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 Key Bindings

Key	Description
z	Move forward
S	Move backward
q	Move left
D	Move right
SHIFT	Move down
SPACE	Move up
arrow up	Look up
arrow down	Look down
arrow left	Look left
arrow right	Look right
F1	Show debug windows

1.3.4 Documentation

The documentation is generated using `Doxygen`. You can visualize it on the `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

This project is licensed under the MIT License - see the `LICENSE` file for details.

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.

Variables

- static constexpr unsigned int MICROSECONDS_PER_SECOND = 1000000
- static constexpr unsigned int MILLISECONDS_PER_SECOND = 1000
- static constexpr int RANDOM_INT_MIN = -1000
- static constexpr int RANDOM_INT_MAX = 1000
- static constexpr float RANDOM_FLOAT_MAX = 1000.0F

5.1.1 Variable Documentation

5.1.1.1 MICROSECONDS_PER_SECOND

```
unsigned int myLib::MICROSECONDS_PER_SECOND = 1000000  [static], [constexpr]
```

Definition at line 11 of file Time.hpp.

Referenced by myLib::Time::asMicroseconds().

5.1.1.2 MILLISECONDS_PER_SECOND

```
unsigned int myLib::MILLISECONDS_PER_SECOND = 1000  [static], [constexpr]
```

Definition at line 12 of file Time.hpp.

Referenced by myLib::Time::asMilliseconds().

5.1.1.3 RANDOM_FLOAT_MAX

```
float myLib::RANDOM_FLOAT_MAX = 1000.0F  [static], [constexpr]
```

Definition at line 15 of file Random.hpp.

Referenced by myLib::Random::randomFloat().

5.1.1.4 RANDOM_INT_MAX

```
int myLib::RANDOM_INT_MAX = 1000  [static], [constexpr]
```

Definition at line 14 of file Random.hpp.

Referenced by myLib::Random::randomFloat().

5.1.1.5 RANDOM_INT_MIN

```
int myLib::RANDOM_INT_MIN = -1000  [static], [constexpr]
```

Definition at line 13 of file Random.hpp.

Referenced by myLib::Random::randomFloat().

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 for camera.
- class Colors
  Class for colors.
- class DescriptorPool
  Class for descriptor pool.
- class DescriptorSetLayout
  Class for descriptor set layout.
- class DescriptorWriter

Class for descriptor writer.
- class Device
- class Engine
- struct FrameInfo
- struct GlobalUbo
- class ImGuiWindowManager

  Class for ImGui window manager.
- class KeyboardController

  Class for keyboard controller.
- class Model

  Class for model.
- class Object

  Class for 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

  Class for shaders.
- struct SimplePushConstantData
- class SwapChain

  Class for swap chain.
- struct SwapChainSupportDetails
- struct Transform3DComponent
- class Window

  Class for window.

Functions

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

Variables

- static constexpr glm::vec3 DEFAULT_POSITION {0.F, 0.F, -2.5F}
- static constexpr glm::vec3 DEFAULT_ROTATION {0.F, 0.F, 0.F}
- static constexpr float DEFAULT_FOV = glm::radians(50.0F)
- static constexpr float DEFAULT_NEAR = 0.1F
- static constexpr float DEFAULT_FAR = 100.F
- static constexpr std::size_t MAX_LIGHTS = 10
- static constexpr float DEFAULT_MOVE_SPEED = 3.F
- static constexpr float DEFAULT_LOOK_SPEED = 1.5F
- static constexpr float DEFAULT_LIGHT_INTENSITY = .2F
- static constexpr float DEFAULT_LIGHT_RADIUS = 0.1F
- static constexpr glm::vec3 DEFAULT_LIGHT_COLOR = glm::vec3(1.F)
- static constexpr VkClearColorValue DEFAULT_CLEAR_COLOR = {{0.0F, 0.0F, 0.0F, 1.0F}}
- static constexpr VkClearDepthStencilValue DEFAULT_CLEAR_DEPTH = {1.0F, 0}
- static constexpr std::string_view SHADERS_BIN_PATH = "shaders/bin/"
- static constexpr uint32_t DEFAULT_WIDTH = 1920
- static constexpr uint32_t DEFAULT_HEIGHT = 1080
- static constexpr std::string_view DEFAULT_TITLE = "VEngine"

5.3.1 Function Documentation

5.3.1.1 hashCombine()

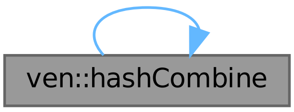
```
template<typename T , typename... Rest>
void ven::hashCombine (
            std::size_t & seed,
            const T & v,
            const Rest &... rest)
```

Definition at line 14 of file Utils.hpp.

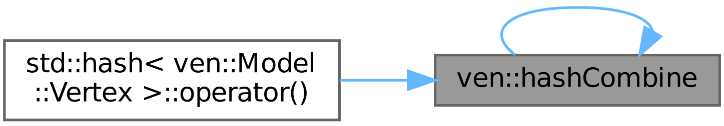
References hashCombine().

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:



5.3.2 Variable Documentation

5.3.2.1 DEFAULT_CLEAR_COLOR

```
VkClearColorValue ven::DEFAULT_CLEAR_COLOR = {{0.0F, 0.0F, 0.0F, 1.0F}} [static], [constexpr]
```

Definition at line 20 of file Renderer.hpp.

5.3.2.2 DEFAULT_CLEAR_DEPTH

```
VkClearDepthStencilValue ven::DEFAULT_CLEAR_DEPTH = {1.0F, 0} [static], [constexpr]
```

Definition at line 21 of file Renderer.hpp.

5.3.2.3 DEFAULT_FAR

```
float ven::DEFAULT_FAR = 100.F [static], [constexpr]
```

Definition at line 18 of file Camera.hpp.

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

5.3.2.4 DEFAULT_FOV

```
float ven::DEFAULT_FOV = glm::radians(50.0F) [static], [constexpr]
```

Definition at line 16 of file Camera.hpp.

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

5.3.2.5 DEFAULT_HEIGHT

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

Definition at line 18 of file Window.hpp.

5.3.2.6 DEFAULT_LIGHT_COLOR

```
glm::vec3 ven::DEFAULT_LIGHT_COLOR = glm::vec3(1.F) [static], [constexpr]
```

Definition at line 20 of file Object.hpp.

5.3.2.7 DEFAULT_LIGHT_INTENSITY

```
float ven::DEFAULT_LIGHT_INTENSITY = .2F [static], [constexpr]
```

Definition at line 18 of file Object.hpp.

5.3.2.8 DEFAULT_LIGHT_RADIUS

```
float ven::DEFAULT_LIGHT_RADIUS = 0.1F [static], [constexpr]
```

Definition at line 19 of file Object.hpp.

5.3.2.9 DEFAULT_LOOK_SPEED

```
float ven::DEFAULT_LOOK_SPEED = 1.5F  [static], [constexpr]
```

Definition at line 15 of file KeyboardController.hpp.

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

5.3.2.10 DEFAULT_MOVE_SPEED

```
float ven::DEFAULT_MOVE_SPEED = 3.F  [static], [constexpr]
```

Definition at line 14 of file KeyboardController.hpp.

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

5.3.2.11 DEFAULT_NEAR

```
float ven::DEFAULT_NEAR = 0.1F  [static], [constexpr]
```

Definition at line 17 of file Camera.hpp.

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

5.3.2.12 DEFAULT_POSITION

```
glm::vec3 ven::DEFAULT_POSITION {0.F, 0.F, -2.5F}  [static], [constexpr]
```

Definition at line 13 of file Camera.hpp.

Referenced by ven::Engine::mainLoop(), and ven::ImGuiWindowManager::render().

5.3.2.13 DEFAULT_ROTATION

```
glm::vec3 ven::DEFAULT_ROTATION {0.F, 0.F, 0.F}  [static], [constexpr]
```

Definition at line 14 of file Camera.hpp.

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

5.3.2.14 DEFAULT_TITLE

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

Definition at line 19 of file Window.hpp.

5.3.2.15 DEFAULT_WIDTH

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

Definition at line 17 of file Window.hpp.

5.3.2.16 MAX_LIGHTS

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

Definition at line 16 of file FrameInfo.hpp.

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

5.3.2.17 SHADERS_BIN_PATH

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

Definition at line 19 of file Shaders.hpp.

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

Chapter 6

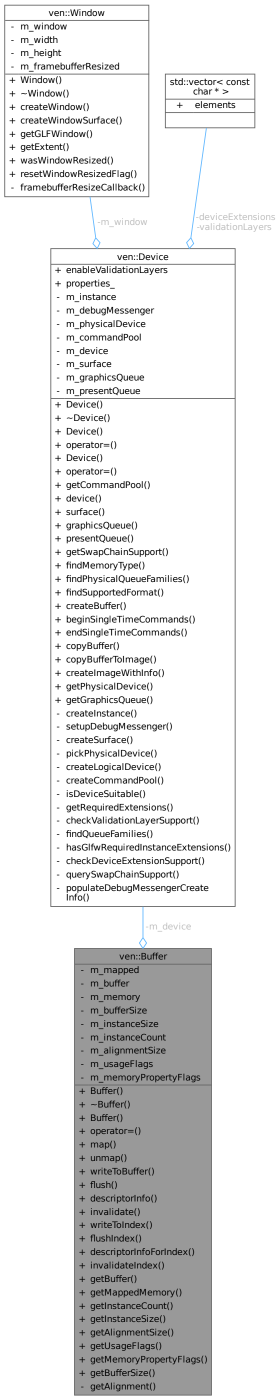
Class Documentation

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 usageFlags, 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 18 of file Buffer.hpp.

