00033         ~RenderSystem() { vkDestroyPipelineLayout(m_device.device(), m_pipelineLayout, nullptr); }
00034
00035         RenderSystem(const RenderSystem&) = delete;
```

```
00036         RenderSystem& operator=(const RenderSystem&) = delete;
00037
00038         void renderObjects(const FrameInfo &frameInfo) const;
00039
00040     private:
00041
00042         void createPipelineLayout(VkDescriptorSetLayout globalSetLayout);
00043         void createPipeline(VkRenderPass renderPass);
00044
00045         Device &m_device;
00046
00047         std::unique_ptr<Shaders> m_shaders;
00048         VkPipelineLayout m_pipelineLayout{nullptr};
00049
00050     }; // class RenderSystem
00051
00052 } // namespace ven
```

7.33 /home/runner/work/VEngine/VEngine/include/VEngine/Utils.hpp File Reference

`#include <functional>`
Include dependency graph for Utils.hpp: This graph shows which files directly or indirectly include this file:

Namespaces

- namespace ven

Functions

- template<typename T , typename... Rest>
  void ven::hashCombine (std::size_t &seed, const T &v, const Rest &... rest)

7.34 Utils.hpp

Go to the documentation of this file.

```
00001 ///
00002 /// @file Utils.hpp
00003 /// @brief
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <functional>
00010
00011 namespace ven {
00012
00013     template<typename T, typename... Rest>
00014     void hashCombine(std::size_t& seed, const T& v, const Rest&... rest) {
00015         seed ^= std::hash<T>{}(v) + 0x9e3779b9 + (seed « 6) + (seed » 2);
00016         (hashCombine(seed, rest), ...);
00017     }
00018
00019 } // namespace ven
```

7.35 /home/runner/work/VEngine/VEngine/include/VEngine/Window.hpp File Reference

This file contains the Window class.

```
#include <string>
#include <GLFW/glfw3.h>
#include <vulkan/vulkan.h>
```

Include dependency graph for Window.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class ven::Window

Namespaces

- namespace ven

Macros

- #define GLFW_INCLUDE_VULKAN

7.35.1 Detailed Description

This file contains the Window class.

Definition in file Window.hpp.

7.35.2 Macro Definition Documentation

7.35.2.1 GLFW_INCLUDE_VULKAN

```
#define GLFW_INCLUDE_VULKAN
```

Definition at line 11 of file Window.hpp.

7.36 Window.hpp

Go to the documentation of this file.

```
00001 ///
00002 /// @file Window.hpp
00003 /// @brief This file contains the Window class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <string>
00010
00011 #define GLFW_INCLUDE_VULKAN
00012 #include <GLFW/glfw3.h>
00013 #include <vulkan/vulkan.h>
00014
00015 namespace ven {
00016
00017     class Window {
00018
00019         public:
00020
00021             Window(const uint32_t width, const uint32_t height, const std::string &title) :
      m_window(createWindow(width, height, title)), m_width(width), m_height(height) {};
00022             ~Window() { glfwDestroyWindow(m_window); glfwTerminate(); m_window = nullptr;};
00023
00024             [[nodiscard]] GLFWwindow* createWindow(uint32_t width, uint32_t height, const std::string
      &title);
00025             void createWindowSurface(VkInstance instance, VkSurfaceKHR* surface) const;
00026
00027             [[nodiscard]] GLFWwindow* getGLFWindow() const { return m_window; };
00028
00029             [[nodiscard]] VkExtent2D getExtent() const { return {m_width, m_height}; };
00030             [[nodiscard]] bool wasWindowResized() const { return m_framebufferResized; }
00031             void resetWindowResizedFlag() { m_framebufferResized = false; }
00032
00033         private:
00034
00035             static void framebufferResizeCallback(GLFWwindow* window, int width, int height);
00036
00037             GLFWwindow* m_window{nullptr};
00038             uint32_t m_width;
00039             uint32_t m_height;
00040
00041             bool m_framebufferResized = false;
00042
00043     }; // class Window
00044
00045 } // namespace ven
```

7.37 /home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/Clock/Clock.hpp File Reference

Clock class for time management.

```
#include <chrono>
#include "myLib/Clock/Time.hpp"
```

Include dependency graph for Clock.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class myLib::Clock

  Class for time management.

Namespaces

- namespace myLib

Typedefs

- using TimePoint = std::chrono::time_point<std::chrono::high_resolution_clock>

  TimePoint is a type alias for a time point which is a very long and complicated type in the standard library.

7.37.1 Detailed Description

Clock class for time management.

Definition in file Clock.hpp.

7.37.2 Typedef Documentation

7.37.2.1 TimePoint

```
using TimePoint = std::chrono::time_point<std::chrono::high_resolution_clock>
```

TimePoint is a type alias for a time point which is a very long and complicated type in the standard library.

Definition at line 16 of file Clock.hpp.

7.38 Clock.hpp

Go to the documentation of this file.

```
00001 ///
00002 /// @file Clock.hpp
00003 /// @brief Clock class for time management
00004 /// @namespace myLib
00005 ///
00006
00007 #pragma once
00008
00009 #include <chrono>
00010
00011 #include "myLib/Clock/Time.hpp"
00012
00013 ///
00014 /// @brief TimePoint is a type alias for a time point which is a very long and complicated type in the
      standard library
00015 ///
00016 using TimePoint = std::chrono::time_point<std::chrono::high_resolution_clock>;
00017
00018 namespace myLib {
00019
00020     ///
00021     /// @brief Class for time management
00022     ///
00023     class Clock {
00024
00025         public:
00026
00027             Clock() : m_start(std::chrono::high_resolution_clock::now()) {};
00028
00029             ~Clock() = default;
00030
00031             ///
00032             /// @brief Restart the clock
00033             ///
00034             void restart() { m_start = std::chrono::high_resolution_clock::now(); };
00035
00036             ///
00037             /// @brief Pause the clock
00038             ///
00039             void pause();
00040
```

```
00041             ///
00042             /// @brief Resume the clock
00043             ///
00044             void resume();
00045
00046             ///
00047             /// @brief Get the elapsed time since the last restart
00048             /// @return Time The elapsed time
00049             ///
00050             [[nodiscard]] Time getElapsedTime() const;
00051
00052         private:
00053
00054             ///
00055             /// @property The start time
00056             ///
00057             TimePoint m_start;
00058
00059             ///
00060             /// @property The pause time
00061             ///
00062             TimePoint m_pause;
00063
00064             ///
00065             /// @property The "is in pause" boolean variable
00066             ///
00067             bool m_paused{false};
00068
00069     }; // Clock
00070
00071 } // namespace myLib
```

7.39 /home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/Clock/Time.hpp File Reference

Class for time management.

This graph shows which files directly or indirectly include this file:

Classes

- class myLib::Time

  Class used for time management.

Namespaces

- namespace myLib

7.39.1 Detailed Description

Class for time management.

Definition in file Time.hpp.

7.40 Time.hpp

Go to the documentation of this file.

```
00001 ///
00002 /// @file Time.hpp
00003 /// @brief Class for time management
00004 /// @namespace myLib
00005 ///
00006
00007 #pragma once
00008
00009 namespace myLib {
00010
00011     ///
00012     /// @class Time
00013     /// @brief Class used for time management
00014     ///
00015     class Time {
00016
00017         public:
00018
00019             ///
00020             /// @brief Construct a new Time object
00021             ///
00022             explicit Time(const double seconds) : m_seconds(seconds) {};
00023
00024             ///
00025             /// @brief Transform the time to seconds
00026             /// @return int The time in seconds
00027             ///
00028             [[nodiscard]] int asSeconds() const { return static_cast<int>(m_seconds); };
00029
00030             ///
00031             /// @brief Transform the time to milliseconds
00032             /// @return int The time in milliseconds
00033             ///
00034             [[nodiscard]] int asMilliseconds() const { return static_cast<int>(m_seconds * 1000); }
00035
00036             ///
00037             /// @brief Transform the time to microseconds
00038             /// @return int The time in microseconds
00039             ///
00040             [[nodiscard]] int asMicroseconds() const { return static_cast<int>(m_seconds * 1000000);
    };
00041
00042         private:
00043
00044             ///
00045             /// @property The time in seconds
00046             ///
00047             double m_seconds{0.0F};
00048
00049     }; // Time
00050
00051 } // namespace myLib
```

7.41 /home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/Random.hpp File Reference

Class for random number generation.

`#include <random>`

Include dependency graph for Random.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class myLib::Random

  Class for random number generation.

Namespaces

- namespace myLib

7.41.1 Detailed Description

Class for random number generation.

Definition in file Random.hpp.

7.42 Random.hpp

Go to the documentation of this file.

```
00001 ///
00002 /// @file Random.hpp
00003 /// @brief Class for random number generation
00004 /// @namespace myLib
00005 ///
00006
00007 #pragma once
00008
00009 #include <random>
00010
00011 namespace myLib {
00012
00013     ///
00014     /// @class Random
00015     /// @brief Class for random number generation
00016     ///
00017     class Random {
00018
00019         public:
00020
00021             ///
00022             /// @brief Generate a random integer between min and max
00023             /// @param min The minimum value
00024             /// @param max The maximum value
00025             /// @return int The random integer
00026             ///
00027             static int randomInt(int min, int max);
00028             static int randomInt() { return randomInt(-1000, 1000); };
00029
00030             ///
00031             /// @param min The minimum value
00032             /// @param max The maximum value
00033             /// @return float The random float
00034             ///
00035             static float randomFloat(float min, float max);
00036             static float randomFloat() { return randomFloat(-1.0f, 1.0f); };
00037
00038     }; // class Random
00039
00040 } // namespace myLib
```

7.43 /home/runner/work/VEngine/VEngine/lib/local/static/my↩ Lib/src/clock.cpp File Reference

```
#include "myLib/Clock/Clock.hpp"
```

Include dependency graph for clock.cpp:

7.44 clock.cpp

Go to the documentation of this file.

```
00001 #include "myLib/Clock/Clock.hpp"
00002
00003 void myLib::Clock::pause()
00004 {
00005     if (m_paused) {
00006         return;
00007     }
00008     m_pause = std::chrono::high_resolution_clock::now();
00009     m_paused = true;
00010 }
00011
00012 void myLib::Clock::resume()
00013 {
00014     if (!m_paused) {
00015         return;
00016     }
00017
00018     m_start += std::chrono::high_resolution_clock::now() - m_pause;
00019     m_paused = false;
00020 }
00021
00022 myLib::Time myLib::Clock::getElapsedTime() const
00023 {
00024     TimePoint now = std::chrono::high_resolution_clock::now();
00025     std::chrono::duration<float> elapsed_time{};
00026     if (m_paused) {
00027         elapsed_time = m_pause - m_start;
00028     } else {
00029         elapsed_time = now - m_start;
00030     }
00031     return Time(elapsed_time.count());
00032 }
```

7.45 /home/runner/work/VEngine/VEngine/lib/local/static/myLib/src/random.cpp File Reference

```
#include "myLib/Random.hpp"
```

Include dependency graph for random.cpp:

7.46 random.cpp

Go to the documentation of this file.

```
00001 #include "myLib/Random.hpp"
00002
00003 int myLib::Random::randomInt(const int min, const int max)
00004 {
00005     std::mt19937 gen(std::random_device{}());
00006     std::uniform_int_distribution<> dis(min, max);
00007     return dis(gen);
00008 }
00009
00010 float myLib::Random::randomFloat(const float min, const float max)
00011 {
00012     return min + static_cast<float>(randomInt(-1000, 1000)) / 1000.0f * (max - min);
00013 }
```

7.47 /home/runner/work/VEngine/VEngine/README.md File Reference

7.48 /home/runner/work/VEngine/VEngine/src/buffer.cpp File Reference

```
#include <cassert>
#include <cstring>
#include "VEngine/Buffer.hpp"
```

Include dependency graph for buffer.cpp:

7.49 buffer.cpp

Go to the documentation of this file.

```
00001 #include <cassert>
00002 #include <cstring>
00003
00004 #include "VEngine/Buffer.hpp"
00005
00006 VkDeviceSize ven::Buffer::getAlignment(const VkDeviceSize instanceSize, const VkDeviceSize
      minOffsetAlignment) {
00007     if (minOffsetAlignment > 0) {
00008         return (instanceSize + minOffsetAlignment - 1) & ~(minOffsetAlignment - 1);
00009     }
00010     return instanceSize;
00011 }
00012
00013 ven::Buffer::Buffer(Device &device, const VkDeviceSize instanceSize, const uint32_t instanceCount,
      const VkBufferUsageFlags usageFlags, const VkMemoryPropertyFlags memoryPropertyFlags, const
      VkDeviceSize minOffsetAlignment) : m_device{device}, m_instanceSize{instanceSize},
      m_instanceCount{instanceCount}, m_alignmentSize(getAlignment(instanceSize, minOffsetAlignment)),
      m_usageFlags{usageFlags}, m_memoryPropertyFlags{memoryPropertyFlags}
00014 {
00015     m_bufferSize = m_alignmentSize * m_instanceCount;
00016     device.createBuffer(m_bufferSize, m_usageFlags, m_memoryPropertyFlags, m_buffer, m_memory);
00017 }
00018
00019 ven::Buffer::~Buffer()
00020 {
00021     unmap();
00022     vkDestroyBuffer(m_device.device(), m_buffer, nullptr);
00023     vkFreeMemory(m_device.device(), m_memory, nullptr);
00024 }
00025
00026 VkResult ven::Buffer::map(const VkDeviceSize size, const VkDeviceSize offset)
00027 {
00028     assert(m_buffer && m_memory && "Called map on m_buffer before create");
00029     return vkMapMemory(m_device.device(), m_memory, offset, size, 0, &m_mapped);
00030 }
00031
00032 void ven::Buffer::unmap()
00033 {
00034     if (m_mapped != nullptr) {
00035         vkUnmapMemory(m_device.device(), m_memory);
00036         m_mapped = nullptr;
00037     }
00038 }
00039
00040 void ven::Buffer::writeToBuffer(const void *data, const VkDeviceSize size, const VkDeviceSize offset)
      const
00041 {
00042     assert(m_mapped && "Cannot copy to unmapped m_buffer");
00043
00044     if (size == VK_WHOLE_SIZE) {
00045         memcpy(m_mapped, data, m_bufferSize);
00046     } else {
00047         char *memOffset = static_cast<char *>(m_mapped);
00048         memOffset += offset;
00049         memcpy(memOffset, data, size);
00050     }
00051 }
00052
00053 VkResult ven::Buffer::flush(const VkDeviceSize size, const VkDeviceSize offset) const
00054 {
00055     VkMappedMemoryRange mappedRange = {};
00056     mappedRange.sType = VK_STRUCTURE_TYPE_MAPPED_MEMORY_RANGE;
00057     mappedRange.memory = m_memory;
00058     mappedRange.offset = offset;
00059     mappedRange.size = size;
00060     return vkFlushMappedMemoryRanges(m_device.device(), 1, &mappedRange);
00061 }
00062
00063 VkResult ven::Buffer::invalidate(const VkDeviceSize size, const VkDeviceSize offset) const
00064 {
00065     VkMappedMemoryRange mappedRange = {};
00066     mappedRange.sType = VK_STRUCTURE_TYPE_MAPPED_MEMORY_RANGE;
00067     mappedRange.memory = m_memory;
00068     mappedRange.offset = offset;
00069     mappedRange.size = size;
00070     return vkInvalidateMappedMemoryRanges(m_device.device(), 1, &mappedRange);
00071 }
```

7.50 /home/runner/work/VEngine/VEngine/src/camera.cpp File Reference

```
#include <cassert>
#include <cmath>
#include "VEngine/Camera.hpp"
```

Include dependency graph for camera.cpp:

7.51 camera.cpp

Go to the documentation of this file.