6.1.2 Constructor & Destructor Documentation

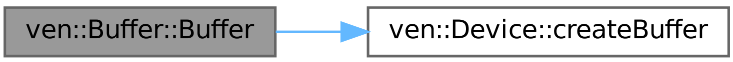
6.1.2.1 Buffer() [1/2]

```
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

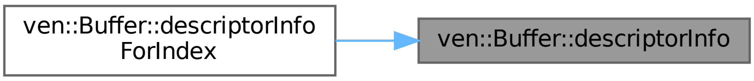
VkDescriptorBufferInfo of specified offset and range

Definition at line 74 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

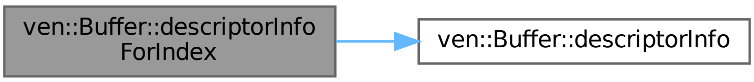
Returns

VkDescriptorBufferInfo for instance at index

Definition at line 113 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

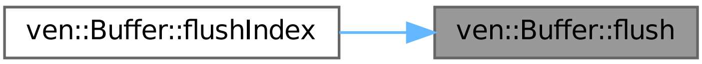
Returns

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.

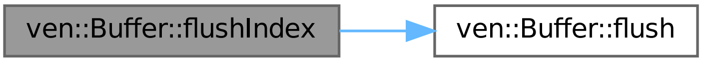
Parameters

index	Used in offset calculation

Definition at line 103 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 130 of file Buffer.hpp.

References m_instanceSize.

6.1.3.7 getBuffer()

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

Definition at line 126 of file Buffer.hpp.

References m_buffer.

6.1.3.8 getBufferSize()

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

Definition at line 133 of file Buffer.hpp.

References m_bufferSize.

6.1.3.9 getInstanceCount()

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

Definition at line 128 of file Buffer.hpp.

References m_instanceCount.

6.1.3.10 getInstanceSize()

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

Definition at line 129 of file Buffer.hpp.

References m_instanceSize.

6.1.3.11 getMappedMemory()

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

Definition at line 127 of file Buffer.hpp.

References m_mapped.

6.1.3.12 getMemoryPropertyFlags()

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

Definition at line 132 of file Buffer.hpp.

References m_memoryPropertyFlags.

6.1.3.13 getUsageFlags()

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

Definition at line 131 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

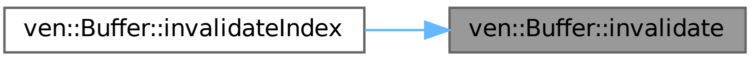
Returns

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

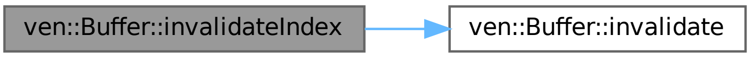
Returns

VkResult of the invalidate call

Definition at line 124 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

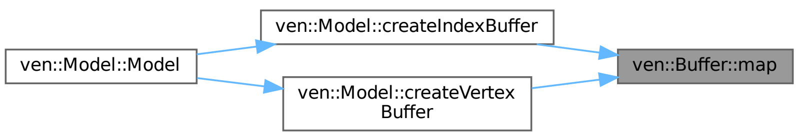
Returns

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

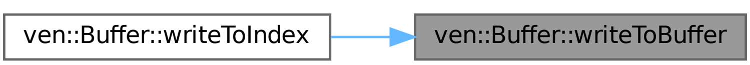
Parameters

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.

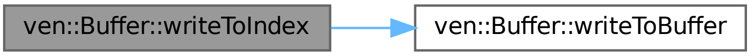
Parameters

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

Definition at line 96 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 155 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 149 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 152 of file Buffer.hpp.

Referenced by Buffer(), and getBufferSize().

6.1.4.4 m_device

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

Definition at line 147 of file Buffer.hpp.

6.1.4.5 m_instanceCount

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

Definition at line 154 of file Buffer.hpp.

Referenced by Buffer(), and getInstanceCount().

6.1.4.6 m_instanceSize

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

Definition at line 153 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 148 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 150 of file Buffer.hpp.

Referenced by Buffer().

6.1.4.9 m_memoryPropertyFlags

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

Definition at line 157 of file Buffer.hpp.

Referenced by Buffer(), and getMemoryPropertyFlags().

6.1.4.10 m_usageFlags

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

Definition at line 156 of file Buffer.hpp.

Referenced by Buffer(), and getUsageFlags().

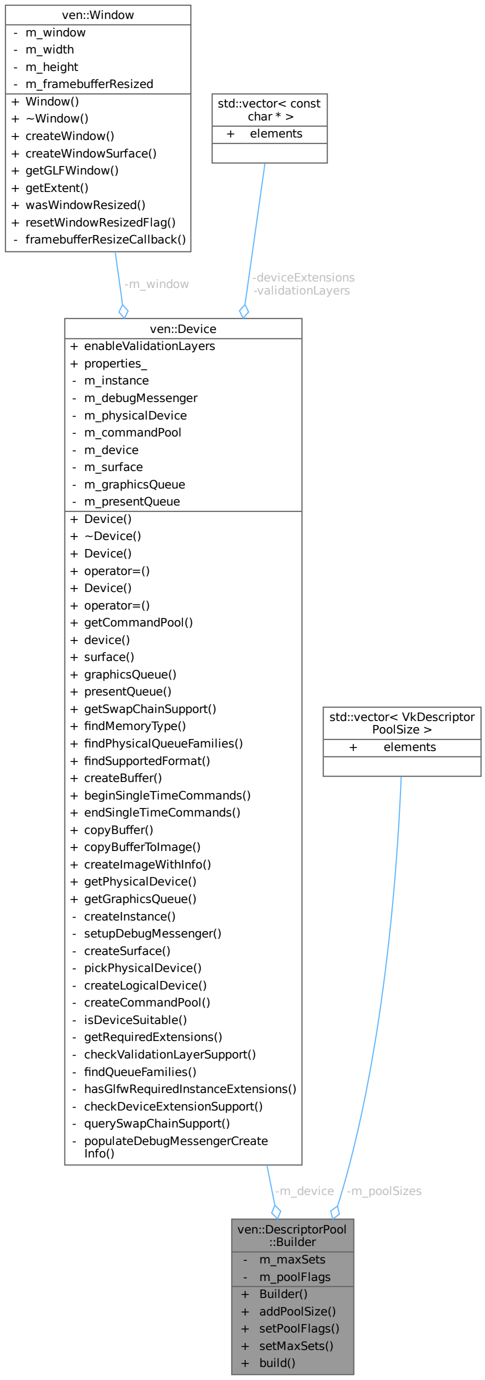
The documentation for this class was generated from the following files:

- /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 <DescriptorPool.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 25 of file DescriptorPool.hpp.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 Builder()

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

Definition at line 29 of file DescriptorPool.hpp.

6.2.3 Member Function Documentation

6.2.3.1 addPoolSize()

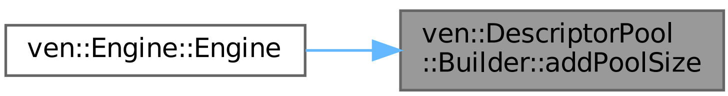
```
ven::DescriptorPool::Builder & ven::DescriptorPool::Builder::addPoolSize (
            VkDescriptorType descriptorType,
            uint32_t count)
```

Definition at line 3 of file descriptorPool.cpp.

References m_poolSizes.

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

Here is the caller graph for this function:



6.2.3.2 build()

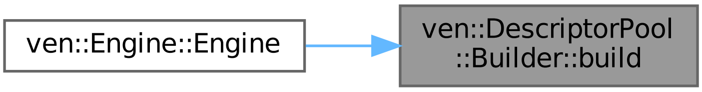
```
std::unique_ptr< DescriptorPool > ven::DescriptorPool::Builder::build () const  [inline],
[nodiscard]
```

Definition at line 34 of file DescriptorPool.hpp.

References m_device, m_maxSets, m_poolFlags, and m_poolSizes.

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

Here is the caller graph for this function:



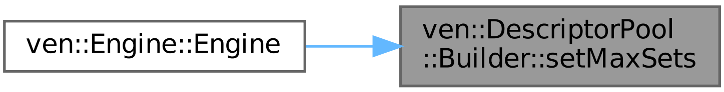
6.2.3.3 setMaxSets()

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

Definition at line 14 of file descriptorPool.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 9 of file descriptorPool.cpp.

6.2.4 Member Data Documentation

6.2.4.1 m_device

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

Definition at line 38 of file DescriptorPool.hpp.

Referenced by build().

6.2.4.2 m_maxSets

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

Definition at line 40 of file DescriptorPool.hpp.

Referenced by build().

6.2.4.3 m_poolFlags

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

Definition at line 41 of file DescriptorPool.hpp.

Referenced by build().

6.2.4.4 m_poolSizes

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

Definition at line 39 of file DescriptorPool.hpp.

Referenced by addPoolSize(), and build().

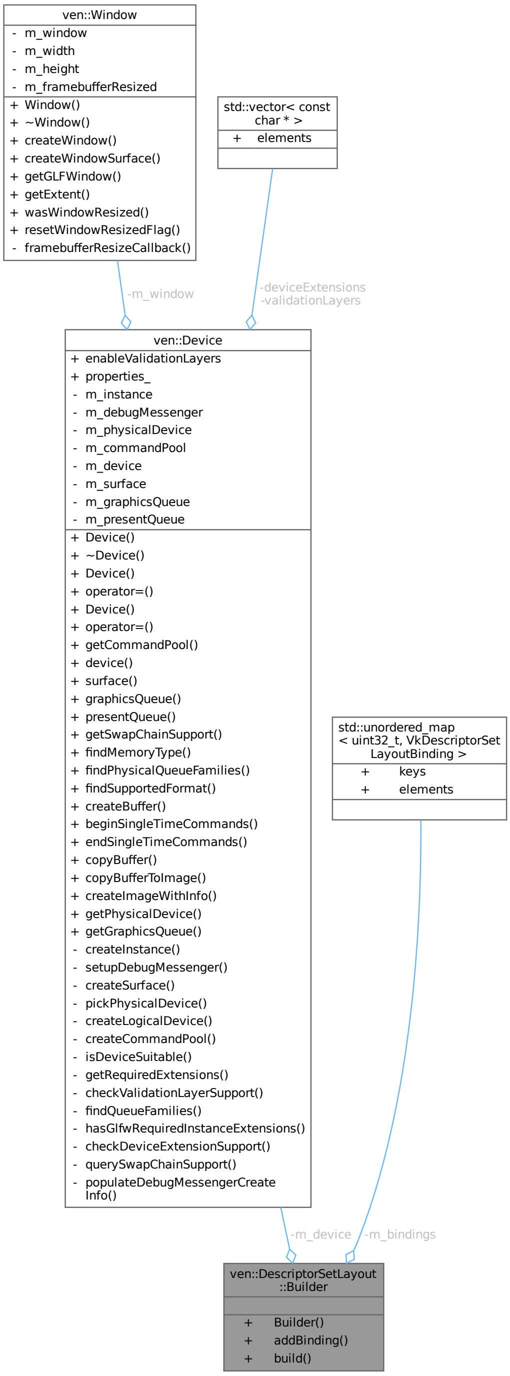
The documentation for this class was generated from the following files:

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

6.3 ven::DescriptorSetLayout::Builder Class Reference

`#include <DescriptorSetLayout.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 DescriptorSetLayout.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 DescriptorSetLayout.hpp.

6.3.3 Member Function Documentation

6.3.3.1 addBinding()

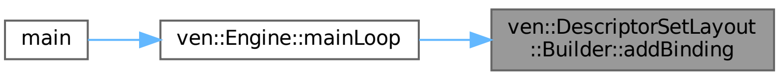
```
ven::DescriptorSetLayout::Builder & ven::DescriptorSetLayout::Builder::addBinding (
            uint32_t binding,
            VkDescriptorType descriptorType,
            VkShaderStageFlags stageFlags,
            uint32_t count = 1)
```

Definition at line 5 of file descriptorSetLayout.cpp.

References m_bindings.

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

Here is the caller graph for this function:



6.3.3.2 build()

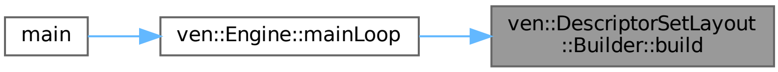
```
std::unique_ptr< DescriptorSetLayout > ven::DescriptorSetLayout::Builder::build () const
[inline]
```

Definition at line 32 of file DescriptorSetLayout.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 37 of file DescriptorSetLayout.hpp.

Referenced by addBinding(), and build().

6.3.4.2 m_device

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

Definition at line 36 of file DescriptorSetLayout.hpp.

Referenced by build().

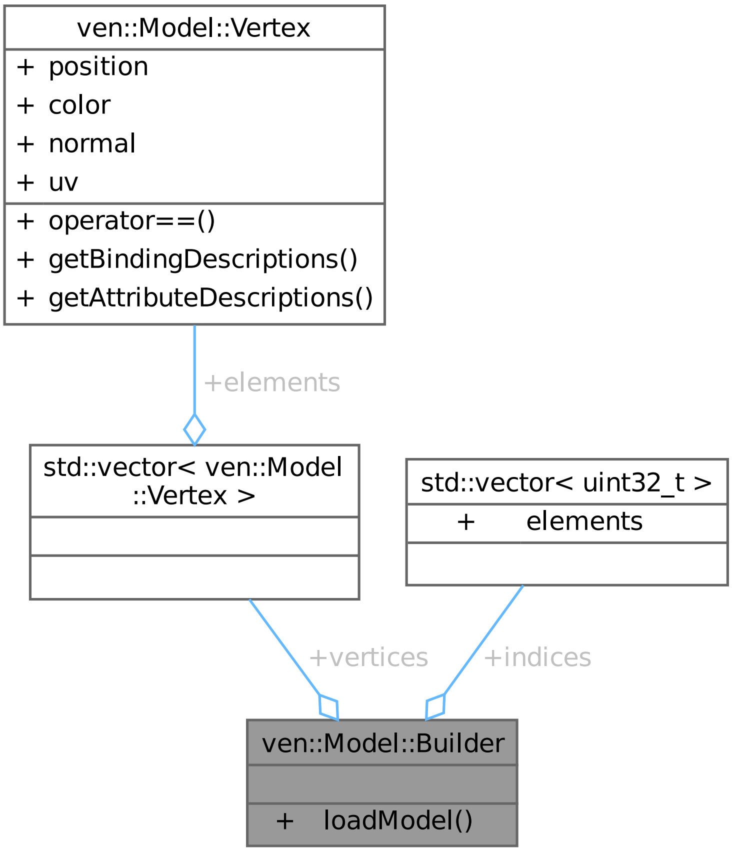
The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorSetLayout.hpp
- /home/runner/work/VEngine/VEngine/src/descriptors/descriptorSetLayout.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 39 of file Model.hpp.

6.4.2 Member Function Documentation

6.4.2.1 loadModel()

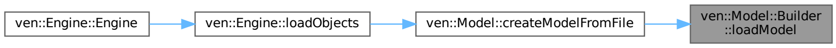
```
void ven::Model::Builder::loadModel (
            const std::string & filename)
```

Definition at line 119 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 41 of file Model.hpp.

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

6.4.3.2 vertices

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

Definition at line 40 of file Model.hpp.

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

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

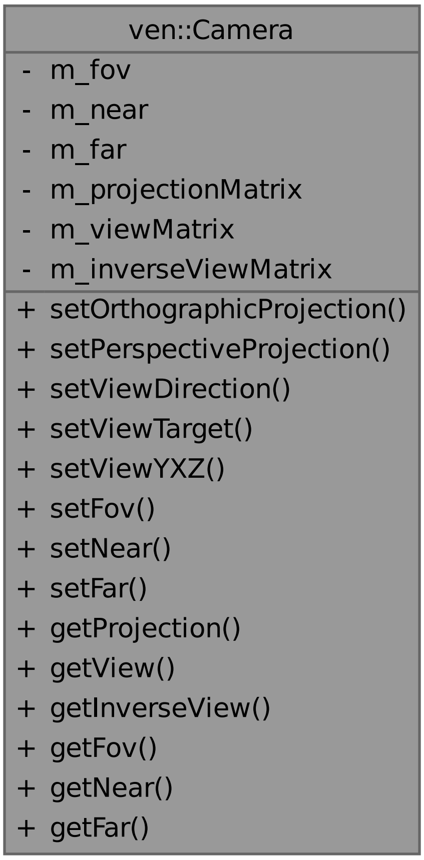
- /home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp
- /home/runner/work/VEngine/VEngine/src/model.cpp

6.5 ven::Camera Class Reference

Class for camera.

`#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 aspect)
- 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)
- void setFov (float fov)
- void setNear (float near)
- void setFar (float far)
- const glm::mat4 & getProjection () const
- const glm::mat4 & getView () const
- const glm::mat4 & getInverseView () const
- float getFov () const
- float getNear () const
- float getFar () const

Private Attributes

- float m_fov {DEFAULT_FOV}
- float m_near {DEFAULT_NEAR}
- float m_far {DEFAULT_FAR}
- glm::mat4 m_projectionMatrix {1.F}
- glm::mat4 m_viewMatrix {1.F}
- glm::mat4 m_inverseViewMatrix {1.F}

6.5.1 Detailed Description

Class for camera.

Definition at line 25 of file Camera.hpp.

6.5.2 Member Function Documentation

6.5.2.1 getFar()

```
float ven::Camera::getFar () const  [inline], [nodiscard]
```

Definition at line 43 of file Camera.hpp.

References m_far.

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

Here is the caller graph for this function:

6.5.2.2 getFov()

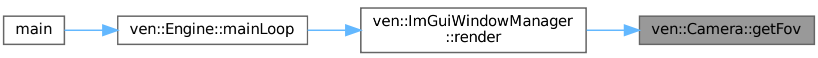
```
float ven::Camera::getFov () const  [inline], [nodiscard]
```

Definition at line 41 of file Camera.hpp.

References m_fov.

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

Here is the caller graph for this function:



6.5.2.3 getInverseView()

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

Definition at line 40 of file Camera.hpp.

References m_inverseViewMatrix.

6.5.2.4 getNear()

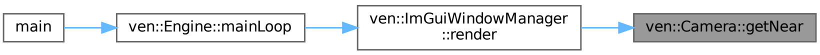
`float ven::Camera::getNear () const [inline], [nodiscard]`

Definition at line 42 of file Camera.hpp.

References m_near.

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

Here is the caller graph for this function:



6.5.2.5 getProjection()

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

Definition at line 38 of file Camera.hpp.

References m_projectionMatrix.

6.5.2.6 getView()

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

Definition at line 39 of file Camera.hpp.

References m_viewMatrix.

6.5.2.7 setFar()

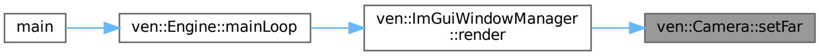
```
void ven::Camera::setFar (
            float far)  [inline]
```

Definition at line 36 of file Camera.hpp.

References m_far.

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

Here is the caller graph for this function:



6.5.2.8 setFov()

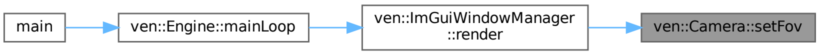
```
void ven::Camera::setFov (
            float fov)  [inline]
```

Definition at line 34 of file Camera.hpp.

References m_fov.

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

Here is the caller graph for this function:



6.5.2.9 setNear()

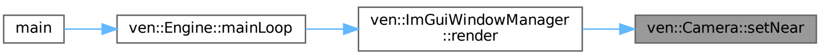
```
void ven::Camera::setNear (
            float near)  [inline]
```

Definition at line 35 of file Camera.hpp.

References m_near.

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

Here is the caller graph for this function:



6.5.2.10 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.

6.5.2.11 setPerspectiveProjection()

```
void ven::Camera::setPerspectiveProjection (
            float aspect)
```

Definition at line 17 of file camera.cpp.

6.5.2.12 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.

6.5.2.13 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 32 of file Camera.hpp.

6.5.2.14 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_far

```
float ven::Camera::m_far {DEFAULT_FAR} [private]
```

Definition at line 49 of file Camera.hpp.

Referenced by getFar(), and setFar().

6.5.3.2 m_fov

```
float ven::Camera::m_fov {DEFAULT_FOV} [private]
```

Definition at line 47 of file Camera.hpp.

Referenced by getFov(), and setFov().

6.5.3.3 m_inverseViewMatrix

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

Definition at line 52 of file Camera.hpp.

Referenced by getInverseView().

6.5.3.4 m_near

```
float ven::Camera::m_near {DEFAULT_NEAR} [private]
```

Definition at line 48 of file Camera.hpp.

Referenced by getNear(), and setNear().

6.5.3.5 m_projectionMatrix

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

Definition at line 50 of file Camera.hpp.

Referenced by getProjection(), and setOrthographicProjection().

6.5.3.6 m_viewMatrix

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

Definition at line 51 of file Camera.hpp.

Referenced by getView().

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

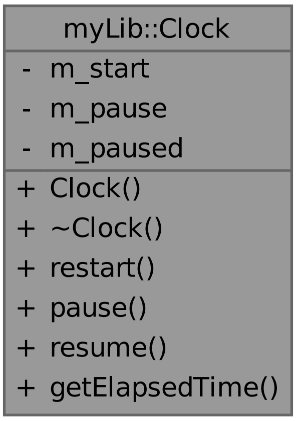
- /home/runner/work/VEngine/VEngine/include/VEngine/Camera.hpp
- /home/runner/work/VEngine/VEngine/src/camera.cpp

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:

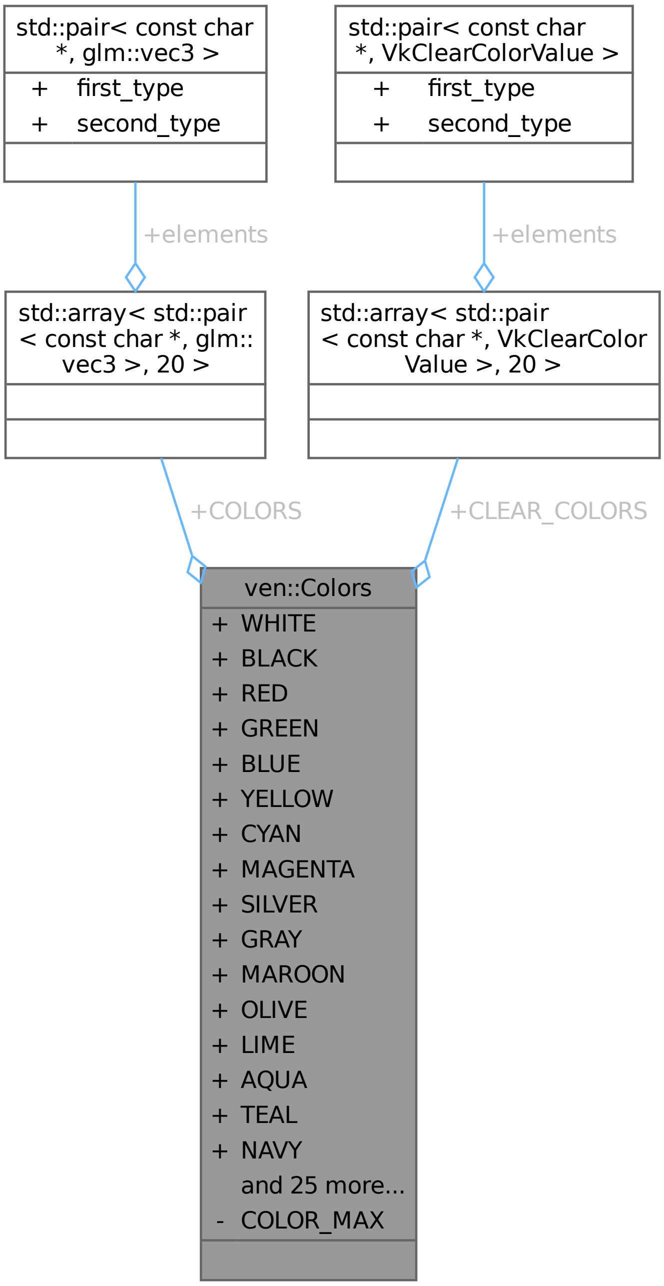
- /home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/Clock/Clock.hpp
- /home/runner/work/VEngine/VEngine/lib/local/static/myLib/src/clock.cpp

6.7 ven::Colors Class Reference

Class for colors.

`#include <Colors.hpp>`

Collaboration diagram for ven::Colors:



Static Public Attributes

- static constexpr glm::vec3 WHITE = glm::vec3(COLOR_MAX, COLOR_MAX, COLOR_MAX) / COLOR_MAX
- static constexpr glm::vec3 BLACK = glm::vec3(0.0F)
- static constexpr glm::vec3 RED = glm::vec3(COLOR_MAX, 0.0F, 0.0F) / COLOR_MAX
- static constexpr glm::vec3 GREEN = glm::vec3(0.0F, COLOR_MAX, 0.0F) / COLOR_MAX
- static constexpr glm::vec3 BLUE = glm::vec3(0.0F, 0.0F, COLOR_MAX) / COLOR_MAX
- static constexpr glm::vec3 YELLOW = glm::vec3(COLOR_MAX, COLOR_MAX, 0.0F) / COLOR_MAX
- static constexpr glm::vec3 CYAN = glm::vec3(0.0F, COLOR_MAX, COLOR_MAX) / COLOR_MAX
- static constexpr glm::vec3 MAGENTA = glm::vec3(COLOR_MAX, 0.0F, COLOR_MAX) / COLOR_MAX
- static constexpr glm::vec3 SILVER = glm::vec3(192.0F, 192.0F, 192.0F) / COLOR_MAX
- static constexpr glm::vec3 GRAY = glm::vec3(128.0F, 128.0F, 128.0F) / COLOR_MAX
- static constexpr glm::vec3 MAROON = glm::vec3(128.0F, 0.0F, 0.0F) / COLOR_MAX
- static constexpr glm::vec3 OLIVE = glm::vec3(128.0F, 128.0F, 0.0F) / COLOR_MAX
- static constexpr glm::vec3 LIME = glm::vec3(0.0F, COLOR_MAX, 0.0F) / COLOR_MAX
- static constexpr glm::vec3 AQUA = glm::vec3(0.0F, COLOR_MAX, COLOR_MAX) / COLOR_MAX
- static constexpr glm::vec3 TEAL = glm::vec3(0.0F, 128.0F, 128.0F) / COLOR_MAX
- static constexpr glm::vec3 NAVY = glm::vec3(0.0F, 0.0F, 128.0F) / COLOR_MAX
- static constexpr glm::vec3 FUCHSIA = glm::vec3(COLOR_MAX, 0.0F, COLOR_MAX) / COLOR_MAX
- static constexpr glm::vec3 NIGHT_BLUE = glm::vec3(25.0F, 25.0F, 112.0F) / COLOR_MAX
- static constexpr glm::vec3 SKY_BLUE = glm::vec3(102.0F, 178.0F, 255.0F) / COLOR_MAX
- static constexpr glm::vec3 SUNSET = glm::vec3(255.0F, 128.0F, 0.0F) / COLOR_MAX
- static constexpr VkClearColorValue WHITE_V = {{1.0F, 1.0F, 1.0F, 1.0F}}
- static constexpr VkClearColorValue BLACK_V = {{0.0F, 0.0F, 0.0F, 1.0F}}
- static constexpr VkClearColorValue RED_V = {{1.0F, 0.0F, 0.0F, 1.0F}}
- static constexpr VkClearColorValue GREEN_V = {{0.0F, 1.0F, 0.0F, 1.0F}}
- static constexpr VkClearColorValue BLUE_V = {{0.0F, 0.0F, 1.0F, 1.0F}}
- static constexpr VkClearColorValue YELLOW_V = {{1.0F, 1.0F, 0.0F, 1.0F}}
- static constexpr VkClearColorValue CYAN_V = {{0.0F, 1.0F, 1.0F, 1.0F}}
- static constexpr VkClearColorValue MAGENTA_V = {{1.0F, 0.0F, 1.0F, 1.0F}}
- static constexpr VkClearColorValue SILVER_V = {{0.75F, 0.75F, 0.75F, 1.0F}}
- static constexpr VkClearColorValue GRAY_V = {{0.5F, 0.5F, 0.5F, 1.0F}}
- static constexpr VkClearColorValue MAROON_V = {{0.5F, 0.0F, 0.0F, 1.0F}}
- static constexpr VkClearColorValue OLIVE_V = {{0.5F, 0.5F, 0.0F, 1.0F}}
- static constexpr VkClearColorValue LIME_V = {{0.0F, 1.0F, 0.0F, 1.0F}}
- static constexpr VkClearColorValue AQUA_V = {{0.0F, 1.0F, 1.0F, 1.0F}}
- static constexpr VkClearColorValue TEAL_V = {{0.0F, 0.5F, 0.5F, 1.0F}}
- static constexpr VkClearColorValue NAVY_V = {{0.0F, 0.0F, 0.5F, 1.0F}}
- static constexpr VkClearColorValue FUCHSIA_V = {{1.0F, 0.0F, 1.0F, 1.0F}}
- static constexpr VkClearColorValue NIGHT_BLUE_V = {{0.1F, 0.1F, 0.44F, 1.0F}}
- static constexpr VkClearColorValue SKY_BLUE_V = {{0.4F, 0.6F, 0.9F, 1.0F}}
- static constexpr VkClearColorValue SUNSET_V = {{1.0F, 0.5F, 0.0F, 1.0F}}
- static constexpr VkClearColorValue NIGHT_MODE_V = {{0.0F, 0.0F, 0.0F, 1.0F}}
- static constexpr std::array< std::pair< const char *, glm::vec3 >, 20 > COLORS
- static constexpr std::array< std::pair< const char *, VkClearColorValue >, 20 > CLEAR_COLORS

Static Private Attributes

- static constexpr float COLOR_MAX = 255.0F

6.7.1 Detailed Description

Class for colors.

Definition at line 24 of file Colors.hpp.

6.7.2 Member Data Documentation

6.7.2.1 AQUA

`glm::vec3 ven::Colors::AQUA = glm::vec3(0.0F, COLOR_MAX, COLOR_MAX) / COLOR_MAX [static], [constexpr]`

Definition at line 43 of file Colors.hpp.

6.7.2.2 AQUA_V

`VkClearColorValue ven::Colors::AQUA_V = {{0.0F, 1.0F, 1.0F, 1.0F}} [static], [constexpr]`

Definition at line 64 of file Colors.hpp.

6.7.2.3 BLACK

`glm::vec3 ven::Colors::BLACK = glm::vec3(0.0F) [static], [constexpr]`

Definition at line 31 of file Colors.hpp.

6.7.2.4 BLACK_V

`VkClearColorValue ven::Colors::BLACK_V = {{0.0F, 0.0F, 0.0F, 1.0F}} [static], [constexpr]`

Definition at line 52 of file Colors.hpp.

6.7.2.5 BLUE

`glm::vec3 ven::Colors::BLUE = glm::vec3(0.0F, 0.0F, COLOR_MAX) / COLOR_MAX [static], [constexpr]`

Definition at line 34 of file Colors.hpp.

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

6.7.2.6 BLUE_V

`VkClearColorValue ven::Colors::BLUE_V = {{0.0F, 0.0F, 1.0F, 1.0F}} [static], [constexpr]`

Definition at line 55 of file Colors.hpp.

6.7.2.7 CLEAR_COLORS

std::array<std::pair<const char*, VkClearColorValue>, 20> ven::Colors::CLEAR_COLORS [static], [constexpr]

Initial value:
```
= {{
                {"White", Colors::WHITE_V},
                {"Black", Colors::BLACK_V},
                {"Red", Colors::RED_V},
                {"Green", Colors::GREEN_V},
                {"Blue", Colors::BLUE_V},
                {"Yellow", Colors::YELLOW_V},
                {"Cyan", Colors::CYAN_V},
                {"Magenta", Colors::MAGENTA_V},
                {"Silver", Colors::SILVER_V},
                {"Gray", Colors::GRAY_V},
                {"Maroon", Colors::MAROON_V},
                {"Olive", Colors::OLIVE_V},
                {"Lime", Colors::LIME_V},
                {"Aqua", Colors::AQUA_V},
                {"Teal", Colors::TEAL_V},
                {"Navy", Colors::NAVY_V},
                {"Fuchsia", Colors::FUCHSIA_V},
                {"Night Blue", Colors::NIGHT_BLUE_V},
                {"Sky Blue", Colors::SKY_BLUE_V},
                {"Sunset", Colors::SUNSET_V}
        }}
```

Definition at line 96 of file Colors.hpp.

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

6.7.2.8 COLOR_MAX

float ven::Colors::COLOR_MAX = 255.0F [static], [constexpr], [private]

Definition at line 26 of file Colors.hpp.

6.7.2.9 COLORS

std::array<std::pair<const char*, glm::vec3>, 20> ven::Colors::COLORS [static], [constexpr]

Initial value:
```
= {{
                {"White", Colors::WHITE},
                {"Black", Colors::BLACK},
                {"Red", Colors::RED},
                {"Green", Colors::GREEN},
                {"Blue", Colors::BLUE},
                {"Yellow", Colors::YELLOW},
                {"Cyan", Colors::CYAN},
                {"Magenta", Colors::MAGENTA},
                {"Silver", Colors::SILVER},
                {"Gray", Colors::GRAY},
                {"Maroon", Colors::MAROON},
                {"Olive", Colors::OLIVE},
                {"Lime", Colors::LIME},
                {"Aqua", Colors::AQUA},
                {"Teal", Colors::TEAL},
                {"Navy", Colors::NAVY},
                {"Fuchsia", Colors::FUCHSIA},
                {"Night Blue", ven::Colors::NIGHT_BLUE},
                {"Sky Blue", Colors::SKY_BLUE},
                {"Sunset", Colors::SUNSET}
        }}
```

Definition at line 73 of file Colors.hpp.

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

6.7.2.10 CYAN

```
glm::vec3 ven::Colors::CYAN = glm::vec3(0.0F, COLOR_MAX, COLOR_MAX) / COLOR_MAX [static],
[constexpr]
```

Definition at line 36 of file Colors.hpp.

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

6.7.2.11 CYAN_V

```
VkClearColorValue ven::Colors::CYAN_V = {{0.0F, 1.0F, 1.0F, 1.0F}} [static], [constexpr]
```

Definition at line 57 of file Colors.hpp.

6.7.2.12 FUCHSIA

```
glm::vec3 ven::Colors::FUCHSIA = glm::vec3(COLOR_MAX, 0.0F, COLOR_MAX) / COLOR_MAX [static],
[constexpr]
```

Definition at line 46 of file Colors.hpp.

6.7.2.13 FUCHSIA_V

```
VkClearColorValue ven::Colors::FUCHSIA_V = {{1.0F, 0.0F, 1.0F, 1.0F}} [static], [constexpr]
```

Definition at line 67 of file Colors.hpp.

6.7.2.14 GRAY

```
glm::vec3 ven::Colors::GRAY = glm::vec3(128.0F, 128.0F, 128.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 39 of file Colors.hpp.