```
00001 #include <cassert>
00002 #include <cmath>
00003
00004 #include "VEngine/Camera.hpp"
00005
00006 void ven::Camera::setOrthographicProjection(const float left, const float right, const float top,
      const float bottom, const float near, const float far)
00007 {
00008     m_projectionMatrix = glm::mat4{1.0F};
00009     m_projectionMatrix[0][0] = 2.F / (right - left);
00010     m_projectionMatrix[1][1] = 2.F / (bottom - top);
00011     m_projectionMatrix[2][2] = 1.F / (far - near);
00012     m_projectionMatrix[3][0] = -(right + left) / (right - left);
00013     m_projectionMatrix[3][1] = -(bottom + top) / (bottom - top);
00014     m_projectionMatrix[3][2] = -near / (far - near);
00015 }
00016
00017 void ven::Camera::setPerspectiveProjection(const float fovy, const float aspect, const float near,
      const float far)
00018 {
00019     assert(glm::abs(aspect - std::numeric_limits<float>::epsilon()) > 0.0F);
00020     const float tanHalfFovy = std::tan(fovy / 2.F);
00021     m_projectionMatrix = glm::mat4{0.0F};
00022     m_projectionMatrix[0][0] = 1.F / (aspect * tanHalfFovy);
00023     m_projectionMatrix[1][1] = 1.F / (tanHalfFovy);
00024     m_projectionMatrix[2][2] = far / (far - near);
00025     m_projectionMatrix[2][3] = 1.F;
00026     m_projectionMatrix[3][2] = -(far * near) / (far - near);
00027 }
00028
00029 void ven::Camera::setViewDirection(const glm::vec3 position, const glm::vec3 direction, const
      glm::vec3 up)
00030 {
00031     const glm::vec3 w{normalize(direction)};
00032     const glm::vec3 u{normalize(cross(w, up))};
00033     const glm::vec3 v{cross(w, u)};
00034
00035     m_viewMatrix = glm::mat4{1.F};
00036     m_viewMatrix[0][0] = u.x;
00037     m_viewMatrix[1][0] = u.y;
00038     m_viewMatrix[2][0] = u.z;
00039     m_viewMatrix[0][1] = v.x;
00040     m_viewMatrix[1][1] = v.y;
00041     m_viewMatrix[2][1] = v.z;
00042     m_viewMatrix[0][2] = w.x;
00043     m_viewMatrix[1][2] = w.y;
00044     m_viewMatrix[2][2] = w.z;
00045     m_viewMatrix[3][0] = -dot(u, position);
00046     m_viewMatrix[3][1] = -dot(v, position);
00047     m_viewMatrix[3][2] = -dot(w, position);
00048
00049     m_inverseViewMatrix = glm::mat4{1.F};
00050     m_inverseViewMatrix[0][0] = u.x;
00051     m_inverseViewMatrix[0][1] = u.y;
00052     m_inverseViewMatrix[0][2] = u.z;
00053     m_inverseViewMatrix[1][0] = v.x;
00054     m_inverseViewMatrix[1][1] = v.y;
00055     m_inverseViewMatrix[1][2] = v.z;
00056     m_inverseViewMatrix[2][0] = w.x;
00057     m_inverseViewMatrix[2][1] = w.y;
00058     m_inverseViewMatrix[2][2] = w.z;
00059     m_inverseViewMatrix[3][0] = position.x;
00060     m_inverseViewMatrix[3][1] = position.y;
00061     m_inverseViewMatrix[3][2] = position.z;
00062 }
```

```
00063
00064 void ven::Camera::setViewYXZ(const glm::vec3 position, const glm::vec3 rotation)
00065 {
00066     const float c3 = glm::cos(rotation.z);
00067     const float s3 = glm::sin(rotation.z);
00068     const float c2 = glm::cos(rotation.x);
00069     const float s2 = glm::sin(rotation.x);
00070     const float c1 = glm::cos(rotation.y);
00071     const float s1 = glm::sin(rotation.y);
00072     const glm::vec3 u{(c1 * c3 + s1 * s2 * s3), (c2 * s3), (c1 * s2 * s3 - c3 * s1)};
00073     const glm::vec3 v{(c3 * s1 * s2 - c1 * s3), (c2 * c3), (c1 * c3 * s2 + s1 * s3)};
00074     const glm::vec3 w{(c2 * s1), (-s2), (c1 * c2)};
00075     m_viewMatrix = glm::mat4{1.F};
00076     m_viewMatrix[0][0] = u.x;
00077     m_viewMatrix[1][0] = u.y;
00078     m_viewMatrix[2][0] = u.z;
00079     m_viewMatrix[0][1] = v.x;
00080     m_viewMatrix[1][1] = v.y;
00081     m_viewMatrix[2][1] = v.z;
00082     m_viewMatrix[0][2] = w.x;
00083     m_viewMatrix[1][2] = w.y;
00084     m_viewMatrix[2][2] = w.z;
00085     m_viewMatrix[3][0] = -dot(u, position);
00086     m_viewMatrix[3][1] = -dot(v, position);
00087     m_viewMatrix[3][2] = -dot(w, position);
00088
00089     m_inverseViewMatrix = glm::mat4{1.F};
00090     m_inverseViewMatrix[0][0] = u.x;
00091     m_inverseViewMatrix[0][1] = u.y;
00092     m_inverseViewMatrix[0][2] = u.z;
00093     m_inverseViewMatrix[1][0] = v.x;
00094     m_inverseViewMatrix[1][1] = v.y;
00095     m_inverseViewMatrix[1][2] = v.z;
00096     m_inverseViewMatrix[2][0] = w.x;
00097     m_inverseViewMatrix[2][1] = w.y;
00098     m_inverseViewMatrix[2][2] = w.z;
00099     m_inverseViewMatrix[3][0] = position.x;
00100     m_inverseViewMatrix[3][1] = position.y;
00101     m_inverseViewMatrix[3][2] = position.z;
00102 }
```

7.52 /home/runner/work/VEngine/VEngine/src/descriptors.cpp File Reference

```
#include <cassert>
#include "VEngine/Descriptors.hpp"
```

Include dependency graph for descriptors.cpp:

7.53 descriptors.cpp

Go to the documentation of this file.

```
00001 #include <cassert>
00002
00003 #include "VEngine/Descriptors.hpp"
00004
00005 ven::DescriptorSetLayout::Builder &ven::DescriptorSetLayout::Builder::addBinding(const uint32_t
      binding, const VkDescriptorType descriptorType, const VkShaderStageFlags stageFlags, const uint32_t
      count)
00006 {
00007     assert(m_bindings.count(binding) == 0 && "Binding already in use");
00008     VkDescriptorSetLayoutBinding layoutBinding{};
00009     layoutBinding.binding = binding;
00010     layoutBinding.descriptorType = descriptorType;
00011     layoutBinding.descriptorCount = count;
00012     layoutBinding.stageFlags = stageFlags;
00013     m_bindings[binding] = layoutBinding;
00014     return *this;
00015 }
00016
00017 ven::DescriptorSetLayout::DescriptorSetLayout(Device &device, const std::unordered_map<uint32_t,
      VkDescriptorSetLayoutBinding>& bindings) : m_device{device}, m_bindings{bindings}
00018 {
00019     std::vector<VkDescriptorSetLayoutBinding> setLayoutBindings{};
```

```
00020     setLayoutBindings.reserve(bindings.size());
00021 for (auto kv : bindings) {
00022         setLayoutBindings.push_back(kv.second);
00023     }
00024 
00025     VkDescriptorSetLayoutCreateInfo descriptorSetLayoutInfo{};
00026     descriptorSetLayoutInfo.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_CREATE_INFO;
00027     descriptorSetLayoutInfo.bindingCount = static_cast<uint32_t>(setLayoutBindings.size());
00028     descriptorSetLayoutInfo.pBindings = setLayoutBindings.data();
00029 
00030     if (vkCreateDescriptorSetLayout(
00031             m_device.device(),
00032             &descriptorSetLayoutInfo,
00033             nullptr,
00034             &m_descriptorSetLayout) != VK_SUCCESS) {
00035         throw std::runtime_error("failed to create descriptor set layout!");
00036     }
00037 }
00038 
00039 ven::DescriptorPool::Builder &ven::DescriptorPool::Builder::addPoolSize(const VkDescriptorType
    descriptorType, const uint32_t count)
00040 {
00041     m_poolSizes.push_back({descriptorType, count});
00042     return *this;
00043 }
00044 
00045 ven::DescriptorPool::Builder &ven::DescriptorPool::Builder::setPoolFlags(const
    VkDescriptorPoolCreateFlags flags)
00046 {
00047     m_poolFlags = flags;
00048     return *this;
00049 }
00050 ven::DescriptorPool::Builder &ven::DescriptorPool::Builder::setMaxSets(const uint32_t count)
00051 {
00052     m_maxSets = count;
00053     return *this;
00054 }
00055 
00056 ven::DescriptorPool::DescriptorPool(Device &device, const uint32_t maxSets, const
    VkDescriptorPoolCreateFlags poolFlags, const std::vector<VkDescriptorPoolSize> &poolSizes) :
    m_device{device}
00057 {
00058     VkDescriptorPoolCreateInfo descriptorPoolInfo{};
00059     descriptorPoolInfo.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_POOL_CREATE_INFO;
00060     descriptorPoolInfo.poolSizeCount = static_cast<uint32_t>(poolSizes.size());
00061     descriptorPoolInfo.pPoolSizes = poolSizes.data();
00062     descriptorPoolInfo.maxSets = maxSets;
00063     descriptorPoolInfo.flags = poolFlags;
00064 
00065     if (vkCreateDescriptorPool(m_device.device(), &descriptorPoolInfo, nullptr, &m_descriptorPool) !=
00066         VK_SUCCESS) {
00067         throw std::runtime_error("failed to create descriptor pool!");
00068     }
00069 }
00070 
00071 bool ven::DescriptorPool::allocateDescriptor(const VkDescriptorSetLayout descriptorSetLayout,
    VkDescriptorSet &descriptor) const
00072 {
00073     VkDescriptorSetAllocateInfo allocInfo{};
00074     allocInfo.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_ALLOCATE_INFO;
00075     allocInfo.descriptorPool = m_descriptorPool;
00076     allocInfo.pSetLayouts = &descriptorSetLayout;
00077     allocInfo.descriptorSetCount = 1;
00078 
00079     // Might want to create a "DescriptorPoolManager" class that handles this case, and builds
00080     // a new pool whenever an old pool fills up. But this is beyond our current scope
00081     return vkAllocateDescriptorSets(m_device.device(), &allocInfo, &descriptor) == VK_SUCCESS;
00082 }
00083 
00084 ven::DescriptorWriter &ven::DescriptorWriter::writeBuffer(const uint32_t binding, const
    VkDescriptorBufferInfo *bufferInfo)
00085 {
00086     assert(setLayout.bindings.count(binding) == 1 && "Layout does not contain specified binding");
00087 
00088     const auto &bindingDescription = m_setLayout.m_bindings[binding];
00089 
00090     assert(bindingDescription.descriptorCount == 1 && "Binding single descriptor info, but binding
    expects multiple");
00091 
00092     VkWriteDescriptorSet write{};
00093     write.sType = VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET;
00094     write.descriptorType = bindingDescription.descriptorType;
00095     write.dstBinding = binding;
00096     write.pBufferInfo = bufferInfo;
00097     write.descriptorCount = 1;
00098 
00099     m_writes.push_back(write);
```

```
00100     return *this;
00101 }
00102
00103 ven::DescriptorWriter &ven::DescriptorWriter::writeImage(const uint32_t binding, const
      VkDescriptorImageInfo *imageInfo)
00104 {
00105     assert(setLayout.bindings.count(binding) == 1 && "Layout does not contain specified binding");
00106
00107     const VkDescriptorSetLayoutBinding &bindingDescription = m_setLayout.m_bindings[binding];
00108
00109     assert(bindingDescription.descriptorCount == 1 && "Binding single descriptor info, but binding
      expects multiple");
00110
00111     VkWriteDescriptorSet write{};
00112     write.sType = VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET;
00113     write.descriptorType = bindingDescription.descriptorType;
00114     write.dstBinding = binding;
00115     write.pImageInfo = imageInfo;
00116     write.descriptorCount = 1;
00117
00118     m_writes.push_back(write);
00119     return *this;
00120 }
00121
00122 bool ven::DescriptorWriter::build(VkDescriptorSet &set)
00123 {
00124     if (!m_pool.allocateDescriptor(m_setLayout.getDescriptorSetLayout(), set)) {
00125         return false;
00126     }
00127     overwrite(set);
00128     return true;
00129 }
00130
00131 void ven::DescriptorWriter::overwrite(const VkDescriptorSet &set)
00132 {
00133     for (auto &write : m_writes) {
00134         write.dstSet = set;
00135     }
00136     vkUpdateDescriptorSets(m_pool.m_device.device(), static_cast<unsigned int>(m_writes.size()),
      m_writes.data(), 0, nullptr);
00137 }
```

7.54 /home/runner/work/VEngine/VEngine/src/device.cpp File Reference

```
#include <cstring>
#include <iostream>
#include <set>
#include <unordered_set>
#include "VEngine/Device.hpp"
```

Include dependency graph for device.cpp:

Functions

- static VKAPI_ATTR VkBool32 VKAPI_CALL debugCallback (const VkDebugUtilsMessageSeverityFlagBits↩ EXT messageSeverity, const VkDebugUtilsMessageTypeFlagsEXT messageType, const VkDebugUtils↩ MessengerCallbackDataEXT ∗pCallbackData, void ∗pUserData)
- VkResult CreateDebugUtilsMessengerEXT (const VkInstance instance, const VkDebugUtilsMessenger↩ CreateInfoEXT ∗pCreateInfo, const VkAllocationCallbacks ∗pAllocator, VkDebugUtilsMessengerEXT ∗p↩ DebugMessenger)
- void DestroyDebugUtilsMessengerEXT (const VkInstance instance, const VkDebugUtilsMessengerEXT debugMessenger, const VkAllocationCallbacks ∗pAllocator)

7.54.1 Function Documentation

7.54.1.1 CreateDebugUtilsMessengerEXT()

```
VkResult CreateDebugUtilsMessengerEXT (
            const VkInstance instance,
```

```
            const VkDebugUtilsMessengerCreateInfoEXT * pCreateInfo,
            const VkAllocationCallbacks * pAllocator,
            VkDebugUtilsMessengerEXT * pDebugMessenger)
```

Definition at line 16 of file device.cpp.

Referenced by ven::Device::setupDebugMessenger().

Here is the caller graph for this function:

7.54.1.2 debugCallback()

```
static VKAPI_ATTR VkBool32 VKAPI_CALL debugCallback (
            const VkDebugUtilsMessageSeverityFlagBitsEXT messageSeverity,
            const VkDebugUtilsMessageTypeFlagsEXT messageType,
            const VkDebugUtilsMessengerCallbackDataEXT * pCallbackData,
            void * pUserData)  [static]
```

Definition at line 8 of file device.cpp.

Referenced by ven::Device::populateDebugMessengerCreateInfo().

Here is the caller graph for this function:

7.54.1.3 DestroyDebugUtilsMessengerEXT()

```
void DestroyDebugUtilsMessengerEXT (
            const VkInstance instance,
            const VkDebugUtilsMessengerEXT debugMessenger,
            const VkAllocationCallbacks * pAllocator)
```

Definition at line 26 of file device.cpp.

Referenced by ven::Device::~Device().

Here is the caller graph for this function:

7.55 device.cpp

Go to the documentation of this file.