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

6.7.2.15 GRAY_V

```
VkClearColorValue ven::Colors::GRAY_V = {{0.5F, 0.5F, 0.5F, 1.0F}} [static], [constexpr]
```

Definition at line 60 of file Colors.hpp.

6.7.2.16 GREEN

```
glm::vec3 ven::Colors::GREEN = glm::vec3(0.0F, COLOR_MAX, 0.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 33 of file Colors.hpp.

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

6.7.2.17 GREEN_V

```
VkClearColorValue ven::Colors::GREEN_V = {{0.0F, 1.0F, 0.0F, 1.0F}} [static], [constexpr]
```

Definition at line 54 of file Colors.hpp.

6.7.2.18 LIME

```
glm::vec3 ven::Colors::LIME = glm::vec3(0.0F, COLOR_MAX, 0.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 42 of file Colors.hpp.

6.7.2.19 LIME_V

```
VkClearColorValue ven::Colors::LIME_V = {{0.0F, 1.0F, 0.0F, 1.0F}} [static], [constexpr]
```

Definition at line 63 of file Colors.hpp.

6.7.2.20 MAGENTA

```
glm::vec3 ven::Colors::MAGENTA = glm::vec3(COLOR_MAX, 0.0F, COLOR_MAX) / COLOR_MAX [static],
[constexpr]
```

Definition at line 37 of file Colors.hpp.

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

6.7.2.21 MAGENTA_V

```
VkClearColorValue ven::Colors::MAGENTA_V = {{1.0F, 0.0F, 1.0F, 1.0F}} [static], [constexpr]
```

Definition at line 58 of file Colors.hpp.

6.7.2.22 MAROON

```
glm::vec3 ven::Colors::MAROON = glm::vec3(128.0F, 0.0F, 0.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 40 of file Colors.hpp.

6.7.2.23 MAROON_V

```
VkClearColorValue ven::Colors::MAROON_V = {{0.5F, 0.0F, 0.0F, 1.0F}} [static], [constexpr]
```

Definition at line 61 of file Colors.hpp.

6.7.2.24 NAVY

```
glm::vec3 ven::Colors::NAVY = glm::vec3(0.0F, 0.0F, 128.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 45 of file Colors.hpp.

6.7.2.25 NAVY_V

```
VkClearColorValue ven::Colors::NAVY_V = {{0.0F, 0.0F, 0.5F, 1.0F}} [static], [constexpr]
```

Definition at line 66 of file Colors.hpp.

6.7.2.26 NIGHT_BLUE

```
glm::vec3 ven::Colors::NIGHT_BLUE = glm::vec3(25.0F, 25.0F, 112.0F) / COLOR_MAX [static],
[constexpr]
```

Definition at line 47 of file Colors.hpp.

6.7.2.27 NIGHT_BLUE_V

```
VkClearColorValue ven::Colors::NIGHT_BLUE_V = {{0.1F, 0.1F, 0.44F, 1.0F}} [static], [constexpr]
```

Definition at line 68 of file Colors.hpp.

6.7.2.28 NIGHT_MODE_V

```
VkClearColorValue ven::Colors::NIGHT_MODE_V = {{0.0F, 0.0F, 0.0F, 1.0F}} [static], [constexpr]
```

Definition at line 71 of file Colors.hpp.

6.7.2.29 OLIVE

```
glm::vec3 ven::Colors::OLIVE = glm::vec3(128.0F, 128.0F, 0.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 41 of file Colors.hpp.

6.7.2.30 OLIVE_V

```
VkClearColorValue ven::Colors::OLIVE_V = {{0.5F, 0.5F, 0.0F, 1.0F}} [static], [constexpr]
```

Definition at line 62 of file Colors.hpp.

6.7.2.31 RED

```
glm::vec3 ven::Colors::RED = glm::vec3(COLOR_MAX, 0.0F, 0.0F) / COLOR_MAX  [static], [constexpr]
```

Definition at line 32 of file Colors.hpp.

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

6.7.2.32 RED_V

```
VkClearColorValue ven::Colors::RED_V = {{1.0F, 0.0F, 0.0F, 1.0F}}  [static], [constexpr]
```

Definition at line 53 of file Colors.hpp.

6.7.2.33 SILVER

```
glm::vec3 ven::Colors::SILVER = glm::vec3(192.0F, 192.0F, 192.0F) / COLOR_MAX  [static], [constexpr]
```

Definition at line 38 of file Colors.hpp.

6.7.2.34 SILVER_V

```
VkClearColorValue ven::Colors::SILVER_V = {{0.75F, 0.75F, 0.75F, 1.0F}}  [static], [constexpr]
```

Definition at line 59 of file Colors.hpp.

6.7.2.35 SKY_BLUE

```
glm::vec3 ven::Colors::SKY_BLUE = glm::vec3(102.0F, 178.0F, 255.0F) / COLOR_MAX  [static],
[constexpr]
```

Definition at line 48 of file Colors.hpp.

6.7.2.36 SKY_BLUE_V

```
VkClearColorValue ven::Colors::SKY_BLUE_V = {{0.4F, 0.6F, 0.9F, 1.0F}}  [static], [constexpr]
```

Definition at line 69 of file Colors.hpp.

6.7.2.37 SUNSET

```
glm::vec3 ven::Colors::SUNSET = glm::vec3(255.0F, 128.0F, 0.0F) / COLOR_MAX  [static], [constexpr]
```

Definition at line 49 of file Colors.hpp.

6.7.2.38 SUNSET_V

```
VkClearColorValue ven::Colors::SUNSET_V = {{1.0F, 0.5F, 0.0F, 1.0F}} [static], [constexpr]
```

Definition at line 70 of file Colors.hpp.

6.7.2.39 TEAL

```
glm::vec3 ven::Colors::TEAL = glm::vec3(0.0F, 128.0F, 128.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 44 of file Colors.hpp.

6.7.2.40 TEAL_V

```
VkClearColorValue ven::Colors::TEAL_V = {{0.0F, 0.5F, 0.5F, 1.0F}} [static], [constexpr]
```

Definition at line 65 of file Colors.hpp.

6.7.2.41 WHITE

```
glm::vec3 ven::Colors::WHITE = glm::vec3(COLOR_MAX, COLOR_MAX, COLOR_MAX) / COLOR_MAX [static],
[constexpr]
```

Definition at line 30 of file Colors.hpp.

6.7.2.42 WHITE_V

```
VkClearColorValue ven::Colors::WHITE_V = {{1.0F, 1.0F, 1.0F, 1.0F}} [static], [constexpr]
```

Definition at line 51 of file Colors.hpp.

6.7.2.43 YELLOW

```
glm::vec3 ven::Colors::YELLOW = glm::vec3(COLOR_MAX, COLOR_MAX, 0.0F) / COLOR_MAX [static],
[constexpr]
```

Definition at line 35 of file Colors.hpp.

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

6.7.2.44 YELLOW_V

```
VkClearColorValue ven::Colors::YELLOW_V = {{1.0F, 1.0F, 0.0F, 1.0F}} [static], [constexpr]
```

Definition at line 56 of file Colors.hpp.

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

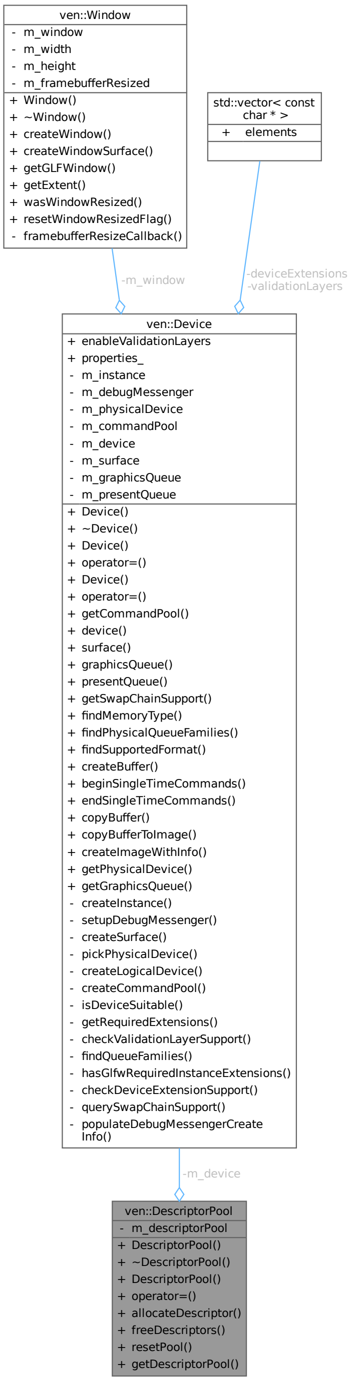
- /home/runner/work/VEngine/VEngine/include/VEngine/Colors.hpp

6.8 ven::DescriptorPool Class Reference

Class for descriptor pool.

`#include <DescriptorPool.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
- VkDescriptorPool getDescriptorPool () const

Private Attributes

- Device & m_device
- VkDescriptorPool m_descriptorPool

Friends

- class DescriptorWriter

6.8.1 Detailed Description

Class for descriptor pool.

Definition at line 21 of file DescriptorPool.hpp.

6.8.2 Constructor & Destructor Documentation

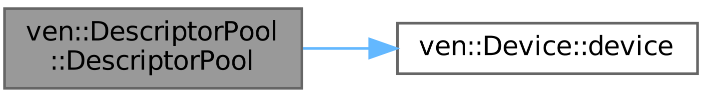
6.8.2.1 DescriptorPool() [1/2]

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

Definition at line 20 of file descriptorPool.cpp.

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

Here is the call graph for this function:



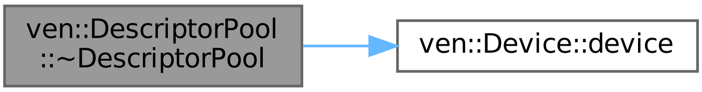
6.8.2.2 ∼DescriptorPool()

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

Definition at line 46 of file DescriptorPool.hpp.

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

Here is the call graph for this function:



6.8.2.3 DescriptorPool() [2/2]

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

6.8.3 Member Function Documentation

6.8.3.1 allocateDescriptor()

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

Definition at line 35 of file descriptorPool.cpp.

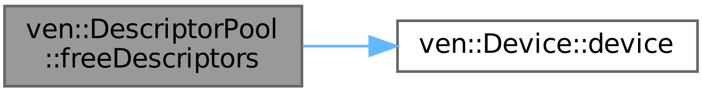
6.8.3.2 freeDescriptors()

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

Definition at line 51 of file DescriptorPool.hpp.

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

Here is the call graph for this function:



6.8.3.3 getDescriptorPool()

```
VkDescriptorPool ven::DescriptorPool::getDescriptorPool () const  [inline], [nodiscard]
```

Definition at line 54 of file DescriptorPool.hpp.

References m_descriptorPool.

6.8.3.4 operator=()

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

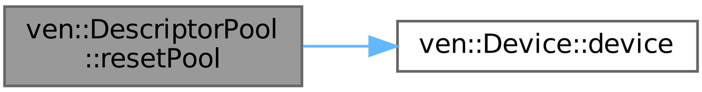
6.8.3.5 resetPool()

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

Definition at line 52 of file DescriptorPool.hpp.

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

Here is the call graph for this function:



6.8.4 Friends And Related Symbol Documentation

6.8.4.1 DescriptorWriter

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

Definition at line 60 of file DescriptorPool.hpp.

6.8.5 Member Data Documentation

6.8.5.1 m_descriptorPool

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

Definition at line 59 of file DescriptorPool.hpp.

Referenced by DescriptorPool(), freeDescriptors(), getDescriptorPool(), resetPool(), and ∼DescriptorPool().

6.8.5.2 m_device

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

Definition at line 58 of file DescriptorPool.hpp.

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

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

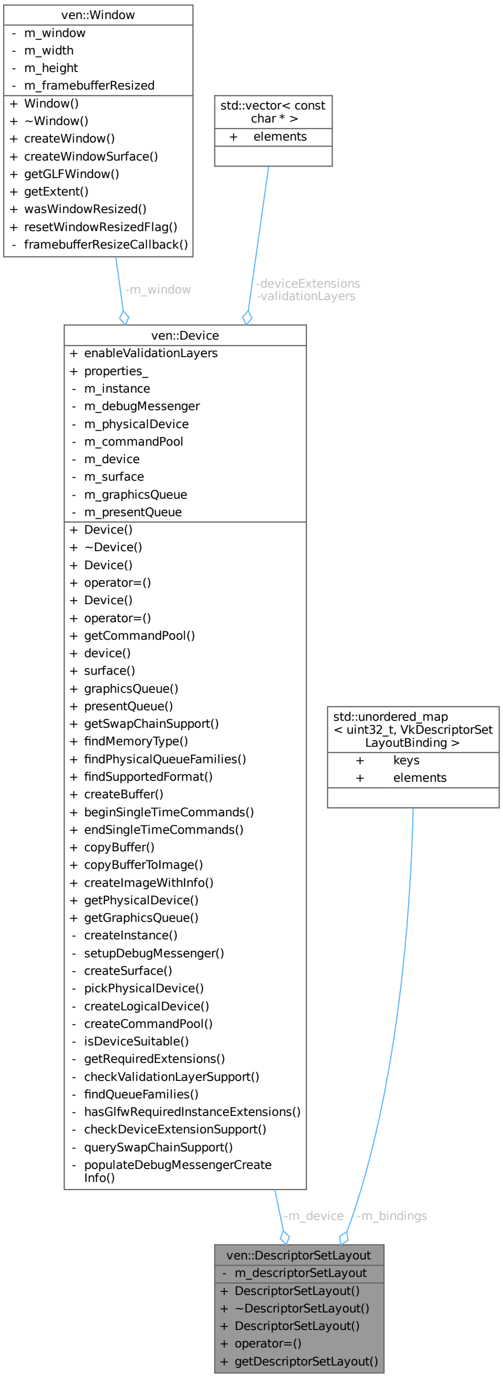
- /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorPool.hpp
- /home/runner/work/VEngine/VEngine/src/descriptors/descriptorPool.cpp

6.9 ven::DescriptorSetLayout Class Reference

Class for descriptor set layout.

```
#include <DescriptorSetLayout.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

6.9.1 Detailed Description

Class for descriptor set layout.

Definition at line 21 of file DescriptorSetLayout.hpp.

6.9.2 Constructor & Destructor Documentation

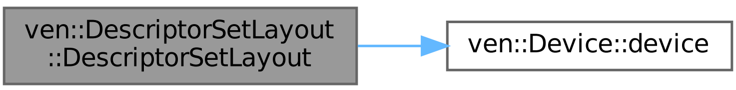
6.9.2.1 DescriptorSetLayout() [1/2]

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

Definition at line 17 of file descriptorSetLayout.cpp.

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

Here is the call graph for this function:



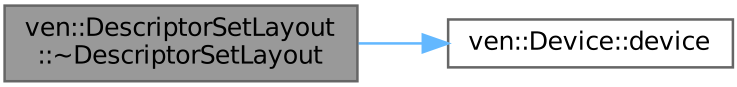
6.9.2.2 ∼DescriptorSetLayout()

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

Definition at line 42 of file DescriptorSetLayout.hpp.

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

Here is the call graph for this function:



6.9.2.3 DescriptorSetLayout() [2/2]

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

6.9.3 Member Function Documentation

6.9.3.1 getDescriptorSetLayout()

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

Definition at line 46 of file DescriptorSetLayout.hpp.

References m_descriptorSetLayout.

6.9.3.2 operator=()

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

6.9.4 Friends And Related Symbol Documentation

6.9.4.1 DescriptorWriter

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

Definition at line 54 of file DescriptorSetLayout.hpp.

6.9.5 Member Data Documentation

6.9.5.1 m_bindings

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

Definition at line 52 of file DescriptorSetLayout.hpp.

Referenced by ven::DescriptorWriter::writeBuffer().

6.9.5.2 m_descriptorSetLayout

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

Definition at line 51 of file DescriptorSetLayout.hpp.

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

6.9.5.3 m_device

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

Definition at line 50 of file DescriptorSetLayout.hpp.

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

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

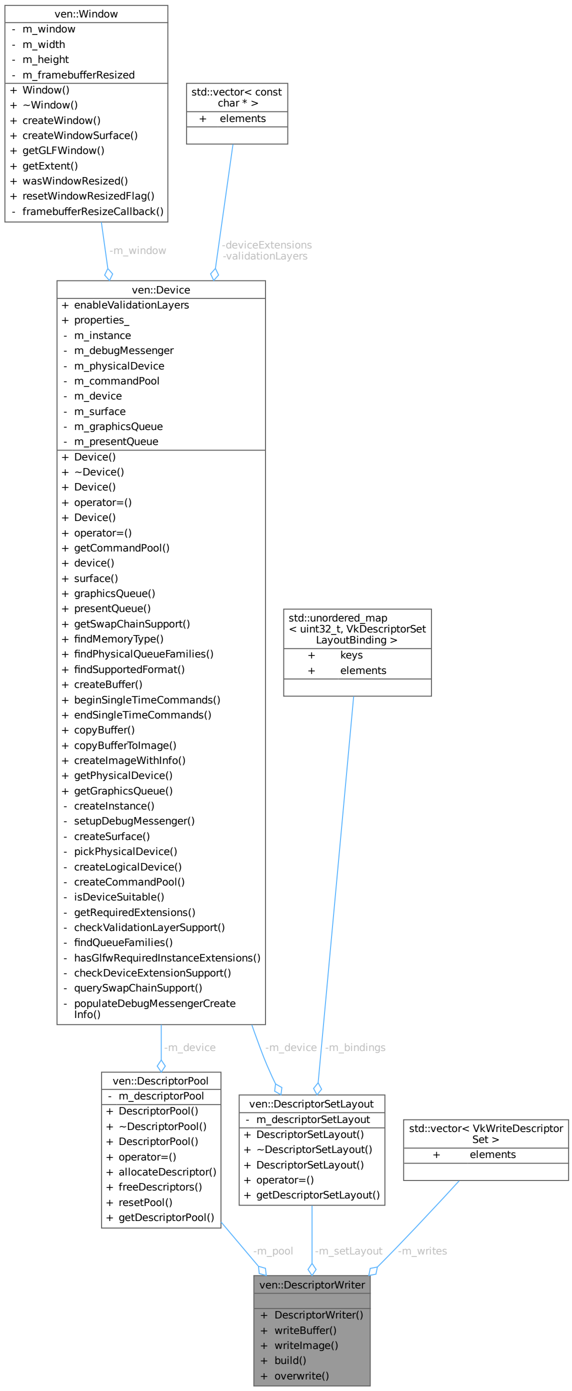
- /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorSetLayout.hpp
- /home/runner/work/VEngine/VEngine/src/descriptors/descriptorSetLayout.cpp

6.10 ven::DescriptorWriter Class Reference

Class for descriptor writer.

```
#include <DescriptorWriter.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

6.10.1 Detailed Description

Class for descriptor writer.

Definition at line 22 of file DescriptorWriter.hpp.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 DescriptorWriter()

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

Definition at line 26 of file DescriptorWriter.hpp.

6.10.3 Member Function Documentation

6.10.3.1 build()

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

Definition at line 43 of file descriptorWriter.cpp.

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

Here is the caller graph for this function:

6.10.3.2 overwrite()

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

Definition at line 52 of file descriptorWriter.cpp.

6.10.3.3 writeBuffer()

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

Definition at line 5 of file descriptorWriter.cpp.

References ven::DescriptorSetLayout::m_bindings, m_setLayout, and m_writes.

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

Here is the caller graph for this function:

6.10.3.4 writeImage()

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

Definition at line 24 of file descriptorWriter.cpp.

6.10.4 Member Data Documentation

6.10.4.1 m_pool

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

Definition at line 37 of file DescriptorWriter.hpp.

6.10.4.2 m_setLayout

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

Definition at line 36 of file DescriptorWriter.hpp.

Referenced by writeBuffer().

6.10.4.3 m_writes

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

Definition at line 38 of file DescriptorWriter.hpp.

Referenced by writeBuffer().

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

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

6.11 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, Vk↩ FormatFeatureFlags 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 layer↩ Count) const
- void createImageWithInfo (const VkImageCreateInfo &imageInfo, VkMemoryPropertyFlags properties, Vk↩ Image &image, VkDeviceMemory &imageMemory) const
- VkPhysicalDevice getPhysicalDevice () const
- VkQueue getGraphicsQueue () const

Public Attributes

- const bool enableValidationLayers = true
- VkPhysicalDeviceProperties 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 m_instance
- VkDebugUtilsMessengerEXT m_debugMessenger
- VkPhysicalDevice m_physicalDevice = VK_NULL_HANDLE
- Window & m_window
- VkCommandPool m_commandPool
- VkDevice m_device
- VkSurfaceKHR m_surface
- VkQueue m_graphicsQueue
- VkQueue m_presentQueue
- const std::vector< const char * > validationLayers = {"VK_LAYER_KHRONOS_validation"}
- const std::vector< const char * > deviceExtensions = {VK_KHR_SWAPCHAIN_EXTENSION_NAME}

6.11.1 Detailed Description

Definition at line 29 of file Device.hpp.

6.11.2 Constructor & Destructor Documentation

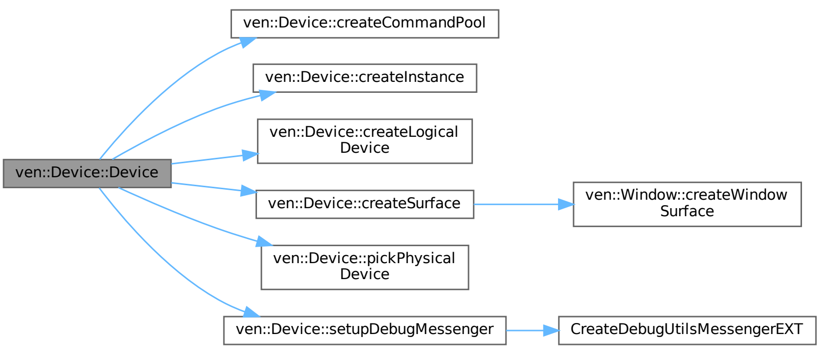
6.11.2.1 Device() [1/3]

```
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:



6.11.2.2 ∼Device()

```
ven::Device::∼Device ()
```

Definition at line 44 of file device.cpp.

References DestroyDebugUtilsMessengerEXT().

Here is the call graph for this function:

6.11.2.3 Device() [2/3]

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

6.11.2.4 Device() [3/3]

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

6.11.3 Member Function Documentation

6.11.3.1 beginSingleTimeCommands()

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

Definition at line 411 of file device.cpp.

6.11.3.2 checkDeviceExtensionSupport()

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

Definition at line 289 of file device.cpp.

6.11.3.3 checkValidationLayerSupport()

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

Definition at line 225 of file device.cpp.

6.11.3.4 copyBuffer()

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

Definition at line 445 of file device.cpp.

6.11.3.5 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.

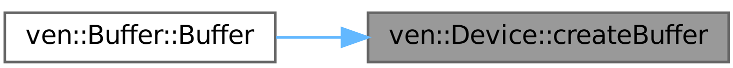
6.11.3.6 createBuffer()

```
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:



6.11.3.7 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:

6.11.3.8 createImageWithInfo()

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

Definition at line 479 of file device.cpp.

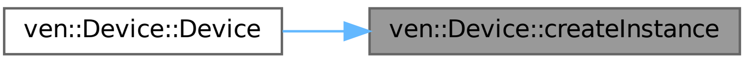
6.11.3.9 createInstance()

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

Definition at line 57 of file device.cpp.

Referenced by Device().

Here is the caller graph for this function:



6.11.3.10 createLogicalDevice()

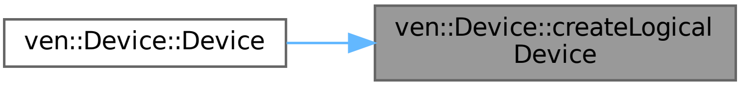
```
void ven::Device::createLogicalDevice ()  [private]
```

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:



6.11.3.11 createSurface()

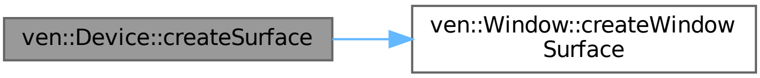
```
void ven::Device::createSurface ()  [inline], [private]
```

Definition at line 76 of file Device.hpp.

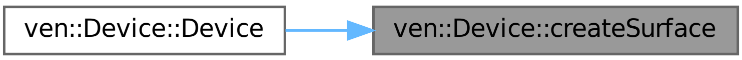
References ven::Window::createWindowSurface(), m_instance, m_surface, and m_window.

Referenced by Device().

Here is the call graph for this function:



Here is the caller graph for this function:



6.11.3.12 device()

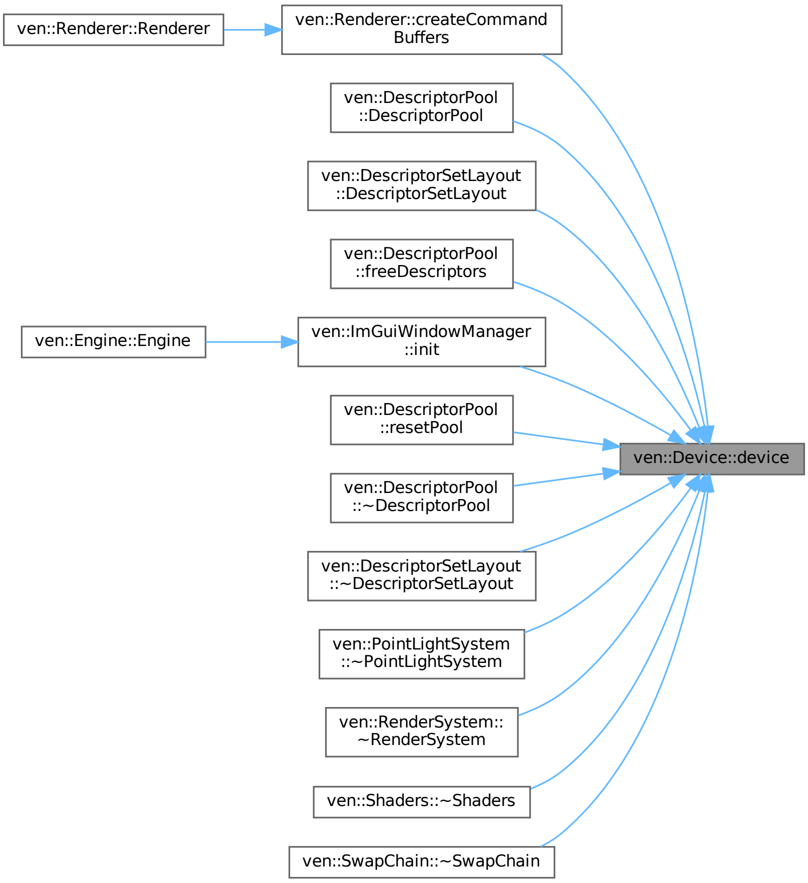
```
VkDevice ven::Device::device () const  [inline], [nodiscard]
```

Definition at line 48 of file Device.hpp.

References m_device.

Referenced by ven::Renderer::createCommandBuffers(), ven::DescriptorPool::DescriptorPool(), ven::DescriptorSetLayout::DescriptorS ven::DescriptorPool::freeDescriptors(), ven::ImGuiWindowManager::init(), ven::DescriptorPool::resetPool(), ven::DescriptorPool::∼DescriptorPool(), ven::DescriptorSetLayout::∼DescriptorSetLayout(), ven::PointLightSystem::∼PointLightSyste ven::RenderSystem::∼RenderSystem(), ven::Shaders::∼Shaders(), and ven::SwapChain::∼SwapChain().