```
00001 #include <cstring>
00002 #include <iostream>
00003 #include <set>
00004 #include <unordered_set>
00005
00006 #include "VEngine/Device.hpp"
00007
00008 static VKAPI_ATTR VkBool32 VKAPI_CALL debugCallback(const VkDebugUtilsMessageSeverityFlagBitsEXT
      messageSeverity, const VkDebugUtilsMessageTypeFlagsEXT messageType, const
      VkDebugUtilsMessengerCallbackDataEXT *pCallbackData, void *pUserData)
00009 {
00010     (void) pUserData; (void) messageSeverity; (void) messageType;
00011
00012     std::cerr « "validation layer: " « pCallbackData->pMessage « '\n';
00013     return VK_FALSE;
00014 }
00015
```

```
00016 VkResult CreateDebugUtilsMessengerEXT(const VkInstance instance, const
      VkDebugUtilsMessengerCreateInfoEXT *pCreateInfo, const VkAllocationCallbacks *pAllocator,
      VkDebugUtilsMessengerEXT *pDebugMessenger)
00017 {
00018     auto func = reinterpret_cast<PFN_vkCreateDebugUtilsMessengerEXT>(vkGetInstanceProcAddr(instance,
      "vkCreateDebugUtilsMessengerEXT"));
00019     if (func != nullptr) {
00020         return func(instance, pCreateInfo, pAllocator, pDebugMessenger);
00021     }
00022
00023     return VK_ERROR_EXTENSION_NOT_PRESENT;
00024 }
00025
00026 void DestroyDebugUtilsMessengerEXT(const VkInstance instance, const VkDebugUtilsMessengerEXT
      debugMessenger, const VkAllocationCallbacks *pAllocator)
00027 {
00028     auto func = reinterpret_cast<PFN_vkDestroyDebugUtilsMessengerEXT>(vkGetInstanceProcAddr(instance,
      "vkDestroyDebugUtilsMessengerEXT"));
00029     if (func != nullptr) {
00030         func(instance, debugMessenger, pAllocator);
00031     }
00032 }
00033
00034 ven::Device::Device(Window &window) : m_window{window}
00035 {
00036     createInstance();
00037     setupDebugMessenger();
00038     createSurface();
00039     pickPhysicalDevice();
00040     createLogicalDevice();
00041     createCommandPool();
00042 }
00043
00044 ven::Device::~Device()
00045 {
00046     vkDestroyCommandPool(device_, commandPool, nullptr);
00047     vkDestroyDevice(device_, nullptr);
00048
00049     if (enableValidationLayers) {
00050         DestroyDebugUtilsMessengerEXT(instance, debugMessenger, nullptr);
00051     }
00052
00053     vkDestroySurfaceKHR(instance, surface_, nullptr);
00054     vkDestroyInstance(instance, nullptr);
00055 }
00056
00057 void ven::Device::createInstance()
00058 {
00059     if (enableValidationLayers && !checkValidationLayerSupport()) {
00060         throw std::runtime_error("validation layers requested, but not available!");
00061     }
00062
00063     VkApplicationInfo appInfo = {};
00064     appInfo.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
00065     appInfo.pApplicationName = "LittleVulkanEngine App";
00066     appInfo.applicationVersion = VK_MAKE_VERSION(1, 0, 0);
00067     appInfo.pEngineName = "No Engine";
00068     appInfo.engineVersion = VK_MAKE_VERSION(1, 0, 0);
00069     appInfo.apiVersion = VK_API_VERSION_1_0;
00070
00071     VkInstanceCreateInfo createInfo = {};
00072     createInfo.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
00073     createInfo.pApplicationInfo = &appInfo;
00074
00075     std::vector<const char *> extensions = getRequiredExtensions();
00076     createInfo.enabledExtensionCount = static_cast<uint32_t>(extensions.size());
00077     createInfo.ppEnabledExtensionNames = extensions.data();
00078
00079     VkDebugUtilsMessengerCreateInfoEXT debugCreateInfo;
00080     if (enableValidationLayers) {
00081         createInfo.enabledLayerCount = static_cast<uint32_t>(validationLayers.size());
00082         createInfo.ppEnabledLayerNames = validationLayers.data();
00083
00084         populateDebugMessengerCreateInfo(debugCreateInfo);
00085         createInfo.pNext = &debugCreateInfo;
00086     } else {
00087         createInfo.enabledLayerCount = 0;
00088         createInfo.pNext = nullptr;
00089     }
00090
00091     if (vkCreateInstance(&createInfo, nullptr, &instance) != VK_SUCCESS) {
00092         throw std::runtime_error("failed to create instance!");
00093     }
00094
00095     hasGlfwRequiredInstanceExtensions();
00096 }
00097
```

```
00098 void ven::Device::pickPhysicalDevice()
00099 {
00100     uint32_t deviceCount = 0;
00101     vkEnumeratePhysicalDevices(instance, &deviceCount, nullptr);
00102     if (deviceCount == 0) {
00103         throw std::runtime_error("failed to find GPUs with Vulkan support!");
00104     }
00105     std::cout « "Device count: " « deviceCount « '\n';
00106     std::vector<VkPhysicalDevice> devices(deviceCount);
00107     vkEnumeratePhysicalDevices(instance, &deviceCount, devices.data());
00108 
00109     for (const auto &device : devices) {
00110         if (isDeviceSuitable(device)) {
00111             physicalDevice = device;
00112             break;
00113         }
00114     }
00115 
00116     if (physicalDevice == VK_NULL_HANDLE) {
00117         throw std::runtime_error("failed to find a suitable GPU!");
00118     }
00119 
00120     vkGetPhysicalDeviceProperties(physicalDevice, &m_properties);
00121     std::cout « "physical device: " « m_properties.deviceName « '\n';
00122 }
00123 
00124 void ven::Device::createLogicalDevice()
00125 {
00126     const QueueFamilyIndices indices = findQueueFamilies(physicalDevice);
00127 
00128     std::vector<VkDeviceQueueCreateInfo> queueCreateInfos;
00129     const std::set<uint32_t> uniqueQueueFamilies = {indices.graphicsFamily, indices.presentFamily};
00130     float queuePriority = 1.0F;
00131 
00132     for (const uint32_t queueFamily : uniqueQueueFamilies) {
00133         VkDeviceQueueCreateInfo queueCreateInfo = {};
00134         queueCreateInfo.sType = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO;
00135         queueCreateInfo.queueFamilyIndex = queueFamily;
00136         queueCreateInfo.queueCount = 1;
00137         queueCreateInfo.pQueuePriorities = &queuePriority;
00138         queueCreateInfos.push_back(queueCreateInfo);
00139     }
00140 
00141     VkPhysicalDeviceFeatures deviceFeatures = {};
00142     deviceFeatures.samplerAnisotropy = VK_TRUE;
00143 
00144     VkDeviceCreateInfo createInfo = {};
00145     createInfo.sType = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
00146 
00147     createInfo.queueCreateInfoCount = static_cast<uint32_t>(queueCreateInfos.size());
00148     createInfo.pQueueCreateInfos = queueCreateInfos.data();
00149 
00150     createInfo.pEnabledFeatures = &deviceFeatures;
00151     createInfo.enabledExtensionCount = static_cast<uint32_t>(deviceExtensions.size());
00152     createInfo.ppEnabledExtensionNames = deviceExtensions.data();
00153 
00154         // might not really be necessary anymore because device specific validation layers
00155         // have been deprecated
00156     if (enableValidationLayers) {
00157         createInfo.enabledLayerCount = static_cast<uint32_t>(validationLayers.size());
00158         createInfo.ppEnabledLayerNames = validationLayers.data();
00159     } else {
00160         createInfo.enabledLayerCount = 0;
00161     }
00162 
00163     if (vkCreateDevice(physicalDevice, &createInfo, nullptr, &device_) != VK_SUCCESS) {
00164         throw std::runtime_error("failed to create logical device!");
00165     }
00166 
00167     vkGetDeviceQueue(device_, indices.graphicsFamily, 0, &graphicsQueue_);
00168     vkGetDeviceQueue(device_, indices.presentFamily, 0, &presentQueue_);
00169 }
00170 
00171 void ven::Device::createCommandPool()
00172 {
00173     const QueueFamilyIndices queueFamilyIndices = findPhysicalQueueFamilies();
00174 
00175     VkCommandPoolCreateInfo poolInfo = {};
00176     poolInfo.sType = VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO;
00177     poolInfo.queueFamilyIndex = queueFamilyIndices.graphicsFamily;
00178     poolInfo.flags = VK_COMMAND_POOL_CREATE_TRANSIENT_BIT |
    VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT;
00179 
00180     if (vkCreateCommandPool(device_, &poolInfo, nullptr, &commandPool) != VK_SUCCESS) {
00181         throw std::runtime_error("failed to create command pool!");
00182     }
00183 }
```

```
00184
00185 bool ven::Device::isDeviceSuitable(const VkPhysicalDevice device) const
00186 {
00187     const QueueFamilyIndices indices = findQueueFamilies(device);
00188     const bool extensionsSupported = checkDeviceExtensionSupport(device);
00189     bool swapChainAdequate = false;
00190
00191     if (extensionsSupported) {
00192         SwapChainSupportDetails swapChainSupport = querySwapChainSupport(device);
00193         swapChainAdequate = !swapChainSupport.formats.empty() &&
      !swapChainSupport.presentModes.empty();
00194     }
00195
00196     VkPhysicalDeviceFeatures supportedFeatures;
00197     vkGetPhysicalDeviceFeatures(device, &supportedFeatures);
00198
00199     return indices.isComplete() && extensionsSupported && swapChainAdequate &&
      (supportedFeatures.samplerAnisotropy != 0U);
00200 }
00201
00202 void ven::Device::populateDebugMessengerCreateInfo(VkDebugUtilsMessengerCreateInfoEXT &createInfo)
00203 {
00204     createInfo = {};
00205     createInfo.sType = VK_STRUCTURE_TYPE_DEBUG_UTILS_MESSENGER_CREATE_INFO_EXT;
00206     createInfo.messageSeverity = VK_DEBUG_UTILS_MESSAGE_SEVERITY_WARNING_BIT_EXT |
00207                                  VK_DEBUG_UTILS_MESSAGE_SEVERITY_ERROR_BIT_EXT;
00208     createInfo.messageType = VK_DEBUG_UTILS_MESSAGE_TYPE_GENERAL_BIT_EXT |
00209                              VK_DEBUG_UTILS_MESSAGE_TYPE_VALIDATION_BIT_EXT |
00210                              VK_DEBUG_UTILS_MESSAGE_TYPE_PERFORMANCE_BIT_EXT;
00211     createInfo.pfnUserCallback = debugCallback;
00212     createInfo.pUserData = nullptr;  // Optional
00213 }
00214
00215 void ven::Device::setupDebugMessenger()
00216 {
00217     if (!enableValidationLayers) { return; }
00218     VkDebugUtilsMessengerCreateInfoEXT createInfo;
00219     populateDebugMessengerCreateInfo(createInfo);
00220     if (CreateDebugUtilsMessengerEXT(instance, &createInfo, nullptr, &debugMessenger) != VK_SUCCESS) {
00221         throw std::runtime_error("failed to set up debug messenger!");
00222     }
00223 }
00224
00225 bool ven::Device::checkValidationLayerSupport() const
00226 {
00227     uint32_t layerCount = 0;
00228     vkEnumerateInstanceLayerProperties(&layerCount, nullptr);
00229
00230     std::vector<VkLayerProperties> availableLayers(layerCount);
00231     vkEnumerateInstanceLayerProperties(&layerCount, availableLayers.data());
00232
00233     for (const char *layerName : validationLayers) {
00234         bool layerFound = false;
00235
00236         for (const auto &layerProperties : availableLayers) {
00237             if (strcmp(layerName, layerProperties.layerName) == 0) {
00238                 layerFound = true;
00239                 break;
00240             }
00241         }
00242         if (!layerFound) {
00243             return false;
00244         }
00245     }
00246
00247     return true;
00248 }
00249
00250 std::vector<const char *> ven::Device::getRequiredExtensions() const
00251 {
00252     uint32_t glfwExtensionCount = 0;
00253     const char **glfwExtensions = nullptr;
00254     glfwExtensions = glfwGetRequiredInstanceExtensions(&glfwExtensionCount);
00255
00256     std::vector<const char *> extensions(glfwExtensions, glfwExtensions + glfwExtensionCount);
00257
00258     if (enableValidationLayers) {
00259         extensions.push_back(VK_EXT_DEBUG_UTILS_EXTENSION_NAME);
00260     }
00261
00262     return extensions;
00263 }
00264
00265 void ven::Device::hasGlfwRequiredInstanceExtensions() const
00266 {
00267     uint32_t extensionCount = 0;
00268     vkEnumerateInstanceExtensionProperties(nullptr, &extensionCount, nullptr);
```

```
00269     std::vector<VkExtensionProperties> extensions(extensionCount);
00270     vkEnumerateInstanceExtensionProperties(nullptr, &extensionCount, extensions.data());
00271 
00272     std::cout « "available extensions:\n";
00273     std::unordered_set<std::string> available;
00274     for (const auto &extension : extensions) {
00275         std::cout « '\t' « extension.extensionName « '\n';
00276         available.insert(extension.extensionName);
00277     }
00278 
00279     std::cout « "required extensions:\n";
00280     const std::vector<const char *> requiredExtensions = getRequiredExtensions();
00281     for (const auto &required : requiredExtensions) {
00282         std::cout « "\t" « required « '\n';
00283         if (available.find(required) == available.end()) {
00284             throw std::runtime_error("Missing required glfw extension");
00285         }
00286     }
00287 }
00288 
00289 bool ven::Device::checkDeviceExtensionSupport(const VkPhysicalDevice device) const
00290 {
00291     uint32_t extensionCount = 0;
00292     vkEnumerateDeviceExtensionProperties(device, nullptr, &extensionCount, nullptr);
00293 
00294     std::vector<VkExtensionProperties> availableExtensions(extensionCount);
00295     vkEnumerateDeviceExtensionProperties(device, nullptr, &extensionCount,
    availableExtensions.data());
00296 
00297     std::set<std::string> requiredExtensions(deviceExtensions.begin(), deviceExtensions.end());
00298      for (const auto &extension : availableExtensions) {
00299          requiredExtensions.erase(extension.extensionName);
00300      }
00301 
00302      return requiredExtensions.empty();
00303 }
00304 
00305 ven::QueueFamilyIndices ven::Device::findQueueFamilies(const VkPhysicalDevice device) const
00306 {
00307     QueueFamilyIndices indices;
00308 
00309     uint32_t queueFamilyCount = 0;
00310     vkGetPhysicalDeviceQueueFamilyProperties(device, &queueFamilyCount, nullptr);
00311     std::vector<VkQueueFamilyProperties> queueFamilies(queueFamilyCount);
00312     vkGetPhysicalDeviceQueueFamilyProperties(device, &queueFamilyCount, queueFamilies.data());
00313     uint32_t index = 0;
00314 
00315     for (const auto &queueFamily : queueFamilies) {
00316         if (queueFamily.queueCount > 0 && ((queueFamily.queueFlags & VK_QUEUE_GRAPHICS_BIT) != 0U)) {
00317             indices.graphicsFamily = index;
00318             indices.graphicsFamilyHasValue = true;
00319         }
00320         VkBool32 presentSupport = 0U;
00321         vkGetPhysicalDeviceSurfaceSupportKHR(device, index, surface_, &presentSupport);
00322         if (queueFamily.queueCount > 0 && (presentSupport != 0U)) {
00323             indices.presentFamily = index;
00324             indices.presentFamilyHasValue = true;
00325         }
00326         if (indices.isComplete()) {
00327             break;
00328         }
00329         index++;
00330     }
00331     return indices;
00332 }
00333 
00334 ven::SwapChainSupportDetails ven::Device::querySwapChainSupport(const VkPhysicalDevice device) const
00335 {
00336     SwapChainSupportDetails details;
00337     vkGetPhysicalDeviceSurfaceCapabilitiesKHR(device, surface_, &details.capabilities);
00338     uint32_t formatCount = 0;
00339 
00340     vkGetPhysicalDeviceSurfaceFormatsKHR(device, surface_, &formatCount, nullptr);
00341     if (formatCount != 0) {
00342         details.formats.resize(formatCount);
00343         vkGetPhysicalDeviceSurfaceFormatsKHR(device, surface_, &formatCount, details.formats.data());
00344     }
00345     uint32_t presentModeCount = 0;
00346     vkGetPhysicalDeviceSurfacePresentModesKHR(device, surface_, &presentModeCount, nullptr);
00347     if (presentModeCount != 0) {
00348         details.presentModes.resize(presentModeCount);
00349         vkGetPhysicalDeviceSurfacePresentModesKHR(device, surface_, &presentModeCount,
    details.presentModes.data());
00350     }
00351 
00352     return details;
00353 }
```