Here is the caller graph for this function:



6.11.3.13 endSingleTimeCommands()

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

Definition at line 430 of file device.cpp.

6.11.3.14 findMemoryType()

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

Definition at line 369 of file device.cpp.

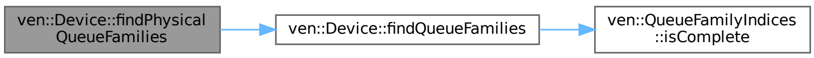
6.11.3.15 findPhysicalQueueFamilies()

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

Definition at line 55 of file Device.hpp.

References findQueueFamilies(), and m_physicalDevice.

Here is the call graph for this function:



6.11.3.16 findQueueFamilies()

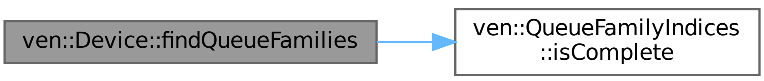
```
ven::QueueFamilyIndices ven::Device::findQueueFamilies (
            VkPhysicalDevice device) const  [private]
```

Definition at line 305 of file device.cpp.

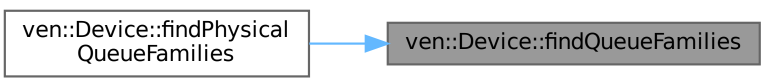
References ven::QueueFamilyIndices::graphicsFamily, ven::QueueFamilyIndices::graphicsFamilyHasValue, ven::QueueFamilyIndices::isComplete(), ven::QueueFamilyIndices::presentFamily, and ven::QueueFamilyIndices::presentFamilyHasV

Referenced by findPhysicalQueueFamilies().

Here is the call graph for this function:



Here is the caller graph for this function:



6.11.3.17 findSupportedFormat()

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

Definition at line 355 of file device.cpp.

6.11.3.18 getCommandPool()

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

Definition at line 47 of file Device.hpp.

References m_commandPool.

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

Here is the caller graph for this function:

6.11.3.19 getGraphicsQueue()

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

Definition at line 68 of file Device.hpp.

References m_graphicsQueue.

6.11.3.20 getPhysicalDevice()

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

Definition at line 67 of file Device.hpp.

References m_physicalDevice.

Referenced by ven::ImGuiWindowManager::init().

Here is the caller graph for this function:

6.11.3.21 getRequiredExtensions()

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

Definition at line 250 of file device.cpp.

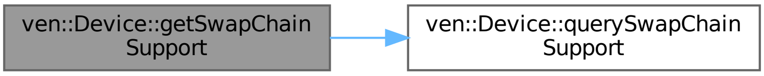
6.11.3.22 getSwapChainSupport()

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

Definition at line 53 of file Device.hpp.

References m_physicalDevice, and querySwapChainSupport().

Here is the call graph for this function:



6.11.3.23 graphicsQueue()

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

Definition at line 50 of file Device.hpp.

References m_graphicsQueue.

Referenced by ven::ImGuiWindowManager::init().

Here is the caller graph for this function:

6.11.3.24 hasGlfwRequiredInstanceExtensions()

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

Definition at line 265 of file device.cpp.

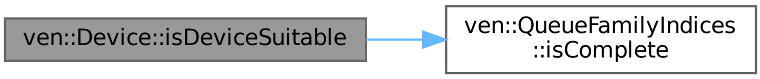
6.11.3.25 isDeviceSuitable()

```
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:



6.11.3.26 operator=() [1/2]

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

6.11.3.27 operator=() [2/2]

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

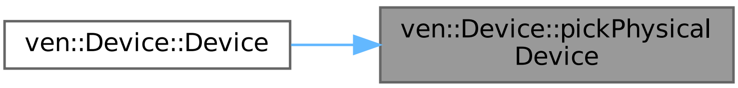
6.11.3.28 pickPhysicalDevice()

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

Definition at line 98 of file device.cpp.

Referenced by Device().

Here is the caller graph for this function:



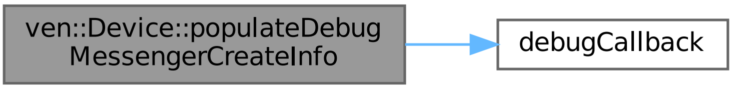
6.11.3.29 populateDebugMessengerCreateInfo()

```
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:



6.11.3.30 presentQueue()

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

Definition at line 51 of file Device.hpp.

References m_presentQueue.

6.11.3.31 querySwapChainSupport()

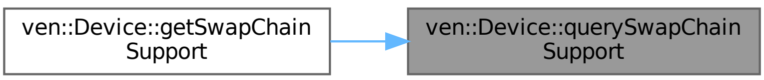
```
ven::SwapChainSupportDetails ven::Device::querySwapChainSupport (
            VkPhysicalDevice device) const  [private]
```

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:



6.11.3.32 setupDebugMessenger()

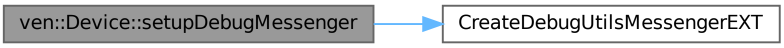
```
void ven::Device::setupDebugMessenger ( ) [private]
```

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:

6.11.3.33 surface()

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

Definition at line 49 of file Device.hpp.

References m_surface.

6.11.4 Member Data Documentation

6.11.4.1 deviceExtensions

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

Definition at line 103 of file Device.hpp.

6.11.4.2 enableValidationLayers

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

Definition at line 36 of file Device.hpp.

6.11.4.3 m_commandPool

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

Definition at line 95 of file Device.hpp.

Referenced by getCommandPool().

6.11.4.4 m_debugMessenger

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

Definition at line 92 of file Device.hpp.

6.11.4.5 m_device

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

Definition at line 97 of file Device.hpp.

Referenced by device().

6.11.4.6 m_graphicsQueue

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

Definition at line 99 of file Device.hpp.

Referenced by getGraphicsQueue(), and graphicsQueue().

6.11.4.7 m_instance

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

Definition at line 91 of file Device.hpp.

Referenced by createSurface().

6.11.4.8 m_physicalDevice

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

Definition at line 93 of file Device.hpp.

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

6.11.4.9 m_presentQueue

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

Definition at line 100 of file Device.hpp.

Referenced by presentQueue().

6.11.4.10 m_surface

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

Definition at line 98 of file Device.hpp.

Referenced by createSurface(), and surface().

6.11.4.11 m_window

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

Definition at line 94 of file Device.hpp.

Referenced by createSurface().

6.11.4.12 properties_

`VkPhysicalDeviceProperties ven::Device::properties_`

Definition at line 70 of file Device.hpp.

6.11.4.13 validationLayers

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

Definition at line 102 of file Device.hpp.

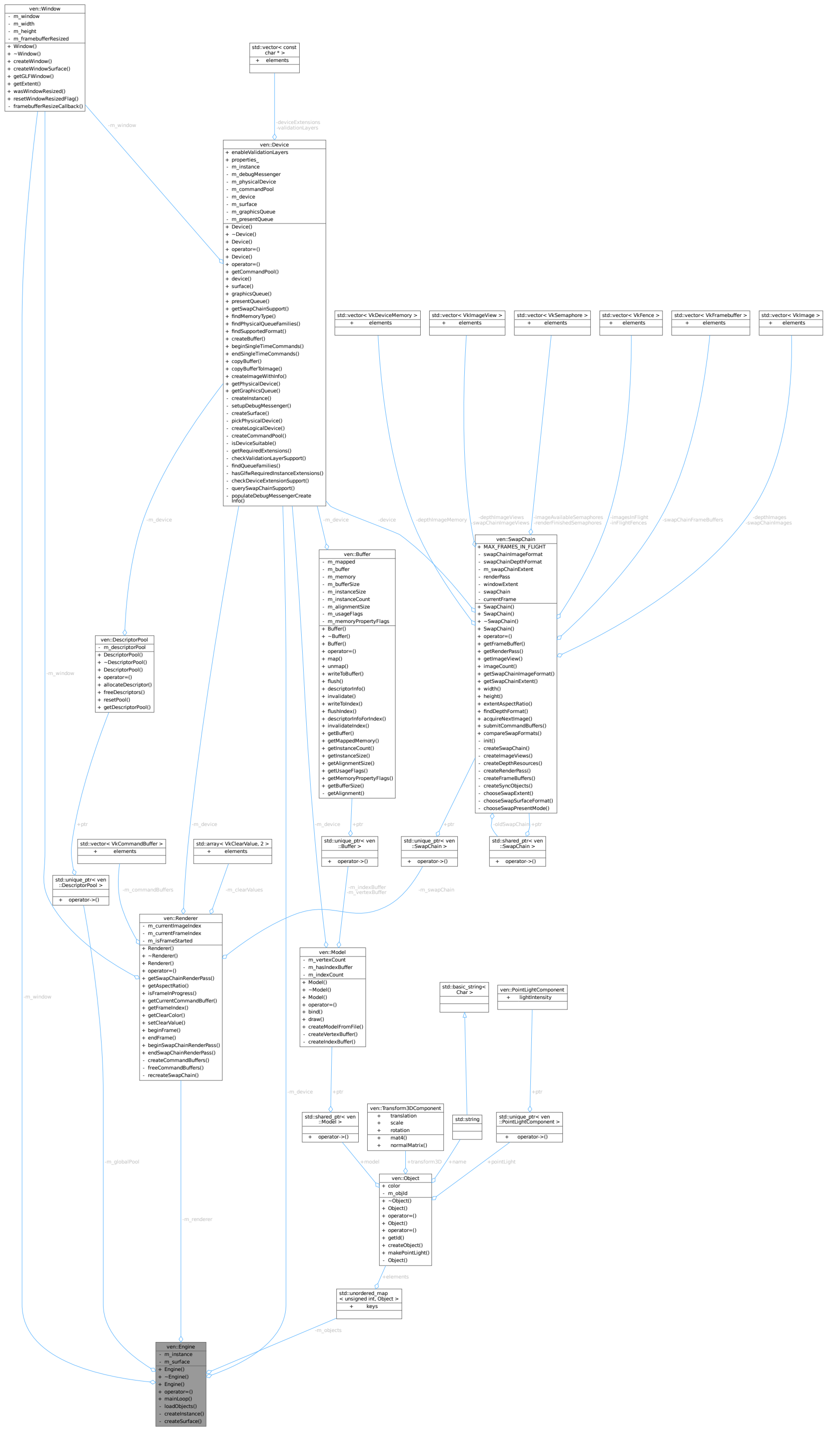
The documentation for this class was generated from the following files:

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

6.12 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
- 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}

6.12.1 Detailed Description

Definition at line 22 of file Engine.hpp.

6.12.2 Constructor & Destructor Documentation

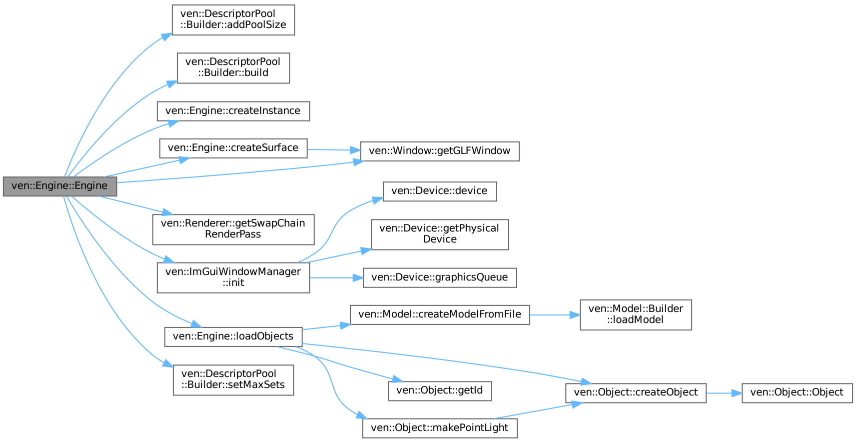
6.12.2.1 Engine() [1/2]

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

Definition at line 15 of file engine.cpp.

References ven::DescriptorPool::Builder::addPoolSize(), ven::DescriptorPool::Builder::build(), createInstance(), createSurface(), ven::Window::getGLFWindow(), ven::Renderer::getSwapChainRenderPass(), ven::ImGuiWindowManager::init(), loadObjects(), m_device, m_globalPool, m_instance, m_renderer, m_window, ven::SwapChain::MAX_FRAMES_IN_FLIGHT, and ven::DescriptorPool::Builder::setMaxSets().

Here is the call graph for this function:



6.12.2.2 ∼Engine()

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

6.12.2.3 Engine() [2/2]

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

6.12.3 Member Function Documentation

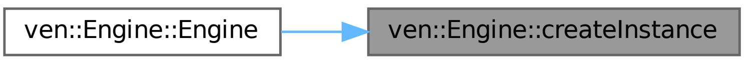
6.12.3.1 createInstance()

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

Definition at line 24 of file engine.cpp.

Referenced by Engine().

Here is the caller graph for this function:



6.12.3.2 createSurface()

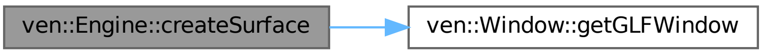
```
void ven::Engine::createSurface ( ) [inline], [private]
```

Definition at line 49 of file Engine.hpp.

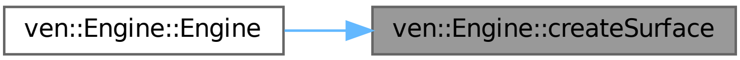
References ven::Window::getGLFWindow(), m_instance, m_surface, and m_window.

Referenced by Engine().

Here is the call graph for this function:



Here is the caller graph for this function:



6.12.3.3 loadObjects()

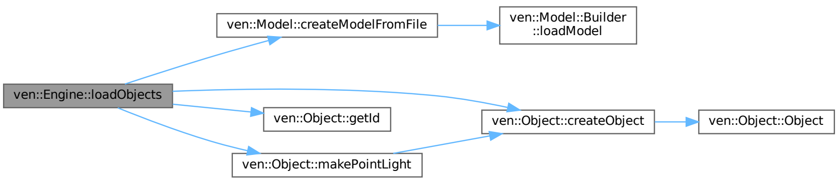
```
void ven::Engine::loadObjects ( ) [private]
```

Definition at line 43 of file engine.cpp.