```
00354
00355 VkFormat ven::Device::findSupportedFormat(const std::vector<VkFormat> &candidates, const VkImageTiling
      tiling, const VkFormatFeatureFlags features) const
00356 {
00357     for (const VkFormat format : candidates) {
00358         VkFormatProperties props;
00359         vkGetPhysicalDeviceFormatProperties(physicalDevice, format, &props);
00360         if (tiling == VK_IMAGE_TILING_LINEAR && (props.linearTilingFeatures & features) == features) {
00361             return format;
00362         } if (tiling == VK_IMAGE_TILING_OPTIMAL && (props.optimalTilingFeatures & features) ==
      features) {
00363             return format;
00364         }
00365     }
00366     throw std::runtime_error("failed to find supported format!");
00367 }
00368
00369 uint32_t ven::Device::findMemoryType(const uint32_t typeFilter, const VkMemoryPropertyFlags
      propertiesp) const
00370 {
00371     VkPhysicalDeviceMemoryProperties memProperties;
00372     vkGetPhysicalDeviceMemoryProperties(physicalDevice, &memProperties);
00373
00374     for (uint32_t i = 0; i < memProperties.memoryTypeCount; i++) {
00375         if (((typeFilter & (1 << i)) != 0U) &&
00376         (memProperties.memoryTypes[i].propertyFlags & propertiesp) == propertiesp) {
00377             return i;
00378         }
00379     }
00380
00381     throw std::runtime_error("failed to find suitable m_memory type!");
00382 }
00383
00384 void ven::Device::createBuffer(const VkDeviceSize size, const VkBufferUsageFlags usage, const
      VkMemoryPropertyFlags propertiesp, VkBuffer &buffer, VkDeviceMemory &bufferMemory) const
00385 {
00386     VkBufferCreateInfo bufferInfo{};
00387     bufferInfo.sType = VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO;
00388     bufferInfo.size = size;
00389     bufferInfo.usage = usage;
00390     bufferInfo.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
00391
00392     if (vkCreateBuffer(device_, &bufferInfo, nullptr, &buffer) != VK_SUCCESS) {
00393         throw std::runtime_error("failed to create vertex m_buffer!");
00394     }
00395
00396     VkMemoryRequirements memRequirements;
00397     vkGetBufferMemoryRequirements(device_, buffer, &memRequirements);
00398
00399     VkMemoryAllocateInfo allocInfo{};
00400     allocInfo.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
00401     allocInfo.allocationSize = memRequirements.size;
00402     allocInfo.memoryTypeIndex = findMemoryType(memRequirements.memoryTypeBits, propertiesp);
00403
00404     if (vkAllocateMemory(device_, &allocInfo, nullptr, &bufferMemory) != VK_SUCCESS) {
00405         throw std::runtime_error("failed to allocate vertex m_buffer m_memory!");
00406     }
00407
00408     vkBindBufferMemory(device_, buffer, bufferMemory, 0);
00409 }
00410
00411 VkCommandBuffer ven::Device::beginSingleTimeCommands() const
00412 {
00413     VkCommandBufferAllocateInfo allocInfo{};
00414     allocInfo.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO;
00415     allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
00416     allocInfo.commandPool = commandPool;
00417     allocInfo.commandBufferCount = 1;
00418
00419     VkCommandBuffer commandBuffer = nullptr;
00420     vkAllocateCommandBuffers(device_, &allocInfo, &commandBuffer);
00421
00422     VkCommandBufferBeginInfo beginInfo{};
00423     beginInfo.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO;
00424     beginInfo.flags = VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT;
00425
00426     vkBeginCommandBuffer(commandBuffer, &beginInfo);
00427     return commandBuffer;
00428 }
00429
00430 void ven::Device::endSingleTimeCommands(const VkCommandBuffer commandBuffer) const
00431 {
00432     vkEndCommandBuffer(commandBuffer);
00433
00434     VkSubmitInfo submitInfo{};
00435     submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
00436     submitInfo.commandBufferCount = 1;
```

```
00437     submitInfo.pCommandBuffers = &commandBuffer;
00438
00439     vkQueueSubmit(graphicsQueue_, 1, &submitInfo, VK_NULL_HANDLE);
00440     vkQueueWaitIdle(graphicsQueue_);
00441
00442     vkFreeCommandBuffers(device_, commandPool, 1, &commandBuffer);
00443 }
00444
00445 void ven::Device::copyBuffer(const VkBuffer srcBuffer, const VkBuffer dstBuffer, const VkDeviceSize
      size) const
00446 {
00447     const VkCommandBuffer commandBuffer = beginSingleTimeCommands();
00448
00449     VkBufferCopy copyRegion{};
00450     copyRegion.srcOffset = 0;  // Optional
00451     copyRegion.dstOffset = 0;  // Optional
00452     copyRegion.size = size;
00453     vkCmdCopyBuffer(commandBuffer, srcBuffer, dstBuffer, 1, &copyRegion);
00454
00455     endSingleTimeCommands(commandBuffer);
00456 }
00457
00458 void ven::Device::copyBufferToImage(const VkBuffer buffer, const VkImage image, const uint32_t width,
      const uint32_t height, const uint32_t layerCount) const
00459 {
00460     const VkCommandBuffer commandBuffer = beginSingleTimeCommands();
00461
00462     VkBufferImageCopy region{};
00463     region.bufferOffset = 0;
00464     region.bufferRowLength = 0;
00465     region.bufferImageHeight = 0;
00466
00467     region.imageSubresource.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
00468     region.imageSubresource.mipLevel = 0;
00469     region.imageSubresource.baseArrayLayer = 0;
00470     region.imageSubresource.layerCount = layerCount;
00471
00472     region.imageOffset = {0, 0, 0};
00473     region.imageExtent = {width, height, 1};
00474
00475     vkCmdCopyBufferToImage(commandBuffer, buffer, image, VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL, 1,
      &region);
00476     endSingleTimeCommands(commandBuffer);
00477 }
00478
00479 void ven::Device::createImageWithInfo(const VkImageCreateInfo &imageInfo, const VkMemoryPropertyFlags
      properties, VkImage &image, VkDeviceMemory &imageMemory) const
00480 {
00481     if (vkCreateImage(device_, &imageInfo, nullptr, &image) != VK_SUCCESS) {
00482         throw std::runtime_error("failed to create image!");
00483     }
00484
00485     VkMemoryRequirements memRequirements;
00486     vkGetImageMemoryRequirements(device_, image, &memRequirements);
00487
00488     VkMemoryAllocateInfo allocInfo{};
00489     allocInfo.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
00490     allocInfo.allocationSize = memRequirements.size;
00491     allocInfo.memoryTypeIndex = findMemoryType(memRequirements.memoryTypeBits, properties);
00492
00493     if (vkAllocateMemory(device_, &allocInfo, nullptr, &imageMemory) != VK_SUCCESS) {
00494         throw std::runtime_error("failed to allocate image m_memory!");
00495     }
00496
00497     if (vkBindImageMemory(device_, image, imageMemory, 0) != VK_SUCCESS) {
00498         throw std::runtime_error("failed to bind image m_memory!");
00499     }
00500 }
```

7.56 /home/runner/work/VEngine/VEngine/src/engine.cpp File Reference

```
#include <chrono>
#include <cmath>
#include <glm/glm.hpp>
#include <glm/gtc/constants.hpp>
#include "VEngine/Engine.hpp"
#include "VEngine/KeyboardController.hpp"
#include "VEngine/System/RenderSystem.hpp"
```

```
#include "VEngine/System/PointLightSystem.hpp"
#include "VEngine/FrameCounter.hpp"
```

Include dependency graph for engine.cpp:

Macros

- #define GLM_FORCE_RADIANS
- #define GLM_FORCE_DEPTH_ZERO_TO_ONE

7.56.1 Macro Definition Documentation

7.56.1.1 GLM_FORCE_DEPTH_ZERO_TO_ONE

```
#define GLM_FORCE_DEPTH_ZERO_TO_ONE
```

Definition at line 5 of file engine.cpp.

7.56.1.2 GLM_FORCE_RADIANS

```
#define GLM_FORCE_RADIANS
```

Definition at line 4 of file engine.cpp.

7.57 engine.cpp

Go to the documentation of this file.

```
00001 #include <chrono>
00002 #include <cmath>
00003
00004 #define GLM_FORCE_RADIANS
00005 #define GLM_FORCE_DEPTH_ZERO_TO_ONE
00006 #include <glm/glm.hpp>
00007 #include <glm/gtc/constants.hpp>
00008
00009 #include "VEngine/Engine.hpp"
00010 #include "VEngine/KeyboardController.hpp"
00011 #include "VEngine/System/RenderSystem.hpp"
00012 #include "VEngine/System/PointLightSystem.hpp"
00013 #include "VEngine/FrameCounter.hpp"
00014
00015
00016 void ven::Engine::loadObjects()
00017 {
00018     std::shared_ptr model = Model::createModelFromFile(m_device, "models/flat_vase.obj");
00019
00020     Object flatVase = Object::createObject();
00021     flatVase.model = model;
00022     flatVase.transform3D.translation = {-.5F, .5F, 0.F};
00023     flatVase.transform3D.scale = {3.F, 1.5F, 3.F};
00024     m_objects.emplace(flatVase.getId(), std::move(flatVase));
00025
00026     model = Model::createModelFromFile(m_device, "models/smooth_vase.obj");
00027     Object smoothVase = Object::createObject();
00028     smoothVase.model = model;
00029     smoothVase.transform3D.translation = {.5F, .5F, 0.F};
00030     smoothVase.transform3D.scale = {3.F, 1.5F, 3.F};
00031     m_objects.emplace(smoothVase.getId(), std::move(smoothVase));
00032
00033     model = Model::createModelFromFile(m_device, "models/quad.obj");
00034     Object floor = Object::createObject();
00035     floor.model = model;
00036     floor.transform3D.translation = {0.F, .5F, 0.F};
00037     floor.transform3D.scale = {3.F, 1.F, 3.F};
```

```
00038     m_objects.emplace(floor.getId(), std::move(floor));
00039
00040     std::vector<glm::vec3> lightColors{
00041             {1.F, .1F, .1F},
00042             {.1F, .1F, 1.F},
00043             {.1F, 1.F, .1F},
00044             {1.F, 1.F, .1F},
00045             {.1F, 1.F, 1.F},
00046             {1.F, 1.F, 1.F}
00047     };
00048
00049     for (std::size_t i = 0; i < lightColors.size(); i++)
00050     {
00051         Object pointLight = Object::makePointLight(0.2F);
00052         pointLight.color = lightColors[i];
00053         auto rotateLight = rotate(glm::mat4(1.F), (static_cast<float>(i) * glm::two_pi<float>()) /
      static_cast<float>(lightColors.size()), {0.F, -1.F, 0.F});
00054         pointLight.transform3D.translation = glm::vec3(rotateLight * glm::vec4(-1.F, -1.F, -1.F,
      1.F));
00055         m_objects.emplace(pointLight.getId(), std::move(pointLight));
00056     }
00057 }
00058
00059 ven::Engine::Engine(const uint32_t width, const uint32_t height, const std::string &title) :
      m_window(width, height, title)
00060 {
00061     createInstance();
00062     createSurface();
00063     m_globalPool =
      DescriptorPool::Builder(m_device).setMaxSets(SwapChain::MAX_FRAMES_IN_FLIGHT).addPoolSize(VK_DESCRIPTOR_TYPE_UNIFORM_BU
      SwapChain::MAX_FRAMES_IN_FLIGHT).build();
00064     loadObjects();
00065 }
00066
00067 void ven::Engine::mainLoop()
00068 {
00069     Camera camera{};
00070     FrameCounter frameCounter{};
00071     KeyboardController cameraController{};
00072     Object viewerObject = Object::createObject();
00073     std::chrono::time_point<std::chrono::system_clock> newTime;
00074     std::chrono::time_point<std::chrono::system_clock> currentTime =
      std::chrono::high_resolution_clock::now();
00075     std::chrono::duration<float> deltaTime{};
00076     float frameTime = NAN;
00077     int frameIndex = 0;
00078     std::unique_ptr<DescriptorSetLayout> globalSetLayout =
      DescriptorSetLayout::Builder(m_device).addBinding(0, VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER,
      VK_SHADER_STAGE_ALL_GRAPHICS).build();
00079     std::vector<std::unique_ptr<Buffer» uboBuffers(SwapChain::MAX_FRAMES_IN_FLIGHT);
00080     std::vector<VkDescriptorSet> globalDescriptorSets(SwapChain::MAX_FRAMES_IN_FLIGHT);
00081     RenderSystem renderSystem(m_device, m_renderer.getSwapChainRenderPass(),
      globalSetLayout->getDescriptorSetLayout());
00082     PointLightSystem pointLightSystem(m_device, m_renderer.getSwapChainRenderPass(),
      globalSetLayout->getDescriptorSetLayout());
00083
00084     for (auto & uboBuffer : uboBuffers)
00085     {
00086         uboBuffer = std::make_unique<Buffer>(m_device, sizeof(GlobalUbo), 1,
      VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT, VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT);
00087         uboBuffer->map();
00088     }
00089     for (std::size_t i = 0; i < globalDescriptorSets.size(); i++) {
00090         VkDescriptorBufferInfo bufferInfo = uboBuffers[i]->descriptorInfo();
00091         DescriptorWriter(*globalSetLayout, *m_globalPool).writeBuffer(0,
      &bufferInfo).build(globalDescriptorSets[i]);
00092     }
00093     camera.setViewTarget(glm::vec3(-1.F, -2.F, -2.F), glm::vec3(0.F, 0.F, 2.5F));
00094     viewerObject.transform3D.translation.z = -2.5F;
00095
00096     while (glfwWindowShouldClose(m_window.getGLFWindow()) == 0)
00097     {
00098         glfwPollEvents();
00099
00100         newTime = std::chrono::high_resolution_clock::now();
00101         deltaTime = newTime - currentTime;
00102         currentTime = newTime;
00103         frameTime = deltaTime.count();
00104         frameCounter.update(frameTime);
00105
00106         cameraController.moveInPlaneXZ(m_window.getGLFWindow(), frameTime, viewerObject);
00107         camera.setViewYXZ(viewerObject.transform3D.translation, viewerObject.transform3D.rotation);
00108         camera.setPerspectiveProjection(glm::radians(50.0F), m_renderer.getAspectRatio(), 0.1F,
      100.F);
00109
00110         if (VkCommandBuffer_T *commandBuffer = m_renderer.beginFrame())
00111         {
```

```
00112             frameIndex = (m_renderer.getFrameIndex());
00113             FrameInfo frameInfo{frameIndex, frameTime, commandBuffer, camera,
      globalDescriptorSets[static_cast<unsigned long>(frameIndex)], m_objects};
00114
00115             GlobalUbo ubo{};
00116             ubo.projection = camera.getProjection();
00117             ubo.view = camera.getView();
00118             ubo.inverseView = camera.getInverseView();
00119             PointLightSystem::update(frameInfo, ubo);
00120             uboBuffers[static_cast<unsigned long>(frameIndex)]->writeToBuffer(&ubo);
00121             uboBuffers[static_cast<unsigned long>(frameIndex)]->flush();
00122
00123             m_renderer.beginSwapChainRenderPass(commandBuffer);
00124             renderSystem.renderObjects(frameInfo);
00125             pointLightSystem.render(frameInfo);
00126             Renderer::endSwapChainRenderPass(commandBuffer);
00127             m_renderer.endFrame();
00128         }
00129     }
00130     vkDeviceWaitIdle(m_device.device());
00131 }
00132
00133 void ven::Engine::createInstance()
00134 {
00135     VkApplicationInfo appInfo{};
00136     appInfo.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
00137     appInfo.pApplicationName = "VEngine App";
00138     appInfo.applicationVersion = VK_MAKE_API_VERSION(0, 1, 0, 0);
00139     appInfo.pEngineName = "VEngine";
00140     appInfo.engineVersion = VK_MAKE_API_VERSION(0, 1, 0, 0);
00141     appInfo.apiVersion = VK_API_VERSION_1_0;
00142     VkInstanceCreateInfo createInfo{};
00143     createInfo.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
00144     createInfo.pApplicationInfo = &appInfo;
00145     uint32_t glfwExtensionCount = 0;
00146     const char** glfwExtensions = glfwGetRequiredInstanceExtensions(&glfwExtensionCount);
00147     createInfo.enabledExtensionCount = glfwExtensionCount;
00148     createInfo.ppEnabledExtensionNames = glfwExtensions;
00149
00150     if (vkCreateInstance(&createInfo, nullptr, &m_instance) != VK_SUCCESS)
00151     {
00152         throw std::runtime_error("Failed to create Vulkan instance");
00153     }
00154 }
```

7.58 /home/runner/work/VEngine/VEngine/src/keyboardController.cpp File Reference

```
#include <cmath>
#include "VEngine/KeyboardController.hpp"
```

Include dependency graph for keyboardController.cpp:

7.59 keyboardController.cpp

Go to the documentation of this file.

```
00001 #include <cmath>
00002
00003 #include "VEngine/KeyboardController.hpp"
00004
00005 void ven::KeyboardController::moveInPlaneXZ(GLFWwindow* window, float dt, Object& object) const
00006 {
00007     glm::vec3 rotate{0};
00008     if (glfwGetKey(window, m_keys.lookLeft) == GLFW_PRESS) { rotate.y -= 1.F; }
00009     if (glfwGetKey(window, m_keys.lookRight) == GLFW_PRESS) { rotate.y += 1.F; }
00010     if (glfwGetKey(window, m_keys.lookUp) == GLFW_PRESS) { rotate.x += 1.F; }
00011     if (glfwGetKey(window, m_keys.lookDown) == GLFW_PRESS) { rotate.x -= 1.F; }
00012
00013     if (dot(rotate, rotate) > std::numeric_limits<float>::epsilon()) {
00014         object.transform3D.rotation += m_lookSpeed * dt * normalize(rotate);
00015     }
00016
00017     object.transform3D.rotation.x = glm::clamp(object.transform3D.rotation.x, -1.5F, 1.5F);
00018     object.transform3D.rotation.y = glm::mod(object.transform3D.rotation.y, glm::two_pi<float>());
```

```
00019
00020     float yaw = object.transform3D.rotation.y;
00021     const glm::vec3 forwardDir{std::sin(yaw), 0.F, std::cos(yaw)};
00022     const glm::vec3 rightDir{forwardDir.z, 0.F, -forwardDir.x};
00023     constexpr  glm::vec3 upDir{0.F, -1.F, 0.F};
00024
00025     glm::vec3 moveDir{0.F};
00026     if (glfwGetKey(window, m_keys.moveForward) == GLFW_PRESS) {moveDir += forwardDir;}
00027     if (glfwGetKey(window, m_keys.moveBackward) == GLFW_PRESS) {moveDir -= forwardDir;}
00028     if (glfwGetKey(window, m_keys.moveRight) == GLFW_PRESS) {moveDir += rightDir;}
00029     if (glfwGetKey(window, m_keys.moveLeft) == GLFW_PRESS) {moveDir -= rightDir;}
00030     if (glfwGetKey(window, m_keys.moveUp) == GLFW_PRESS) {moveDir += upDir;}
00031     if (glfwGetKey(window, m_keys.moveDown) == GLFW_PRESS) {moveDir -= upDir;}
00032
00033     if (dot(moveDir, moveDir) > std::numeric_limits<float>::epsilon()) {
00034         object.transform3D.translation += m_moveSpeed * dt * normalize(moveDir);
00035     }
00036 }
```

7.60 /home/runner/work/VEngine/VEngine/src/main.cpp File Reference

```
#include <iostream>
#include "VEngine/Engine.hpp"
```

Include dependency graph for main.cpp:

Functions

- int main ()

7.60.1 Function Documentation

7.60.1.1 main()

```
int main ()
```

Definition at line 7 of file main.cpp.

References ven::Engine::mainLoop().

Here is the call graph for this function:

7.61 main.cpp

Go to the documentation of this file.

```
00001 #include <iostream>
00002
00003 #include "VEngine/Engine.hpp"
00004
00005 using namespace ven;
00006
00007 int main()
00008 {
00009     try {
00010         Engine engine{};
00011         engine.mainLoop();
00012     } catch (const std::exception &e) {
00013         std::cerr « "std exception: " « e.what() « '\n';
00014         return VEN_FAILURE;
00015     } catch (...) {
00016         std::cerr « "Unknown error\n";
00017         return VEN_FAILURE;
00018     }
00019     return VEN_SUCCESS;
00020 }
```

7.62 /home/runner/work/VEngine/VEngine/src/model.cpp File Reference

```
#include <cassert>
#include <cstring>
#include <unordered_map>
#include <tiny_obj_loader.h>
#include <glm/gtx/hash.hpp>
#include "VEngine/Model.hpp"
#include "VEngine/Utils.hpp"
```

Include dependency graph for model.cpp:

Classes

- struct std::hash< ven::Model::Vertex >

Namespaces

- namespace std

  STL namespace.

Macros

- #define TINYOBJLOADER_IMPLEMENTATION
- #define GLM_ENABLE_EXPERIMENTAL

7.62.1 Macro Definition Documentation

7.62.1.1 GLM_ENABLE_EXPERIMENTAL

```
#define GLM_ENABLE_EXPERIMENTAL
```

Definition at line 8 of file model.cpp.

7.62.1.2 TINYOBJLOADER_IMPLEMENTATION

```
#define TINYOBJLOADER_IMPLEMENTATION
```

Definition at line 5 of file model.cpp.

7.63 model.cpp

Go to the documentation of this file.

```
00001 #include <cassert>
00002 #include <cstring>
00003 #include <unordered_map>
00004
00005 #define TINYOBJLOADER_IMPLEMENTATION
00006 #include <tiny_obj_loader.h>
00007
00008 #define GLM_ENABLE_EXPERIMENTAL
00009 #include <glm/gtx/hash.hpp>
00010
00011 #include "VEngine/Model.hpp"
00012 #include "VEngine/Utils.hpp"
00013
00014 namespace std {
00015     template<>
00016     struct hash<ven::Model::Vertex> {
00017         size_t operator()(ven::Model::Vertex const &vertex) const {
00018             size_t seed = 0;
00019             ven::hashCombine(seed, vertex.position, vertex.color, vertex.normal, vertex.uv);
00020             return seed;
00021         }
00022     };
00023 }
00024
00025 ven::Model::Model(Device &device, const Builder &builder) : m_device{device}, m_vertexCount(0),
      m_indexCount(0)
00026 {
00027     createVertexBuffer(builder.vertices);
00028     createIndexBuffer(builder.indices);
00029 }
00030
00031 ven::Model::~Model() = default;
00032
00033 void ven::Model::createVertexBuffer(const std::vector<Vertex> &vertices)
00034 {
00035     m_vertexCount = static_cast<uint32_t>(vertices.size());
00036     assert(m_vertexCount >= 3 && "Vertex count must be at least 3");
00037     const VkDeviceSize bufferSize = sizeof(vertices[0]) * m_vertexCount;
00038     uint32_t vertexSize = sizeof(vertices[0]);
00039
00040     Buffer stagingBuffer{m_device, vertexSize, m_vertexCount, VK_BUFFER_USAGE_TRANSFER_SRC_BIT,
      VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_COHERENT_BIT};
00041
00042     stagingBuffer.map();
00043     stagingBuffer.writeToBuffer(vertices.data());
00044
00045     m_vertexBuffer = std::make_unique<Buffer>(m_device, vertexSize, m_vertexCount,
      VK_BUFFER_USAGE_VERTEX_BUFFER_BIT | VK_BUFFER_USAGE_TRANSFER_DST_BIT,
      VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT);
00046
00047     m_device.copyBuffer(stagingBuffer.getBuffer(), m_vertexBuffer->getBuffer(), bufferSize);
00048 }
00049
00050 void ven::Model::createIndexBuffer(const std::vector<uint32_t> &indices)
00051 {
00052     m_indexCount = static_cast<uint32_t>(indices.size());
00053     m_hasIndexBuffer = m_indexCount > 0;
00054
00055     if (!m_hasIndexBuffer) {
00056         return;
00057     }
00058
00059     const VkDeviceSize bufferSize = sizeof(indices[0]) * m_indexCount;
00060     uint32_t indexSize = sizeof(indices[0]);
00061
00062     Buffer stagingBuffer{m_device, indexSize, m_indexCount, VK_BUFFER_USAGE_TRANSFER_SRC_BIT,
      VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT | VK_MEMORY_PROPERTY_HOST_COHERENT_BIT};
00063
00064     stagingBuffer.map();
00065     stagingBuffer.writeToBuffer(indices.data());
00066
00067     m_indexBuffer = std::make_unique<Buffer>(m_device, indexSize, m_indexCount,
      VK_BUFFER_USAGE_INDEX_BUFFER_BIT | VK_BUFFER_USAGE_TRANSFER_DST_BIT,
      VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT);
00068
00069     m_device.copyBuffer(stagingBuffer.getBuffer(), m_indexBuffer->getBuffer(), bufferSize);
00070 }
00071
00072 void ven::Model::draw(const VkCommandBuffer commandBuffer) const
00073 {
00074     if (m_hasIndexBuffer) {
00075         vkCmdDrawIndexed(commandBuffer, m_indexCount, 1, 0, 0, 0);
```

```
00076     } else {
00077         vkCmdDraw(commandBuffer, m_vertexCount, 1, 0, 0);
00078     }
00079 }
00080
00081 void ven::Model::bind(const VkCommandBuffer commandBuffer) const
00082 {
00083     const VkBuffer buffers[] = {m_vertexBuffer->getBuffer()};
00084     constexpr VkDeviceSize offsets[] = {0};
00085     vkCmdBindVertexBuffers(commandBuffer, 0, 1, buffers, offsets);
00086
00087     if (m_hasIndexBuffer) {
00088         vkCmdBindIndexBuffer(commandBuffer, m_indexBuffer->getBuffer(), 0, VK_INDEX_TYPE_UINT32);
00089     }
00090 }
00091
00092 std::unique_ptr<ven::Model> ven::Model::createModelFromFile(Device &device, const std::string
      &filename)
00093 {
00094     Builder builder{};
00095     builder.loadModel(filename);
00096     return std::make_unique<Model>(device, builder);
00097 }
00098
00099 std::vector<VkVertexInputBindingDescription> ven::Model::Vertex::getBindingDescriptions()
00100 {
00101     std::vector<VkVertexInputBindingDescription> bindingDescriptions(1);
00102     bindingDescriptions[0].binding = 0;
00103     bindingDescriptions[0].stride = sizeof(Vertex);
00104     bindingDescriptions[0].inputRate = VK_VERTEX_INPUT_RATE_VERTEX;
00105     return bindingDescriptions;
00106 }
00107
00108 std::vector<VkVertexInputAttributeDescription> ven::Model::Vertex::getAttributeDescriptions()
00109 {
00110     std::vector<VkVertexInputAttributeDescription> attributeDescriptions{};
00111
00112     attributeDescriptions.push_back({0, 0, VK_FORMAT_R32G32B32_SFLOAT, offsetof(Vertex, position)});
00113     attributeDescriptions.push_back({1, 0, VK_FORMAT_R32G32B32_SFLOAT, offsetof(Vertex, color)});
00114     attributeDescriptions.push_back({2, 0, VK_FORMAT_R32G32B32_SFLOAT, offsetof(Vertex, normal)});
00115     attributeDescriptions.push_back({3, 0, VK_FORMAT_R32G32_SFLOAT, offsetof(Vertex, uv)});
00116
00117     return attributeDescriptions;
00118 }
00119
00120 void ven::Model::Builder::loadModel(const std::string &filename)
00121 {
00122     tinyobj::attrib_t attrib;
00123     std::vector<tinyobj::shape_t> shapes;
00124     std::vector<tinyobj::material_t> materials;
00125     std::string warn;
00126     std::string err;
00127
00128     if (!LoadObj(&attrib, &shapes, &materials, &warn, &err, filename.c_str()))
00129     {
00130         throw std::runtime_error(warn + err);
00131     }
00132
00133     vertices.clear();
00134     indices.clear();
00135
00136     std::unordered_map<Vertex, uint32_t> uniqueVertices{};
00137     for (const auto &shape : shapes) {
00138         for (const auto &index : shape.mesh.indices) {
00139             Vertex vertex{};
00140             if (index.vertex_index >= 0) {
00141                 vertex.position = {
00142                         attrib.vertices[3 * static_cast<size_t>(index.vertex_index) + 0],
00143                         attrib.vertices[3 * static_cast<size_t>(index.vertex_index) + 1],
00144                         attrib.vertices[3 * static_cast<size_t>(index.vertex_index) + 2]
00145                 };
00146
00147                 vertex.color = {
00148                         attrib.colors[3 * static_cast<size_t>(index.vertex_index) + 0],
00149                         attrib.colors[3 * static_cast<size_t>(index.vertex_index) + 1],
00150                         attrib.colors[3 * static_cast<size_t>(index.vertex_index) + 2]
00151                 };
00152             }
00153
00154             if (index.normal_index >= 0) {
00155                 vertex.normal = {
00156                         attrib.normals[3 * static_cast<size_t>(index.normal_index) + 0],
00157                         attrib.normals[3 * static_cast<size_t>(index.normal_index) + 1],
00158                         attrib.normals[3 * static_cast<size_t>(index.normal_index) + 2]
00159                 };
00160             }
00161
```

```
00162             if (index.texcoord_index >= 0) {
00163                 vertex.uv = {
00164                         attrib.texcoords[2 * static_cast<size_t>(index.texcoord_index) + 0],
00165                         attrib.texcoords[2 * static_cast<size_t>(index.texcoord_index) + 1]
00166                 };
00167             }
00168
00169             if (!uniqueVertices.contains(vertex)) {
00170                 uniqueVertices[vertex] = static_cast<uint32_t>(vertices.size());
00171                 vertices.push_back(vertex);
00172             }
00173             indices.push_back(uniqueVertices[vertex]);
00174         }
00175     }
00176 }
```

7.64 /home/runner/work/VEngine/VEngine/src/object.cpp File Reference

`#include "VEngine/Object.hpp"`
Include dependency graph for object.cpp:

7.65 object.cpp

Go to the documentation of this file.

```
00001 #include "VEngine/Object.hpp"
00002
00003 glm::mat4 ven::Transform3DComponent::mat4() const {
00004     const float c3 = glm::cos(rotation.z);
00005     const float s3 = glm::sin(rotation.z);
00006     const float c2 = glm::cos(rotation.x);
00007     const float s2 = glm::sin(rotation.x);
00008     const float c1 = glm::cos(rotation.y);
00009     const float s1 = glm::sin(rotation.y);
00010     return glm::mat4{
00011             {
00012                     scale.x * (c1 * c3 + s1 * s2 * s3),
00013                     scale.x * (c2 * s3),
00014                     scale.x * (c1 * s2 * s3 - c3 * s1),
00015                     0.0F,
00016             },
00017             {
00018                     scale.y * (c3 * s1 * s2 - c1 * s3),
00019                     scale.y * (c2 * c3),
00020                     scale.y * (c1 * c3 * s2 + s1 * s3),
00021                     0.0F,
00022             },
00023             {
00024                     scale.z * (c2 * s1),
00025                     scale.z * (-s2),
00026                     scale.z * (c1 * c2),
00027                     0.0F,
00028             },
00029             {
00030                     translation.x,
00031                     translation.y,
00032                     translation.z,
00033                     1.0F
00034             }
00035     };
00036 }
00037
00038 glm::mat3 ven::Transform3DComponent::normalMatrix() const
00039 {
00040     const float c3 = glm::cos(rotation.z);
00041     const float s3 = glm::sin(rotation.z);
00042     const float c2 = glm::cos(rotation.x);
00043     const float s2 = glm::sin(rotation.x);
00044     const float c1 = glm::cos(rotation.y);
00045     const float s1 = glm::sin(rotation.y);
00046     const glm::vec3 invScale = 1.0F / scale;
00047
00048     return glm::mat3{
00049             {
00050                     invScale.x * (c1 * c3 + s1 * s2 * s3),
00051                     invScale.x * (c2 * s3),
```

```
00052                       invScale.x * (c1 * s2 * s3 - c3 * s1)
00053             },
00054             {
00055                       invScale.y * (c3 * s1 * s2 - c1 * s3),
00056                       invScale.y * (c2 * c3),
00057                       invScale.y * (c1 * c3 * s2 + s1 * s3)
00058             },
00059             {
00060                       invScale.z * (c2 * s1),
00061                       invScale.z * (-s2),
00062                       invScale.z * (c1 * c2)
00063             }
00064     };
00065 }
00066
00067 ven::Object ven::Object::makePointLight(const float intensity, const float radius, const glm::vec3
      color)
00068 {
00069     Object obj = Object::createObject();
00070     obj.color = color;
00071     obj.transform3D.scale.x = radius;
00072     obj.pointLight = std::make_unique<PointLightComponent>();
00073     obj.pointLight->lightIntensity = intensity;
00074     return obj;
00075 }
```

7.66 /home/runner/work/VEngine/VEngine/src/renderer.cpp File Reference

`#include "VEngine/Renderer.hpp"`
Include dependency graph for renderer.cpp:

7.67 renderer.cpp

Go to the documentation of this file.