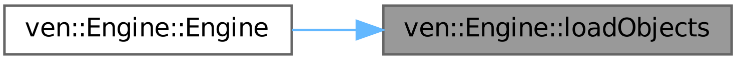
References ven::Colors::BLUE, ven::Object::color, ven::Model::createModelFromFile(), ven::Object::createObject(), ven::Colors::CYAN, ven::Object::getId(), ven::Colors::GREEN, ven::Colors::MAGENTA, ven::Object::makePointLight(), ven::Object::model, ven::Object::name, ven::Colors::RED, ven::Transform3DComponent::scale, ven::Object::transform3D, ven::Transform3DComponent::translation, and ven::Colors::YELLOW.

Referenced by Engine().

Here is the call graph for this function:



Here is the caller graph for this function:



6.12.3.4 mainLoop()

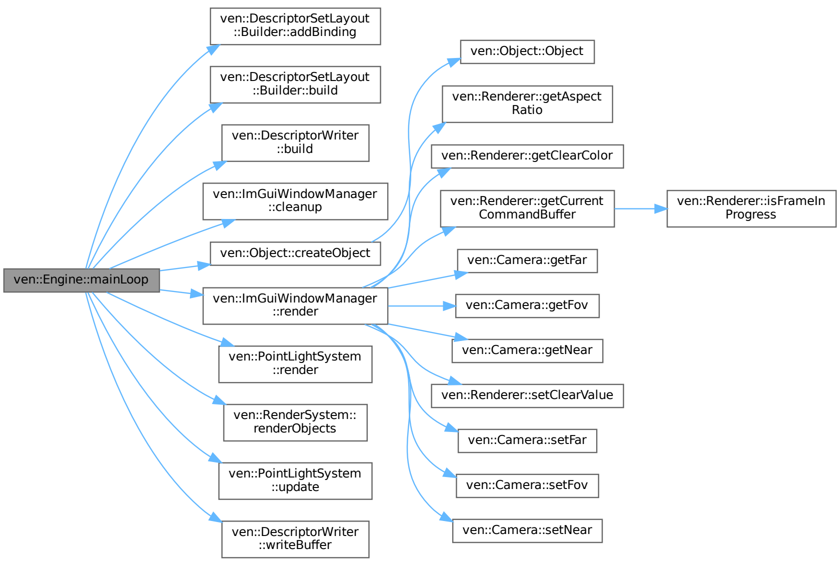
```
void ven::Engine::mainLoop ()
```

Definition at line 90 of file engine.cpp.

References ven::DescriptorSetLayout::Builder::addBinding(), ven::DescriptorSetLayout::Builder::build(), ven::DescriptorWriter::build(), ven::ImGuiWindowManager::cleanup(), ven::Object::createObject(), ven::DEFAULT_POSITION, ven::SwapChain::MAX_FRAMES_IN ven::ImGuiWindowManager::render(), ven::PointLightSystem::render(), ven::RenderSystem::renderObjects(), ven::Transform3DComponent::rotation, 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:

6.12.3.5 operator=()

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

6.12.4 Member Data Documentation

6.12.4.1 m_device

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

Definition at line 39 of file Engine.hpp.

Referenced by Engine().

6.12.4.2 m_globalPool

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

Definition at line 42 of file Engine.hpp.

Referenced by Engine().

6.12.4.3 m_instance

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

Definition at line 45 of file Engine.hpp.

Referenced by createSurface(), and Engine().

6.12.4.4 m_objects

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

Definition at line 43 of file Engine.hpp.

6.12.4.5 m_renderer

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

Definition at line 40 of file Engine.hpp.

Referenced by Engine().

6.12.4.6 m_surface

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

Definition at line 46 of file Engine.hpp.

Referenced by createSurface().

6.12.4.7 m_window

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

Definition at line 38 of file Engine.hpp.

Referenced by createSurface(), and Engine().

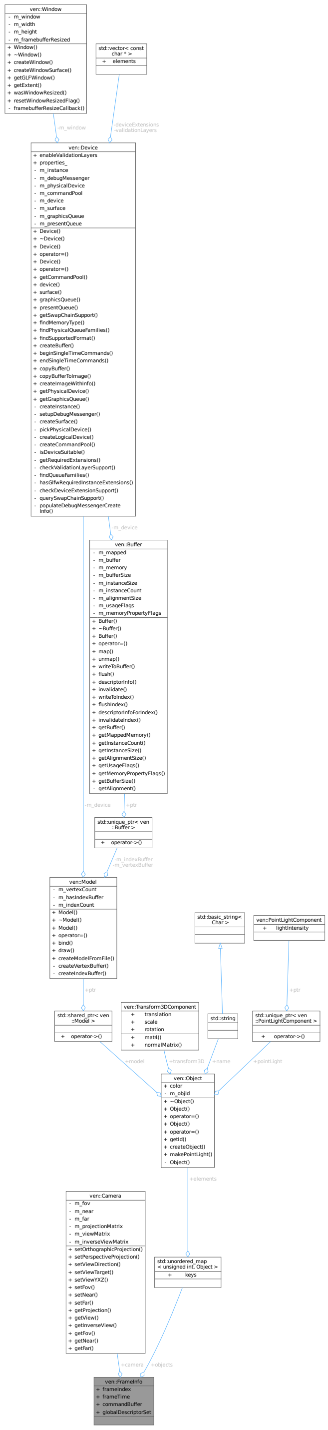
The documentation for this class was generated from the following files:

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

6.13 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.

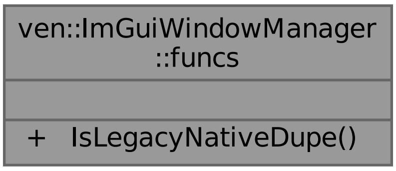
Referenced by ven::PointLightSystem::render(), ven::RenderSystem::renderObjects(), and ven::PointLightSystem::update().

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

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

6.14 ven::ImGuiWindowManager::funcs Struct Reference

Collaboration diagram for ven::ImGuiWindowManager::funcs:



Static Public Member Functions

- static bool IsLegacyNativeDupe (ImGuiKey key)

6.14.1 Detailed Description

Definition at line 42 of file ImGuiWindowManager.hpp.

6.14.2 Member Function Documentation

6.14.2.1 IsLegacyNativeDupe()

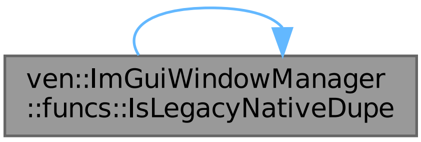
```
static bool ven::ImGuiWindowManager::funcs::IsLegacyNativeDupe (
            ImGuiKey key)  [inline], [static]
```

Definition at line 42 of file ImGuiWindowManager.hpp.

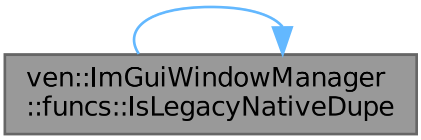
References IsLegacyNativeDupe().

Referenced by IsLegacyNativeDupe().

Here is the call graph for this function:



Here is the caller graph for this function:



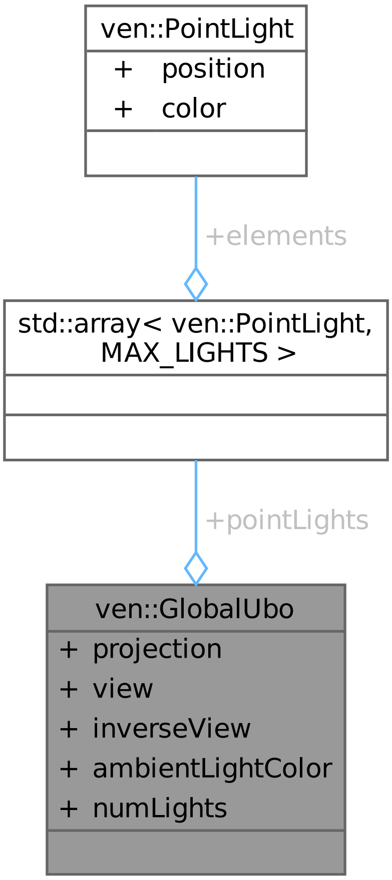
The documentation for this struct was generated from the following file:

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

6.15 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

6.15.1 Detailed Description

Definition at line 24 of file FrameInfo.hpp.

6.15.2 Member Data Documentation

6.15.2.1 ambientLightColor

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

Definition at line 29 of file FrameInfo.hpp.

6.15.2.2 inverseView

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

Definition at line 28 of file FrameInfo.hpp.

6.15.2.3 numLights

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

Definition at line 31 of file FrameInfo.hpp.

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

6.15.2.4 pointLights

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

Definition at line 30 of file FrameInfo.hpp.

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

6.15.2.5 projection

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

Definition at line 26 of file FrameInfo.hpp.

6.15.2.6 view

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

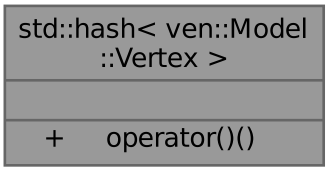
Definition at line 27 of file FrameInfo.hpp.

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

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

6.16 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

6.16.1 Detailed Description

Definition at line 15 of file model.cpp.

6.16.2 Member Function Documentation

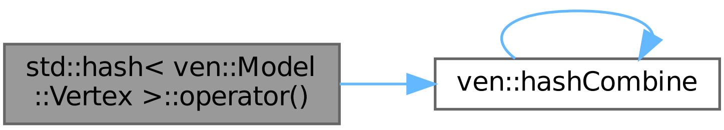
6.16.2.1 operator()()

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

Definition at line 16 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:

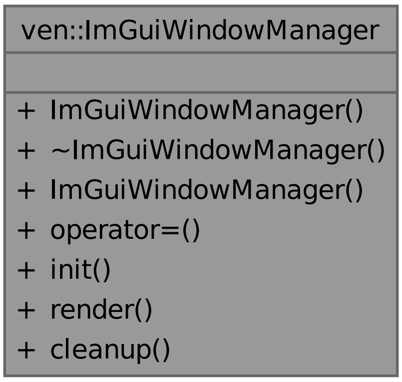
- /home/runner/work/VEngine/VEngine/src/model.cpp

6.17 ven::ImGuiWindowManager Class Reference

Class for ImGui window manager.

```
#include <ImGuiWindowManager.hpp>
```

Collaboration diagram for ven::ImGuiWindowManager:



Classes

- struct funcs

Public Member Functions

- ImGuiWindowManager ()=default
- ∼ImGuiWindowManager ()=default
- ImGuiWindowManager (const ImGuiWindowManager &)=delete
- ImGuiWindowManager & operator= (const ImGuiWindowManager &)=delete

Static Public Member Functions

- static void init (GLFWwindow *window, VkInstance instance, Device *device, VkRenderPass renderPass)
- static void render (Renderer *renderer, std::unordered_map< unsigned int, Object > &objects, ImGuiIO &io, Object &cameraObj, Camera &camera, KeyboardController &cameraController, VkPhysicalDevice physicalDevice)
- static void cleanup ()

6.17.1 Detailed Description

Class for ImGui window manager.

Definition at line 26 of file ImGuiWindowManager.hpp.

6.17.2 Constructor & Destructor Documentation

6.17.2.1 ImGuiWindowManager() [1/2]

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

6.17.2.2 ∼ImGuiWindowManager()

```
ven::ImGuiWindowManager::∼ImGuiWindowManager ()  [default]
```

6.17.2.3 ImGuiWindowManager() [2/2]

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

6.17.3 Member Function Documentation

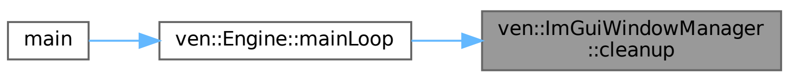
6.17.3.1 cleanup()

```
void ven::ImGuiWindowManager::cleanup ()  [static]
```

Definition at line 238 of file ImGuiWindowManager.cpp.

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

Here is the caller graph for this function:



6.17.3.2 init()

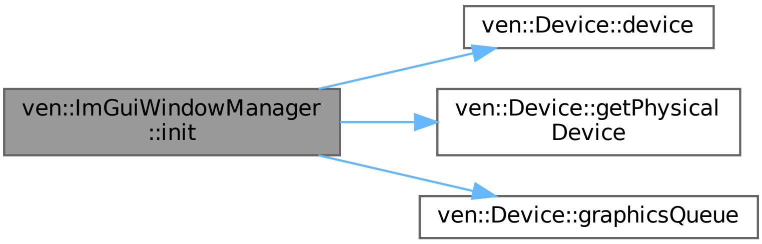
```
void ven::ImGuiWindowManager::init (
            GLFWwindow * window,
            VkInstance instance,
            Device * device,
            VkRenderPass renderPass)  [static]
```

Definition at line 8 of file ImGuiWindowManager.cpp.

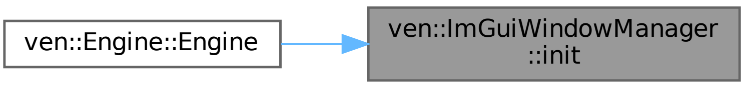
References ven::Device::device(), ven::Device::getPhysicalDevice(), and ven::Device::graphicsQueue().

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

Here is the call graph for this function:



Here is the caller graph for this function:



6.17.3.3 operator=()

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

6.17.3.4 render()

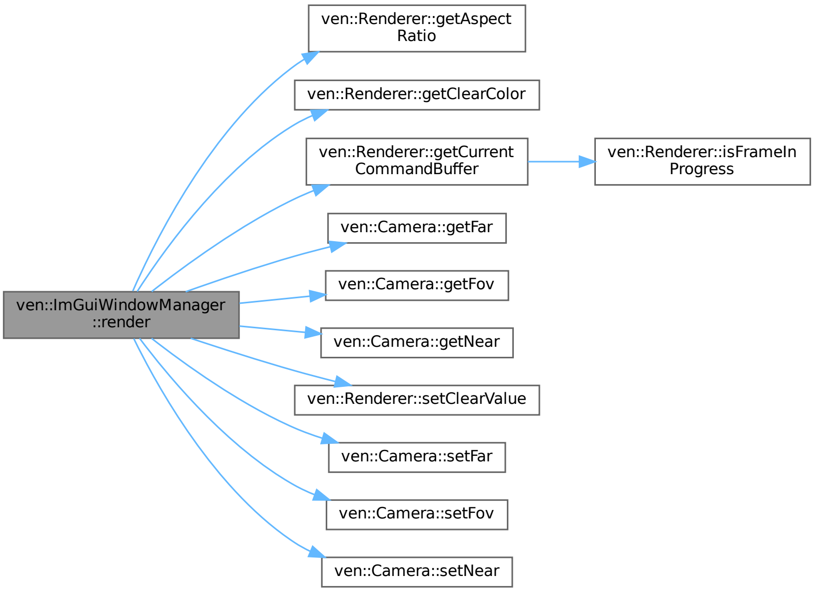
```
void ven::ImGuiWindowManager::render (
            Renderer * renderer,
            std::unordered_map< unsigned int, Object > & objects,
            ImGuiIO & io,
            Object & cameraObj,
            Camera & camera,
            KeyboardController & cameraController,
            VkPhysicalDevice physicalDevice) [static]
```

Definition at line 54 of file ImGuiWindowManager.cpp.