```
00001 #include "VEngine/Renderer.hpp"
00002
00003 void ven::Renderer::createCommandBuffers()
00004 {
00005     m_commandBuffers.resize(SwapChain::MAX_FRAMES_IN_FLIGHT);
00006     VkCommandBufferAllocateInfo allocInfo{};
00007     allocInfo.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO;
00008     allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
00009     allocInfo.commandPool = m_device.getCommandPool();
00010     allocInfo.commandBufferCount = static_cast<uint32_t>(m_commandBuffers.size());
00011
00012     if (vkAllocateCommandBuffers(m_device.device(), &allocInfo, m_commandBuffers.data()) !=
      VK_SUCCESS) {
00013         throw std::runtime_error("Failed to allocate command buffers");
00014     }
00015 }
00016
00017 void ven::Renderer::freeCommandBuffers()
00018 {
00019     vkFreeCommandBuffers(m_device.device(), m_device.getCommandPool(),
      static_cast<uint32_t>(m_commandBuffers.size()), m_commandBuffers.data());
00020     m_commandBuffers.clear();
00021 }
00022
00023 void ven::Renderer::recreateSwapChain()
00024 {
00025     VkExtent2D extent = m_window.getExtent();
00026     while (extent.width == 0 || extent.height == 0) {
00027         extent = m_window.getExtent();
00028         glfwWaitEvents();
00029     }
00030     vkDeviceWaitIdle(m_device.device());
00031     if (m_swapChain == nullptr) {
00032         m_swapChain = std::make_unique<SwapChain>(m_device, extent);
00033     } else {
00034         std::shared_ptr<SwapChain> oldSwapChain = std::move(m_swapChain);
00035         m_swapChain = std::make_unique<SwapChain>(m_device, extent, oldSwapChain);
00036         if (!oldSwapChain->compareSwapFormats(*m_swapChain)) {
```

```
00037             throw std::runtime_error("Swap chain image/depth format changed");
00038         }
00039     }
00040     // well be back
00041 }
00042
00043 VkCommandBuffer ven::Renderer::beginFrame()
00044 {
00045     assert(!isFrameStarted && "Can't start new frame while previous one is still in progress");
00046
00047     const VkResult result = m_swapChain->acquireNextImage(&m_currentImageIndex);
00048     if (result == VK_ERROR_OUT_OF_DATE_KHR) {
00049         recreateSwapChain();
00050         return nullptr;
00051     }
00052
00053     if (result != VK_SUCCESS && result != VK_SUBOPTIMAL_KHR) {
00054         throw std::runtime_error("Failed to acquire swap chain image");
00055     }
00056
00057     m_isFrameStarted = true;
00058
00059     VkCommandBuffer_T *commandBuffer = getCurrentCommandBuffer();
00060     VkCommandBufferBeginInfo beginInfo{};
00061     beginInfo.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO;
00062
00063     if (vkBeginCommandBuffer(commandBuffer, &beginInfo) != VK_SUCCESS) {
00064         throw std::runtime_error("Failed to begin recording command m_buffer");
00065     }
00066     return commandBuffer;
00067 }
00068
00069 void ven::Renderer::endFrame()
00070 {
00071     assert(isFrameStarted && "Can't end frame that hasn't been started");
00072
00073     VkCommandBuffer_T *commandBuffer = getCurrentCommandBuffer();
00074     if (vkEndCommandBuffer(commandBuffer) != VK_SUCCESS) {
00075         throw std::runtime_error("Failed to record command m_buffer");
00076     }
00077     VkResult result = m_swapChain->submitCommandBuffers(&commandBuffer, &m_currentImageIndex);
00078     if (result == VK_ERROR_OUT_OF_DATE_KHR || result == VK_SUBOPTIMAL_KHR ||
    m_window.wasWindowResized()) {
00079         m_window.resetWindowResizedFlag();
00080         recreateSwapChain();
00081     }
00082     else if (result != VK_SUCCESS) {
00083         throw std::runtime_error("Failed to submit command m_buffer");
00084     }
00085
00086     m_isFrameStarted = false;
00087     m_currentFrameIndex = (m_currentFrameIndex + 1) % SwapChain::MAX_FRAMES_IN_FLIGHT;
00088 }
00089
00090 void ven::Renderer::beginSwapChainRenderPass(const VkCommandBuffer commandBuffer) const
00091 {
00092     assert(isFrameStarted && "Can't begin render pass when frame not in progress");
00093     assert(commandBuffer == getCurrentCommandBuffer() && "Can't begin render pass on command m_buffer
    from a different frame");
00094
00095     VkRenderPassBeginInfo renderPassInfo{};
00096     renderPassInfo.sType = VK_STRUCTURE_TYPE_RENDER_PASS_BEGIN_INFO;
00097     renderPassInfo.renderPass = m_swapChain->getRenderPass();
00098     renderPassInfo.framebuffer = m_swapChain->getFrameBuffer(m_currentImageIndex);
00099
00100     renderPassInfo.renderArea.offset = {0, 0};
00101     renderPassInfo.renderArea.extent = m_swapChain->getSwapChainExtent();
00102
00103     std::array<VkClearValue, 2> clearValues{};
00104     clearValues[0].color = {{0.01F, 0.01F, 0.01F, 1.0F}};
00105     clearValues[1].depthStencil = {1.0F, 0};
00106     renderPassInfo.clearValueCount = static_cast<uint32_t>(clearValues.size());
00107     renderPassInfo.pClearValues = clearValues.data();
00108
00109     vkCmdBeginRenderPass(commandBuffer, &renderPassInfo, VK_SUBPASS_CONTENTS_INLINE);
00110
00111     VkViewport viewport{};
00112     viewport.x = 0.0F;
00113     viewport.y = 0.0F;
00114     viewport.width = static_cast<float>(m_swapChain->getSwapChainExtent().width);
00115     viewport.height = static_cast<float>(m_swapChain->getSwapChainExtent().height);
00116     viewport.minDepth = 0.0F;
00117     viewport.maxDepth = 1.0F;
00118     const VkRect2D scissor{{0, 0}, m_swapChain->getSwapChainExtent()};
00119     vkCmdSetViewport(commandBuffer, 0, 1, &viewport);
00120     vkCmdSetScissor(commandBuffer, 0, 1, &scissor);
00121 }
```

```
00122
00123 void ven::Renderer::endSwapChainRenderPass(const VkCommandBuffer commandBuffer)
00124 {
00125     assert(isFrameStarted && "Can't end render pass when frame not in progress");
00126     assert(commandBuffer == getCurrentCommandBuffer() && "Can't end render pass on command m_buffer
      from a different frame");
00127
00128     vkCmdEndRenderPass(commandBuffer);
00129 }
```

7.68 /home/runner/work/VEngine/VEngine/src/shaders.cpp File Reference

```
#include <stdexcept>
#include <fstream>
#include "VEngine/Shaders.hpp"
```

Include dependency graph for shaders.cpp:

7.69 shaders.cpp

Go to the documentation of this file.

```
00001 #include <stdexcept>
00002 #include <fstream>
00003
00004 #include "VEngine/Shaders.hpp"
00005
00006 ven::Shaders::~Shaders()
00007 {
00008     vkDestroyShaderModule(m_device.device(), m_vertShaderModule, nullptr);
00009     vkDestroyShaderModule(m_device.device(), m_fragShaderModule, nullptr);
00010     vkDestroyPipeline(m_device.device(), m_graphicsPipeline, nullptr);
00011 }
00012
00013 std::vector<char> ven::Shaders::readFile(const std::string &filename)
00014 {
00015     std::ifstream file(filename, std::ios::ate | std::ios::binary);
00016
00017     if (!file.is_open()) {
00018         throw std::runtime_error("failed to open file!");
00019     }
00020
00021     const std::streamsize fileSize = file.tellg();
00022     std::vector<char> buffer(static_cast<unsigned long>(fileSize));
00023
00024     file.seekg(0);
00025     file.read(buffer.data(), fileSize);
00026
00027     file.close();
00028     return buffer;
00029 }
00030
00031 void ven::Shaders::createGraphicsPipeline(const std::string& vertFilepath, const std::string&
      fragFilepath, const PipelineConfigInfo& configInfo)
00032 {
00033     const std::vector<char> vertCode = readFile(vertFilepath);
00034     const std::vector<char> fragCode = readFile(fragFilepath);
00035
00036     createShaderModule(vertCode, &m_vertShaderModule);
00037     createShaderModule(fragCode, &m_fragShaderModule);
00038
00039     VkPipelineShaderStageCreateInfo shaderStages[2];
00040     shaderStages[0].sType = VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_CREATE_INFO;
00041     shaderStages[0].stage = VK_SHADER_STAGE_VERTEX_BIT;
00042     shaderStages[0].module = m_vertShaderModule;
00043     shaderStages[0].pName = "main";
00044     shaderStages[0].flags = 0;
00045     shaderStages[0].pNext = nullptr;
00046     shaderStages[0].pSpecializationInfo = nullptr;
00047
00048     shaderStages[1].sType = VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_CREATE_INFO;
00049     shaderStages[1].stage = VK_SHADER_STAGE_FRAGMENT_BIT;
00050     shaderStages[1].module = m_fragShaderModule;
```

```
00051     shaderStages[1].pName = "main";
00052     shaderStages[1].flags = 0;
00053     shaderStages[1].pNext = nullptr;
00054     shaderStages[1].pSpecializationInfo = nullptr;
00055 
00056     const auto& bindingDescriptions = configInfo.bindingDescriptions;
00057     const auto& attributeDescriptions = configInfo.attributeDescriptions;
00058     VkPipelineVertexInputStateCreateInfo vertexInputInfo{};
00059     vertexInputInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_STATE_CREATE_INFO;
00060     vertexInputInfo.vertexAttributeDescriptionCount =
      static_cast<uint32_t>(attributeDescriptions.size());
00061     vertexInputInfo.vertexBindingDescriptionCount = static_cast<uint32_t>(bindingDescriptions.size());
00062     vertexInputInfo.pVertexAttributeDescriptions = attributeDescriptions.data();
00063     vertexInputInfo.pVertexBindingDescriptions = bindingDescriptions.data();
00064 
00065 
00066     VkPipelineViewportStateCreateInfo viewportInfo{};
00067     viewportInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_STATE_CREATE_INFO;
00068     viewportInfo.viewportCount = 1;
00069     viewportInfo.pViewports = nullptr;
00070     viewportInfo.scissorCount = 1;
00071     viewportInfo.pScissors = nullptr;
00072 
00073 
00074     VkGraphicsPipelineCreateInfo pipelineInfo{};
00075     pipelineInfo.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
00076     pipelineInfo.stageCount = 2;
00077     pipelineInfo.pStages = shaderStages;
00078     pipelineInfo.pVertexInputState = &vertexInputInfo;
00079     pipelineInfo.pInputAssemblyState = &configInfo.inputAssemblyInfo;
00080     pipelineInfo.pViewportState = &viewportInfo;
00081     pipelineInfo.pRasterizationState = &configInfo.rasterizationInfo;
00082     pipelineInfo.pMultisampleState = &configInfo.multisampleInfo;
00083 
00084     pipelineInfo.pColorBlendState = &configInfo.colorBlendInfo;
00085     pipelineInfo.pDepthStencilState = &configInfo.depthStencilInfo;
00086     pipelineInfo.pDynamicState = &configInfo.dynamicStateInfo;
00087 
00088     pipelineInfo.layout = configInfo.pipelineLayout;
00089     pipelineInfo.renderPass = configInfo.renderPass;
00090     pipelineInfo.subpass = configInfo.subpass;
00091 
00092     pipelineInfo.basePipelineIndex = -1;
00093     pipelineInfo.basePipelineHandle = VK_NULL_HANDLE;
00094 
00095     if (vkCreateGraphicsPipelines(m_device.device(), VK_NULL_HANDLE, 1, &pipelineInfo, nullptr,
      &m_graphicsPipeline) != VK_SUCCESS) {
00096         throw std::runtime_error("failed to create graphics pipeline");
00097     }
00098 }
00099 
00100 void ven::Shaders::createShaderModule(const std::vector<char> &code, VkShaderModule *shaderModule)
      const
00101 {
00102     VkShaderModuleCreateInfo createInfo{};
00103     createInfo.sType = VK_STRUCTURE_TYPE_SHADER_MODULE_CREATE_INFO;
00104     createInfo.codeSize = code.size();
00105     createInfo.pCode = reinterpret_cast<const uint32_t*>(code.data());
00106 
00107     if (vkCreateShaderModule(m_device.device(), &createInfo, nullptr, shaderModule) != VK_SUCCESS) {
00108         throw std::runtime_error("failed to create shader module");
00109     }
00110 }
00111 
00112 void ven::Shaders::defaultPipelineConfigInfo(PipelineConfigInfo& configInfo)
00113 {
00114     configInfo.inputAssemblyInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_INPUT_ASSEMBLY_STATE_CREATE_INFO;
00115     configInfo.inputAssemblyInfo.topology = VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST;
00116     configInfo.inputAssemblyInfo.primitiveRestartEnable = VK_FALSE;
00117 
00118     configInfo.rasterizationInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_STATE_CREATE_INFO;
00119     configInfo.rasterizationInfo.depthClampEnable = VK_FALSE;
00120     configInfo.rasterizationInfo.rasterizerDiscardEnable = VK_FALSE;
00121     configInfo.rasterizationInfo.polygonMode = VK_POLYGON_MODE_FILL;
00122     configInfo.rasterizationInfo.lineWidth = 1.0F;
00123     configInfo.rasterizationInfo.cullMode = VK_CULL_MODE_NONE;
00124     configInfo.rasterizationInfo.frontFace = VK_FRONT_FACE_COUNTER_CLOCKWISE;
00125     configInfo.rasterizationInfo.depthBiasEnable = VK_FALSE;
00126     configInfo.rasterizationInfo.depthBiasConstantFactor = 0.0F;
00127     configInfo.rasterizationInfo.depthBiasClamp = 0.0F;
00128     configInfo.rasterizationInfo.depthBiasSlopeFactor = 0.0F;
00129 
00130     configInfo.multisampleInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_MULTISAMPLE_STATE_CREATE_INFO;
00131     configInfo.multisampleInfo.sampleShadingEnable = VK_FALSE;
00132     configInfo.multisampleInfo.rasterizationSamples = VK_SAMPLE_COUNT_1_BIT;
00133     configInfo.multisampleInfo.minSampleShading = 1.0F;
00134     configInfo.multisampleInfo.pSampleMask = nullptr;
```

```
00135     configInfo.multisampleInfo.alphaToCoverageEnable = VK_FALSE;
00136     configInfo.multisampleInfo.alphaToOneEnable = VK_FALSE;
00137
00138     configInfo.colorBlendAttachment.colorWriteMask = VK_COLOR_COMPONENT_R_BIT |
      VK_COLOR_COMPONENT_G_BIT | VK_COLOR_COMPONENT_B_BIT | VK_COLOR_COMPONENT_A_BIT;
00139     configInfo.colorBlendAttachment.blendEnable = VK_FALSE;
00140     configInfo.colorBlendAttachment.srcColorBlendFactor = VK_BLEND_FACTOR_ONE;
00141     configInfo.colorBlendAttachment.dstColorBlendFactor = VK_BLEND_FACTOR_ZERO;
00142     configInfo.colorBlendAttachment.colorBlendOp = VK_BLEND_OP_ADD;
00143     configInfo.colorBlendAttachment.srcAlphaBlendFactor = VK_BLEND_FACTOR_ONE;
00144     configInfo.colorBlendAttachment.dstAlphaBlendFactor = VK_BLEND_FACTOR_ZERO;
00145     configInfo.colorBlendAttachment.alphaBlendOp = VK_BLEND_OP_ADD;
00146
00147     configInfo.colorBlendInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_COLOR_BLEND_STATE_CREATE_INFO;
00148     configInfo.colorBlendInfo.logicOpEnable = VK_FALSE;
00149     configInfo.colorBlendInfo.logicOp = VK_LOGIC_OP_COPY;
00150     configInfo.colorBlendInfo.attachmentCount = 1;
00151     configInfo.colorBlendInfo.pAttachments = &configInfo.colorBlendAttachment;
00152     configInfo.colorBlendInfo.blendConstants[0] = 0.0F;
00153     configInfo.colorBlendInfo.blendConstants[1] = 0.0F;
00154     configInfo.colorBlendInfo.blendConstants[2] = 0.0F;
00155     configInfo.colorBlendInfo.blendConstants[3] = 0.0F;
00156
00157     configInfo.depthStencilInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_DEPTH_STENCIL_STATE_CREATE_INFO;
00158     configInfo.depthStencilInfo.depthTestEnable = VK_TRUE;
00159     configInfo.depthStencilInfo.depthWriteEnable = VK_TRUE;
00160     configInfo.depthStencilInfo.depthCompareOp = VK_COMPARE_OP_LESS;
00161     configInfo.depthStencilInfo.depthBoundsTestEnable = VK_FALSE;
00162     configInfo.depthStencilInfo.minDepthBounds = 0.0F;
00163     configInfo.depthStencilInfo.maxDepthBounds = 1.0F;
00164     configInfo.depthStencilInfo.stencilTestEnable = VK_FALSE;
00165     configInfo.depthStencilInfo.front = {};
00166     configInfo.depthStencilInfo.back = {};
00167
00168     configInfo.dynamicStateEnables = {VK_DYNAMIC_STATE_VIEWPORT, VK_DYNAMIC_STATE_SCISSOR};
00169     configInfo.dynamicStateInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_DYNAMIC_STATE_CREATE_INFO;
00170     configInfo.dynamicStateInfo.pDynamicStates = configInfo.dynamicStateEnables.data();
00171     configInfo.dynamicStateInfo.dynamicStateCount =
      static_cast<uint32_t>(configInfo.dynamicStateEnables.size());
00172     configInfo.dynamicStateInfo.flags = 0;
00173     configInfo.bindingDescriptions = Model::Vertex::getBindingDescriptions();
00174     configInfo.attributeDescriptions = Model::Vertex::getAttributeDescriptions();
00175 }
```

7.70 /home/runner/work/VEngine/VEngine/src/swapChain.cpp File Reference

```
#include <iostream>
#include <limits>
#include <stdexcept>
#include "VEngine/SwapChain.hpp"
```

Include dependency graph for swapChain.cpp:

7.71 swapChain.cpp

Go to the documentation of this file.

```
00001 #include <iostream>
00002 #include <limits>
00003 #include <stdexcept>
00004
00005 #include "VEngine/SwapChain.hpp"
00006
00007 ven::SwapChain::~SwapChain()
00008 {
00009     for (VkImageView_T *imageView : swapChainImageViews) {
00010         vkDestroyImageView(device.device(), imageView, nullptr);
00011     }
00012     swapChainImageViews.clear();
00013
00014     if (swapChain != nullptr) {
00015         vkDestroySwapchainKHR(device.device(), swapChain, nullptr);
00016         swapChain = nullptr;
```

```
00017     }
00018
00019     for (size_t i = 0; i < depthImages.size(); i++) {
00020         vkDestroyImageView(device.device(), depthImageViews[i], nullptr);
00021         vkDestroyImage(device.device(), depthImages[i], nullptr);
00022         vkFreeMemory(device.device(), depthImageMemorys[i], nullptr);;
00023     }
00024
00025     for (VkFramebuffer_T *framebuffer : swapChainFramebuffers) {
00026         vkDestroyFramebuffer(device.device(), framebuffer, nullptr);
00027     }
00028
00029     vkDestroyRenderPass(device.device(), renderPass, nullptr);
00030
00031     // cleanup synchronization objects
00032     for (size_t i = 0; i < MAX_FRAMES_IN_FLIGHT; i++) {
00033         vkDestroySemaphore(device.device(), renderFinishedSemaphores[i], nullptr);
00034         vkDestroySemaphore(device.device(), imageAvailableSemaphores[i], nullptr);
00035         vkDestroyFence(device.device(), inFlightFences[i], nullptr);
00036     }
00037 }
00038
00039 void ven::SwapChain::init()
00040 {
00041     createSwapChain();
00042     createImageViews();
00043     createRenderPass();
00044     createDepthResources();
00045     createFramebuffers();
00046     createSyncObjects();
00047 }
00048
00049 VkResult ven::SwapChain::acquireNextImage(uint32_t *imageIndex) const
00050 {
00051     vkWaitForFences(device.device(), 1, &inFlightFences[currentFrame], VK_TRUE,
      std::numeric_limits<uint64_t>::max());
00052
00053     return vkAcquireNextImageKHR(device.device(), swapChain, std::numeric_limits<uint64_t>::max(),
      imageAvailableSemaphores[currentFrame], VK_NULL_HANDLE, imageIndex);;
00054 }
00055
00056 VkResult ven::SwapChain::submitCommandBuffers(const VkCommandBuffer *buffers, const uint32_t
      *imageIndex)
00057 {
00058     if (imagesInFlight[*imageIndex] != VK_NULL_HANDLE) {
00059         vkWaitForFences(device.device(), 1, &imagesInFlight[*imageIndex], VK_TRUE, UINT64_MAX);
00060     }
00061     imagesInFlight[*imageIndex] = inFlightFences[currentFrame];
00062
00063     VkSubmitInfo submitInfo = {};
00064     submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
00065
00066     const VkSemaphore waitSemaphores[] = {imageAvailableSemaphores[currentFrame]};
00067     constexpr VkPipelineStageFlags waitStages[] = {VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT};
00068     submitInfo.waitSemaphoreCount = 1;
00069     submitInfo.pWaitSemaphores = waitSemaphores;
00070     submitInfo.pWaitDstStageMask = waitStages;
00071
00072     submitInfo.commandBufferCount = 1;
00073     submitInfo.pCommandBuffers = buffers;
00074
00075     const VkSemaphore signalSemaphores[] = {renderFinishedSemaphores[currentFrame]};
00076     submitInfo.signalSemaphoreCount = 1;
00077     submitInfo.pSignalSemaphores = signalSemaphores;
00078
00079     vkResetFences(device.device(), 1, &inFlightFences[currentFrame]);
00080     if (vkQueueSubmit(device.graphicsQueue(), 1, &submitInfo, inFlightFences[currentFrame]) !=
      VK_SUCCESS) {
00081         throw std::runtime_error("failed to submit draw command m_buffer!");
00082     }
00083
00084     VkPresentInfoKHR presentInfo = {};
00085     presentInfo.sType = VK_STRUCTURE_TYPE_PRESENT_INFO_KHR;
00086
00087     presentInfo.waitSemaphoreCount = 1;
00088     presentInfo.pWaitSemaphores = signalSemaphores;
00089
00090     const VkSwapchainKHR swapChains[] = {swapChain};
00091     presentInfo.swapchainCount = 1;
00092     presentInfo.pSwapchains = swapChains;
00093
00094     presentInfo.pImageIndices = imageIndex;
00095
00096     const VkResult result = vkQueuePresentKHR(device.presentQueue(), &presentInfo);
00097
00098     currentFrame = (currentFrame + 1) % MAX_FRAMES_IN_FLIGHT;
00099
```

```
00100     return result;
00101 }
00102
00103 void ven::SwapChain::createSwapChain()
00104 {
00105     const SwapChainSupportDetails swapChainSupport = device.getSwapChainSupport();
00106
00107     const VkSurfaceFormatKHR surfaceFormat = chooseSwapSurfaceFormat(swapChainSupport.formats);
00108     const VkPresentModeKHR presentMode = chooseSwapPresentMode(swapChainSupport.presentModes);
00109     const VkExtent2D extent = chooseSwapExtent(swapChainSupport.capabilities);
00110
00111     uint32_t imageCount = swapChainSupport.capabilities.minImageCount + 1;
00112     if (swapChainSupport.capabilities.maxImageCount > 0 && imageCount >
      swapChainSupport.capabilities.maxImageCount) {
00113         imageCount = swapChainSupport.capabilities.maxImageCount;
00114     }
00115
00116     VkSwapchainCreateInfoKHR createInfo = {};
00117     createInfo.sType = VK_STRUCTURE_TYPE_SWAPCHAIN_CREATE_INFO_KHR;
00118     createInfo.surface = device.surface();
00119
00120     createInfo.minImageCount = imageCount;
00121     createInfo.imageFormat = surfaceFormat.format;
00122     createInfo.imageColorSpace = surfaceFormat.colorSpace;
00123     createInfo.imageExtent = extent;
00124     createInfo.imageArrayLayers = 1;
00125     createInfo.imageUsage = VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT;
00126
00127     const QueueFamilyIndices indices = device.findPhysicalQueueFamilies();
00128     const uint32_t queueFamilyIndices[] = {indices.graphicsFamily, indices.presentFamily};
00129
00130     if (indices.graphicsFamily != indices.presentFamily) {
00131         createInfo.imageSharingMode = VK_SHARING_MODE_CONCURRENT;
00132         createInfo.queueFamilyIndexCount = 2;
00133         createInfo.pQueueFamilyIndices = queueFamilyIndices;
00134     } else {
00135         createInfo.imageSharingMode = VK_SHARING_MODE_EXCLUSIVE;
00136         createInfo.queueFamilyIndexCount = 0;       // Optional
00137         createInfo.pQueueFamilyIndices = nullptr;  // Optional
00138     }
00139
00140     createInfo.preTransform = swapChainSupport.capabilities.currentTransform;
00141     createInfo.compositeAlpha = VK_COMPOSITE_ALPHA_OPAQUE_BIT_KHR;
00142
00143     createInfo.presentMode = presentMode;
00144     createInfo.clipped = VK_TRUE;
00145
00146     createInfo.oldSwapchain = oldSwapChain == nullptr ? VK_NULL_HANDLE : oldSwapChain->swapChain;
00147
00148     if (vkCreateSwapchainKHR(device.device(), &createInfo, nullptr, &swapChain) != VK_SUCCESS) {
00149         throw std::runtime_error("failed to create swap chain!");
00150     }
00151
00152     // we only specified a minimum number of images in the swap chain, so the implementation is
00153     // allowed to create a swap chain with more. That's why we'll first query the final number of
00154     // images with vkGetSwapchainImagesKHR, then resize the container and finally call it again to
00155     // retrieve the handles.
00156     vkGetSwapchainImagesKHR(device.device(), swapChain, &imageCount, nullptr);
00157     swapChainImages.resize(imageCount);
00158     vkGetSwapchainImagesKHR(device.device(), swapChain, &imageCount, swapChainImages.data());
00159
00160     swapChainImageFormat = surfaceFormat.format;
00161     m_swapChainExtent = extent;
00162 }
00163
00164 void ven::SwapChain::createImageViews()
00165 {
00166     swapChainImageViews.resize(swapChainImages.size());
00167     for (size_t i = 0; i < swapChainImages.size(); i++) {
00168         VkImageViewCreateInfo viewInfo{};
00169         viewInfo.sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO;
00170         viewInfo.image = swapChainImages[i];
00171         viewInfo.viewType = VK_IMAGE_VIEW_TYPE_2D;
00172         viewInfo.format = swapChainImageFormat;
00173         viewInfo.subresourceRange.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
00174         viewInfo.subresourceRange.baseMipLevel = 0;
00175         viewInfo.subresourceRange.levelCount = 1;
00176         viewInfo.subresourceRange.baseArrayLayer = 0;
00177         viewInfo.subresourceRange.layerCount = 1;
00178
00179         if (vkCreateImageView(device.device(), &viewInfo, nullptr, &swapChainImageViews[i]) !=
      VK_SUCCESS) {
00180             throw std::runtime_error("failed to create texture image view!");
00181         }
00182     }
00183 }
00184
```

```
00185 void ven::SwapChain::createRenderPass()
00186 {
00187     VkAttachmentDescription depthAttachment{};
00188     depthAttachment.format = findDepthFormat();
00189     depthAttachment.samples = VK_SAMPLE_COUNT_1_BIT;
00190     depthAttachment.loadOp = VK_ATTACHMENT_LOAD_OP_CLEAR;
00191     depthAttachment.storeOp = VK_ATTACHMENT_STORE_OP_DONT_CARE;
00192     depthAttachment.stencilLoadOp = VK_ATTACHMENT_LOAD_OP_DONT_CARE;
00193     depthAttachment.stencilStoreOp = VK_ATTACHMENT_STORE_OP_DONT_CARE;
00194     depthAttachment.initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
00195     depthAttachment.finalLayout = VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL;
00196 
00197     VkAttachmentReference depthAttachmentRef{};
00198     depthAttachmentRef.attachment = 1;
00199     depthAttachmentRef.layout = VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL;
00200 
00201     VkAttachmentDescription colorAttachment = {};
00202     colorAttachment.format = getSwapChainImageFormat();
00203     colorAttachment.samples = VK_SAMPLE_COUNT_1_BIT;
00204     colorAttachment.loadOp = VK_ATTACHMENT_LOAD_OP_CLEAR;
00205     colorAttachment.storeOp = VK_ATTACHMENT_STORE_OP_STORE;
00206     colorAttachment.stencilStoreOp = VK_ATTACHMENT_STORE_OP_DONT_CARE;
00207     colorAttachment.stencilLoadOp = VK_ATTACHMENT_LOAD_OP_DONT_CARE;
00208     colorAttachment.initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
00209     colorAttachment.finalLayout = VK_IMAGE_LAYOUT_PRESENT_SRC_KHR;
00210 
00211     VkAttachmentReference colorAttachmentRef = {};
00212     colorAttachmentRef.attachment = 0;
00213     colorAttachmentRef.layout = VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL;
00214 
00215     VkSubpassDescription subpass = {};
00216     subpass.pipelineBindPoint = VK_PIPELINE_BIND_POINT_GRAPHICS;
00217     subpass.colorAttachmentCount = 1;
00218     subpass.pColorAttachments = &colorAttachmentRef;
00219     subpass.pDepthStencilAttachment = &depthAttachmentRef;
00220 
00221     VkSubpassDependency dependency = {};
00222     dependency.srcSubpass = VK_SUBPASS_EXTERNAL;
00223     dependency.srcAccessMask = 0;
00224     dependency.srcStageMask = VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT |
      VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT;
00225     dependency.dstSubpass = 0;
00226     dependency.dstStageMask = VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT |
      VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT;
00227     dependency.dstAccessMask = VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT |
      VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT;
00228 
00229     const std::array<VkAttachmentDescription, 2> attachments = {colorAttachment, depthAttachment};
00230     VkRenderPassCreateInfo renderPassInfo = {};
00231     renderPassInfo.sType = VK_STRUCTURE_TYPE_RENDER_PASS_CREATE_INFO;
00232     renderPassInfo.attachmentCount = static_cast<uint32_t>(attachments.size());
00233     renderPassInfo.pAttachments = attachments.data();
00234     renderPassInfo.subpassCount = 1;
00235     renderPassInfo.pSubpasses = &subpass;
00236     renderPassInfo.dependencyCount = 1;
00237     renderPassInfo.pDependencies = &dependency;
00238 
00239     if (vkCreateRenderPass(device.device(), &renderPassInfo, nullptr, &renderPass) != VK_SUCCESS) {
00240         throw std::runtime_error("failed to create render pass!");
00241     }
00242 }
00243 
00244 void ven::SwapChain::createFramebuffers()
00245 {
00246     swapChainFramebuffers.resize(imageCount());
00247     for (size_t i = 0; i < imageCount(); i++) {
00248         std::array<VkImageView, 2> attachments = {swapChainImageViews[i], depthImageViews[i]};
00249 
00250         const VkExtent2D swapChainExtent = getSwapChainExtent();
00251         VkFramebufferCreateInfo framebufferInfo = {};
00252         framebufferInfo.sType = VK_STRUCTURE_TYPE_FRAMEBUFFER_CREATE_INFO;
00253         framebufferInfo.renderPass = renderPass;
00254         framebufferInfo.attachmentCount = static_cast<uint32_t>(attachments.size());
00255         framebufferInfo.pAttachments = attachments.data();
00256         framebufferInfo.width = swapChainExtent.width;
00257         framebufferInfo.height = swapChainExtent.height;
00258         framebufferInfo.layers = 1;
00259 
00260         if (vkCreateFramebuffer(device.device(), &framebufferInfo, nullptr, &swapChainFramebuffers[i])
      != VK_SUCCESS) {
00261             throw std::runtime_error("failed to create framebuffer!");
00262         }
00263     }
00264 }
00265 
00266 void ven::SwapChain::createDepthResources()
00267 {
```

```
00268     const VkFormat depthFormat = findDepthFormat();
00269     const VkExtent2D swapChainExtent = getSwapChainExtent();
00270 
00271     swapChainDepthFormat = depthFormat;
00272     depthImages.resize(imageCount());
00273     depthImageMemorys.resize(imageCount());
00274     depthImageViews.resize(imageCount());
00275 
00276     for (size_t i = 0; i < depthImages.size(); i++) {
00277         VkImageCreateInfo imageInfo{};
00278         imageInfo.sType = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
00279         imageInfo.imageType = VK_IMAGE_TYPE_2D;
00280         imageInfo.extent.width = swapChainExtent.width;
00281         imageInfo.extent.height = swapChainExtent.height;
00282         imageInfo.extent.depth = 1;
00283         imageInfo.mipLevels = 1;
00284         imageInfo.arrayLayers = 1;
00285         imageInfo.format = depthFormat;
00286         imageInfo.tiling = VK_IMAGE_TILING_OPTIMAL;
00287         imageInfo.initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
00288         imageInfo.usage = VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT;
00289         imageInfo.samples = VK_SAMPLE_COUNT_1_BIT;
00290         imageInfo.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
00291         imageInfo.flags = 0;
00292 
00293         device.createImageWithInfo(imageInfo, VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT, depthImages[i],
      depthImageMemorys[i]);
00294 
00295         VkImageViewCreateInfo viewInfo{};
00296         viewInfo.sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO;
00297         viewInfo.image = depthImages[i];
00298         viewInfo.viewType = VK_IMAGE_VIEW_TYPE_2D;
00299         viewInfo.format = depthFormat;
00300         viewInfo.subresourceRange.aspectMask = VK_IMAGE_ASPECT_DEPTH_BIT;
00301         viewInfo.subresourceRange.baseMipLevel = 0;
00302         viewInfo.subresourceRange.levelCount = 1;
00303         viewInfo.subresourceRange.baseArrayLayer = 0;
00304         viewInfo.subresourceRange.layerCount = 1;
00305 
00306         if (vkCreateImageView(device.device(), &viewInfo, nullptr, &depthImageViews[i]) != VK_SUCCESS)
      {
00307             throw std::runtime_error("failed to create texture image view!");
00308         }
00309     }
00310 }
00311 
00312 void ven::SwapChain::createSyncObjects()
00313 {
00314     imageAvailableSemaphores.resize(MAX_FRAMES_IN_FLIGHT);
00315     renderFinishedSemaphores.resize(MAX_FRAMES_IN_FLIGHT);
00316     inFlightFences.resize(MAX_FRAMES_IN_FLIGHT);
00317     imagesInFlight.resize(imageCount(), VK_NULL_HANDLE);
00318 
00319     VkSemaphoreCreateInfo semaphoreInfo = {};
00320     semaphoreInfo.sType = VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO;
00321 
00322     VkFenceCreateInfo fenceInfo = {};
00323     fenceInfo.sType = VK_STRUCTURE_TYPE_FENCE_CREATE_INFO;
00324     fenceInfo.flags = VK_FENCE_CREATE_SIGNALED_BIT;
00325 
00326     for (size_t i = 0; i < MAX_FRAMES_IN_FLIGHT; i++) {
00327         if (vkCreateSemaphore(device.device(), &semaphoreInfo, nullptr, &imageAvailableSemaphores[i])
      != VK_SUCCESS ||
00328             vkCreateSemaphore(device.device(), &semaphoreInfo, nullptr, &renderFinishedSemaphores[i])
      != VK_SUCCESS ||
00329             vkCreateFence(device.device(), &fenceInfo, nullptr, &inFlightFences[i]) != VK_SUCCESS) {
00330             throw std::runtime_error("failed to create synchronization objects for a frame!");
00331         }
00332     }
00333 }
00334 
00335 VkSurfaceFormatKHR ven::SwapChain::chooseSwapSurfaceFormat(const std::vector<VkSurfaceFormatKHR>
      &availableFormats)
00336 {
00337     for (const auto &availableFormat : availableFormats) {
00338         if (availableFormat.format == VK_FORMAT_B8G8R8A8_UNORM && availableFormat.colorSpace ==
      VK_COLOR_SPACE_SRGB_NONLINEAR_KHR) {
00339             return availableFormat;
00340         }
00341     }
00342 
00343     return availableFormats[0];
00344 }
00345 
00346 VkPresentModeKHR ven::SwapChain::chooseSwapPresentMode(const std::vector<VkPresentModeKHR>
      &availablePresentModes)
00347 {
```

```
00348     for (const auto &availablePresentMode : availablePresentModes) {
00349         if (availablePresentMode == VK_PRESENT_MODE_MAILBOX_KHR) {
00350             std::cout « "Present mode: Mailbox\n";
00351             return availablePresentMode;
00352         }
00353     }
00354
00355     for (const auto &availablePresentMode : availablePresentModes) {
00356       if (availablePresentMode == VK_PRESENT_MODE_IMMEDIATE_KHR) {
00357         std::cout « "Present mode: Immediate" « '\n';
00358         return availablePresentMode;
00359       }
00360     }
00361
00362   std::cout « "Present mode: V-Sync\n";
00363   return VK_PRESENT_MODE_FIFO_KHR;
00364 }
00365
00366 VkExtent2D ven::SwapChain::chooseSwapExtent(const VkSurfaceCapabilitiesKHR &capabilities) const
00367 {
00368     if (capabilities.currentExtent.width != std::numeric_limits<uint32_t>::max()) {
00369         return capabilities.currentExtent;
00370     }
00371     VkExtent2D actualExtent = windowExtent;
00372     actualExtent.width = std::max(capabilities.minImageExtent.width,
      std::min(capabilities.maxImageExtent.width, actualExtent.width));
00373     actualExtent.height = std::max(capabilities.minImageExtent.height,
      std::min(capabilities.maxImageExtent.height, actualExtent.height));
00374
00375     return actualExtent;
00376 }
00377
00378 VkFormat ven::SwapChain::findDepthFormat() const
00379 {
00380     return device.findSupportedFormat(
00381         {VK_FORMAT_D32_SFLOAT, VK_FORMAT_D32_SFLOAT_S8_UINT, VK_FORMAT_D24_UNORM_S8_UINT},
00382         VK_IMAGE_TILING_OPTIMAL,
00383         VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT);
00384 }
```

7.72 /home/runner/work/VEngine/VEngine/src/system/pointLightSystem.cpp File Reference

```
#include <glm/glm.hpp>
#include "VEngine/System/PointLightSystem.hpp"
#include "VEngine/Constant.hpp"
```

Include dependency graph for pointLightSystem.cpp:

Classes

- struct PointLightPushConstants

Macros

- #define GLM_FORCE_RADIANS
- #define GLM_FORCE_DEPTH_ZERO_TO_ONE

7.72.1 Macro Definition Documentation

7.72.1.1 GLM_FORCE_DEPTH_ZERO_TO_ONE

```
#define GLM_FORCE_DEPTH_ZERO_TO_ONE
```

Definition at line 2 of file pointLightSystem.cpp.

7.72.1.2 GLM_FORCE_RADIANS

```
#define GLM_FORCE_RADIANS
```

Definition at line 1 of file pointLightSystem.cpp.

7.73 pointLightSystem.cpp

Go to the documentation of this file.

```
00001 #define GLM_FORCE_RADIANS
00002 #define GLM_FORCE_DEPTH_ZERO_TO_ONE
00003 #include <glm/glm.hpp>
00004
00005 #include "VEngine/System/PointLightSystem.hpp"
00006 #include "VEngine/Constant.hpp"
00007
00008
00009 struct PointLightPushConstants {
00010     glm::vec4 position{};
00011     glm::vec4 color{};
00012     float radius;
00013 };
00014
00015 ven::PointLightSystem::PointLightSystem(Device& device, const VkRenderPass renderPass,const
      VkDescriptorSetLayout globalSetLayout) : m_device{device}
00016 {
00017     createPipelineLayout(globalSetLayout);
00018     createPipeline(renderPass);
00019 }
00020
00021 void ven::PointLightSystem::createPipelineLayout(const VkDescriptorSetLayout globalSetLayout)
00022 {
00023     VkPushConstantRange pushConstantRange{};
00024     pushConstantRange.stageFlags = VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT;
00025     pushConstantRange.offset = 0;
00026     pushConstantRange.size = sizeof(PointLightPushConstants);
00027
00028     const std::vector<VkDescriptorSetLayout> descriptorSetLayouts{globalSetLayout};
00029
00030     VkPipelineLayoutCreateInfo pipelineLayoutInfo{};
00031     pipelineLayoutInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_LAYOUT_CREATE_INFO;
00032     pipelineLayoutInfo.setLayoutCount = static_cast<uint32_t>(descriptorSetLayouts.size());
00033     pipelineLayoutInfo.pSetLayouts = descriptorSetLayouts.data();
00034     pipelineLayoutInfo.pushConstantRangeCount = 1;
00035     pipelineLayoutInfo.pPushConstantRanges = &pushConstantRange;
00036     if (vkCreatePipelineLayout(m_device.device(), &pipelineLayoutInfo, nullptr, &m_pipelineLayout) !=
      VK_SUCCESS)
00037     {
00038         throw std::runtime_error("Failed to create pipeline layout");
00039     }
00040 }
00041
00042 void ven::PointLightSystem::createPipeline(const VkRenderPass renderPass)
00043 {
00044     PipelineConfigInfo pipelineConfig{};
00045     Shaders::defaultPipelineConfigInfo(pipelineConfig);
00046     pipelineConfig.attributeDescriptions.clear();
00047     pipelineConfig.bindingDescriptions.clear();
00048     pipelineConfig.renderPass = renderPass;
00049     pipelineConfig.pipelineLayout = m_pipelineLayout;
00050     m_shaders = std::make_unique<Shaders>(m_device, std::string(SHADERS_BIN_PATH) +
      "point_light_vert.spv", std::string(SHADERS_BIN_PATH) + "point_light_frag.spv", pipelineConfig);
00051 }
00052
00053 void ven::PointLightSystem::render(const FrameInfo &frameInfo) const
00054 {
00055     m_shaders->bind(frameInfo.commandBuffer);
00056
00057     vkCmdBindDescriptorSets(frameInfo.commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS,
      m_pipelineLayout, 0, 1, &frameInfo.globalDescriptorSet, 0, nullptr);
00058
00059     for (auto &kv : frameInfo.objects)
00060     {
00061         Object &object = kv.second;
00062         if (object.pointLight == nullptr) continue;
00063         PointLightPushConstants push{};
00064         push.position = glm::vec4(object.transform3D.translation, 1.F);
00065         push.color = glm::vec4(object.color, object.pointLight->lightIntensity);
```

```
00066         push.radius = object.transform3D.scale.x;
00067         vkCmdPushConstants(frameInfo.commandBuffer, m_pipelineLayout, VK_SHADER_STAGE_VERTEX_BIT |
      VK_SHADER_STAGE_FRAGMENT_BIT, 0, sizeof(PointLightPushConstants), &push);
00068         vkCmdDraw(frameInfo.commandBuffer, 6, 1, 0, 0);
00069     }
00070
00071 }
00072
00073 void ven::PointLightSystem::update(const FrameInfo &frameInfo, GlobalUbo &ubo)
00074 {
00075     const auto rotateLight = glm::rotate(glm::mat4(1.F), frameInfo.frameTime, {0.F, -1.F, 0.F});
00076     unsigned long lightIndex = 0;
00077     for (auto &kv : frameInfo.objects)
00078     {
00079         Object &object = kv.second;
00080         if (object.pointLight == nullptr) continue;
00081         assert(lightIndex < MAX_LIGHTS && "Too many lights");
00082         object.transform3D.translation = glm::vec3(rotateLight *
      glm::vec4(object.transform3D.translation, 1.F));
00083         ubo.pointLights[lightIndex].position = glm::vec4(object.transform3D.translation, 1.F);
00084         ubo.pointLights[lightIndex].color = glm::vec4(object.color,
      object.pointLight->lightIntensity);
00085         lightIndex++;
00086     }
00087     ubo.numLights = static_cast<int>(lightIndex);
00088 }
```

7.74 /home/runner/work/VEngine/VEngine/src/system/renderSystem.cpp File Reference

```
#include "VEngine/System/RenderSystem.hpp"
#include "VEngine/Constant.hpp"
```

Include dependency graph for renderSystem.cpp:

7.75 renderSystem.cpp

Go to the documentation of this file.

```
00001 #include "VEngine/System/RenderSystem.hpp"
00002 #include "VEngine/Constant.hpp"
00003
00004
00005 ven::RenderSystem::RenderSystem(Device& device, const VkRenderPass renderPass,const
      VkDescriptorSetLayout globalSetLayout) : m_device{device}
00006 {
00007     createPipelineLayout(globalSetLayout);
00008     createPipeline(renderPass);
00009 }
00010
00011 void ven::RenderSystem::createPipelineLayout(const VkDescriptorSetLayout globalSetLayout)
00012 {
00013     VkPushConstantRange pushConstantRange{};
00014     pushConstantRange.stageFlags = VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT;
00015     pushConstantRange.offset = 0;
00016     pushConstantRange.size = sizeof(SimplePushConstantData);
00017
00018     const std::vector<VkDescriptorSetLayout> descriptorSetLayouts{globalSetLayout};
00019
00020     VkPipelineLayoutCreateInfo pipelineLayoutInfo{};
00021     pipelineLayoutInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_LAYOUT_CREATE_INFO;
00022     pipelineLayoutInfo.setLayoutCount = static_cast<uint32_t>(descriptorSetLayouts.size());
00023     pipelineLayoutInfo.pSetLayouts = descriptorSetLayouts.data();
00024     pipelineLayoutInfo.pushConstantRangeCount = 1;
00025     pipelineLayoutInfo.pPushConstantRanges = &pushConstantRange;
00026     if (vkCreatePipelineLayout(m_device.device(), &pipelineLayoutInfo, nullptr, &m_pipelineLayout) !=
      VK_SUCCESS)
00027     {
00028         throw std::runtime_error("Failed to create pipeline layout");
00029     }
00030 }
00031
00032 void ven::RenderSystem::createPipeline(const VkRenderPass renderPass)
00033 {
00034     PipelineConfigInfo pipelineConfig{};
```

```
00035     Shaders::defaultPipelineConfigInfo(pipelineConfig);
00036     pipelineConfig.renderPass = renderPass;
00037     pipelineConfig.pipelineLayout = m_pipelineLayout;
00038     m_shaders = std::make_unique<Shaders>(m_device, std::string(SHADERS_BIN_PATH) + "shader_vert.spv",
      std::string(SHADERS_BIN_PATH) + "shader_frag.spv", pipelineConfig);
00039 }
00040
00041 void ven::RenderSystem::renderObjects(const FrameInfo &frameInfo) const
00042 {
00043     m_shaders->bind(frameInfo.commandBuffer);
00044
00045     vkCmdBindDescriptorSets(frameInfo.commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS,
      m_pipelineLayout, 0, 1, &frameInfo.globalDescriptorSet, 0, nullptr);
00046
00047     for (auto &kv : frameInfo.objects)
00048     {
00049         Object &object = kv.second;
00050         if (object.model == nullptr) continue;
00051         SimplePushConstantData push{};
00052         push.modelMatrix = object.transform3D.mat4();
00053         push.normalMatrix = object.transform3D.normalMatrix();
00054         vkCmdPushConstants(frameInfo.commandBuffer, m_pipelineLayout, VK_SHADER_STAGE_VERTEX_BIT |
      VK_SHADER_STAGE_FRAGMENT_BIT, 0, sizeof(SimplePushConstantData), &push);
00055         object.model->bind(frameInfo.commandBuffer);
00056         object.model->draw(frameInfo.commandBuffer);
00057     }
00058 }
```

7.76 /home/runner/work/VEngine/VEngine/src/window.cpp File Reference

```
#include <stdexcept>
#include "VEngine/Window.hpp"
```

Include dependency graph for window.cpp:

7.77 window.cpp

Go to the documentation of this file.

```
00001 #include <stdexcept>
00002
00003 #include "VEngine/Window.hpp"
00004
00005 GLFWwindow* ven::Window::createWindow(const uint32_t width, const uint32_t height, const std::string
      &title)
00006 {
00007     if (glfwInit() == GLFW_FALSE) {
00008         throw std::runtime_error("Failed to initialize GLFW");
00009     }
00010
00011     glfwWindowHint(GLFW_CLIENT_API, GLFW_NO_API);
00012     glfwWindowHint(GLFW_RESIZABLE, GLFW_TRUE);
00013
00014     GLFWwindow *window = glfwCreateWindow(static_cast<int>(width), static_cast<int>(height),
      title.c_str(), nullptr, nullptr);
00015     if (window == nullptr) {
00016         glfwTerminate();
00017         throw std::runtime_error("Failed to create window");
00018     }
00019     glfwSetWindowUserPointer(window, this);
00020     glfwSetFramebufferSizeCallback(window, framebufferResizeCallback);
00021     return window;
00022 }
00023
00024 void ven::Window::createWindowSurface(const VkInstance instance, VkSurfaceKHR *surface) const
00025 {
00026     if (glfwCreateWindowSurface(instance, m_window, nullptr, surface) != VK_SUCCESS) {
00027         throw std::runtime_error("Failed to create window surface");
00028     }
00029 }
00030
00031 void ven::Window::framebufferResizeCallback(GLFWwindow *window, const int width, const int height)
00032 {
00033     auto *app = static_cast<Window *>(glfwGetWindowUserPointer(window));
00034     app->m_framebufferResized = true;
00035     app->m_width = static_cast<uint32_t>(width);
00036     app->m_height = static_cast<uint32_t>(height);
00037 }
```

Index