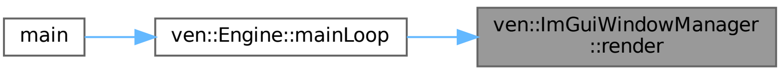
References ven::Colors::CLEAR_COLORS, ven::Colors::COLORS, ven::DEFAULT_FAR, ven::DEFAULT_FOV, ven::DEFAULT_LOOK_SPEED, ven::DEFAULT_MOVE_SPEED, ven::DEFAULT_NEAR, ven::DEFAULT_POSITION, ven::DEFAULT_ROTATION, ven::Renderer::getAspectRatio(), ven::Renderer::getClearColor(), ven::Renderer::getCurrentCommandBu ven::Camera::getFar(), ven::Camera::getFov(), ven::Camera::getNear(), ven::Colors::GRAY, ven::KeyboardController::m_lookSpeed, ven::KeyboardController::m_moveSpeed, ven::Transform3DComponent::rotation, ven::Renderer::setClearValue(), ven::Camera::setFar(), ven::Camera::setFov(), ven::Camera::setNear(), ven::Object::transform3D, and ven::Transform3DComponent::

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

Here is the call graph for this function:



Here is the caller graph for this function:



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

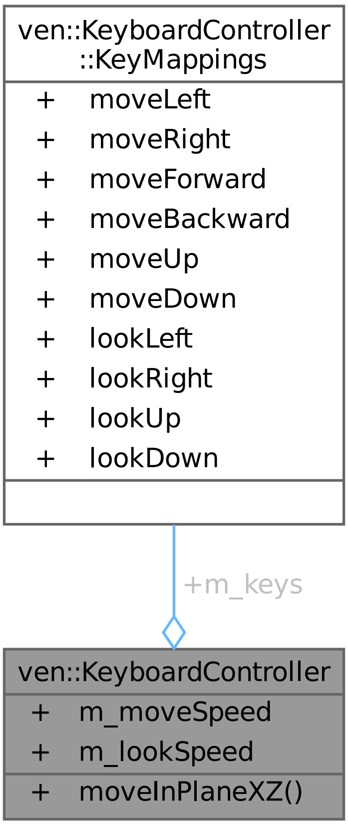
- /home/runner/work/VEngine/VEngine/include/VEngine/ImGuiWindowManager.hpp
- /home/runner/work/VEngine/VEngine/src/ImGuiWindowManager.cpp

6.18 ven::KeyboardController Class Reference

Class for keyboard controller.

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

Collaboration diagram for ven::KeyboardController:



Classes

- struct KeyMappings

Public Member Functions

- void moveInPlaneXZ (GLFWwindow ∗window, float dt, Object &object, bool ∗showDebugWindow) const

Public Attributes

- KeyMappings m_keys {}
- float m_moveSpeed {DEFAULT_MOVE_SPEED}
- float m_lookSpeed {DEFAULT_LOOK_SPEED}

6.18.1 Detailed Description

Class for keyboard controller.

Definition at line 22 of file KeyboardController.hpp.

6.18.2 Member Function Documentation

6.18.2.1 moveInPlaneXZ()

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

6.18.3 Member Data Documentation

6.18.3.1 m_keys

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

Definition at line 41 of file KeyboardController.hpp.

Referenced by moveInPlaneXZ().

6.18.3.2 m_lookSpeed

```
float ven::KeyboardController::m_lookSpeed {DEFAULT_LOOK_SPEED}
```

Definition at line 43 of file KeyboardController.hpp.

Referenced by moveInPlaneXZ(), and ven::ImGuiWindowManager::render().

6.18.3.3 m_moveSpeed

```
float ven::KeyboardController::m_moveSpeed {DEFAULT_MOVE_SPEED}
```

Definition at line 42 of file KeyboardController.hpp.

Referenced by moveInPlaneXZ(), and ven::ImGuiWindowManager::render().

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

6.19 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

6.19.1 Detailed Description

Definition at line 26 of file KeyboardController.hpp.

6.19.2 Member Data Documentation

6.19.2.1 lookDown

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

Definition at line 36 of file KeyboardController.hpp.

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

6.19.2.2 lookLeft

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

Definition at line 33 of file KeyboardController.hpp.

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

6.19.2.3 lookRight

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

Definition at line 34 of file KeyboardController.hpp.

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

6.19.2.4 lookUp

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

Definition at line 35 of file KeyboardController.hpp.

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

6.19.2.5 moveBackward

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

Definition at line 30 of file KeyboardController.hpp.

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

6.19.2.6 moveDown

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

Definition at line 32 of file KeyboardController.hpp.

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

6.19.2.7 moveForward

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

Definition at line 29 of file KeyboardController.hpp.

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

6.19.2.8 moveLeft

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

Definition at line 27 of file KeyboardController.hpp.

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

6.19.2.9 moveRight

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

Definition at line 28 of file KeyboardController.hpp.

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

6.19.2.10 moveUp

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

Definition at line 31 of file KeyboardController.hpp.

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

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

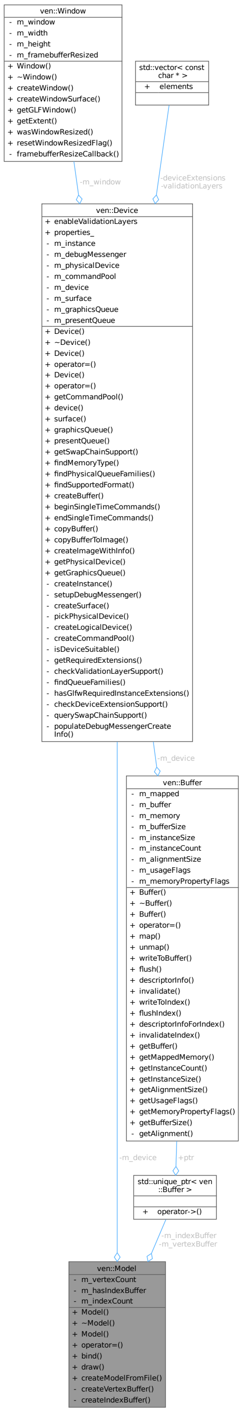
- /home/runner/work/VEngine/VEngine/include/VEngine/KeyboardController.hpp

6.20 ven::Model Class Reference

Class for model.

`#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

6.20.1 Detailed Description

Class for model.

Definition at line 21 of file Model.hpp.

6.20.2 Constructor & Destructor Documentation

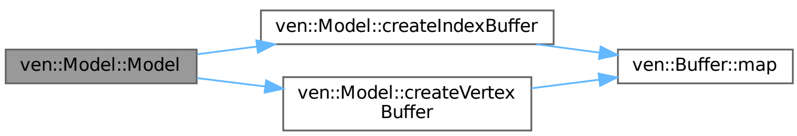
6.20.2.1 Model() [1/2]

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

Definition at line 24 of file model.cpp.

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

Here is the call graph for this function:



6.20.2.2 ∼Model()

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

6.20.2.3 Model() [2/2]

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

6.20.3 Member Function Documentation

6.20.3.1 bind()

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

Definition at line 80 of file model.cpp.

6.20.3.2 createIndexBuffer()

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

Definition at line 49 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:

6.20.3.3 createModelFromFile()

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

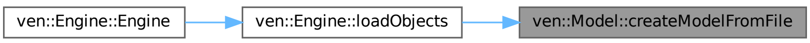
Definition at line 91 of file model.cpp.

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:



6.20.3.4 createVertexBuffer()

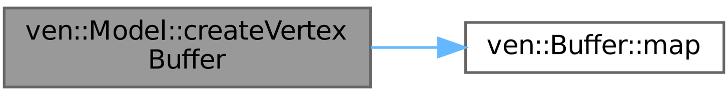
```
void ven::Model::createVertexBuffer (
            const std::vector< Vertex > & vertices)  [private]
```

Definition at line 32 of file model.cpp.

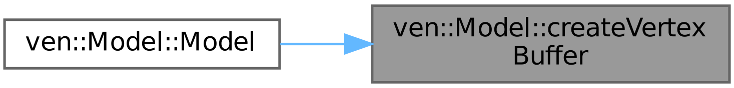
References ven::Buffer::map().

Referenced by Model().

Here is the call graph for this function:



Here is the caller graph for this function:



6.20.3.5 draw()

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

Definition at line 71 of file model.cpp.

6.20.3.6 operator=()

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

6.20.4 Member Data Documentation

6.20.4.1 m_device

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

Definition at line 62 of file Model.hpp.

6.20.4.2 m_hasIndexBuffer

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

Definition at line 66 of file Model.hpp.

6.20.4.3 m_indexBuffer

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

Definition at line 67 of file Model.hpp.

6.20.4.4 m_indexCount

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

Definition at line 68 of file Model.hpp.

6.20.4.5 m_vertexBuffer

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

Definition at line 63 of file Model.hpp.

6.20.4.6 m_vertexCount

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

Definition at line 64 of file Model.hpp.

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

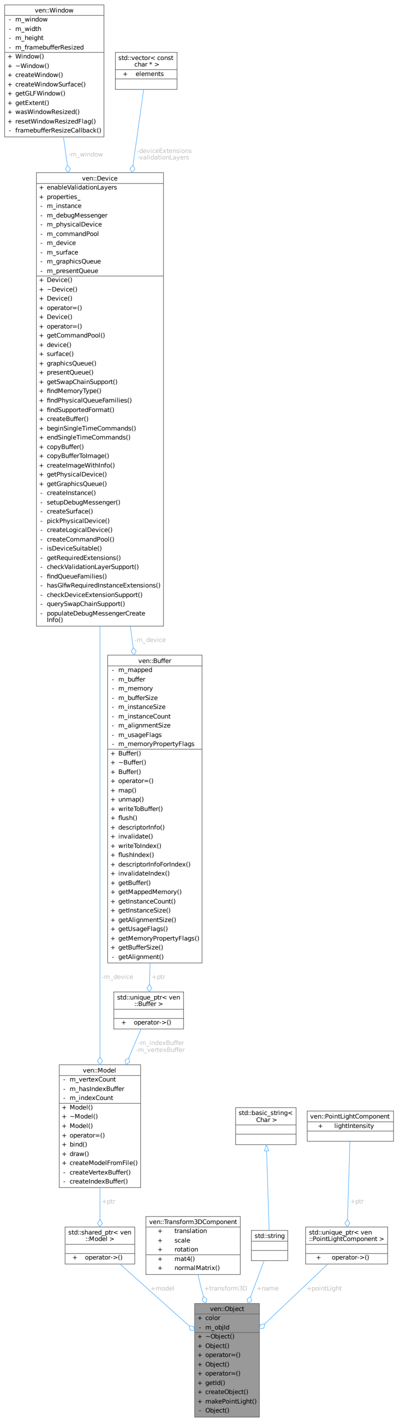
- /home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp
- /home/runner/work/VEngine/VEngine/src/model.cpp

6.21 ven::Object Class Reference

Class for object.

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

Collaboration diagram for ven::Object:



Public Types

- using Map = std::unordered_map<unsigned int, Object>

Public Member Functions

- ~Object ()=default

- Object (const Object &)=delete
- Object & operator= (const Object &)=delete
- Object (Object &&)=default
- Object & operator= (Object &&)=default
- unsigned int getId () const

Static Public Member Functions

- static Object createObject ()
- static Object makePointLight (float intensity=DEFAULT_LIGHT_INTENSITY, float radius=DEFAULT_LIGHT_RADIUS, glm::vec3 color=DEFAULT_LIGHT_COLOR)

Public Attributes

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

Private Member Functions

- Object (const unsigned int objId)

Private Attributes

- unsigned int m_objId

6.21.1 Detailed Description

Class for object.

Definition at line 40 of file Object.hpp.

6.21.2 Member Typedef Documentation

6.21.2.1 Map

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

Definition at line 44 of file Object.hpp.

6.21.3 Constructor & Destructor Documentation

6.21.3.1 ∼Object()

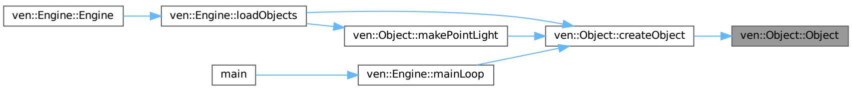
```
ven::Object::~Object () [default]
```

6.21.3.2 Object() [1/3]

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

Referenced by createObject().

Here is the caller graph for this function:



6.21.3.3 Object() [2/3]

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

6.21.3.4 Object() [3/3]

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

Definition at line 66 of file Object.hpp.

6.21.4 Member Function Documentation

6.21.4.1 createObject()

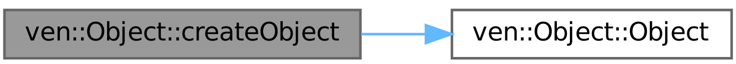
```
static Object ven::Object::createObject ()  [inline], [static]
```

Definition at line 53 of file Object.hpp.

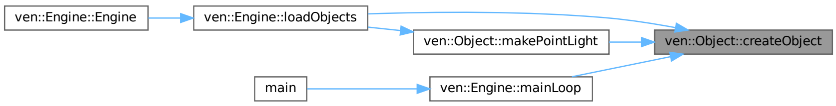
References Object().

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:



6.21.4.2 getId()

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

Definition at line 56 of file Object.hpp.

References m_objId.

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

Here is the caller graph for this function:

6.21.4.3 makePointLight()

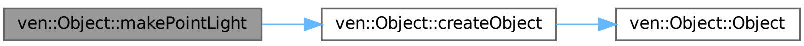
```
ven::Object ven::Object::makePointLight (
            float intensity = DEFAULT_LIGHT_INTENSITY,
            float radius = DEFAULT_LIGHT_RADIUS,
            glm::vec3 color = DEFAULT_LIGHT_COLOR)  [static]
```

Definition at line 67 of file object.cpp.

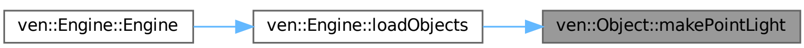
References color, createObject(), pointLight, ven::Transform3DComponent::scale, and transform3D.

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

Here is the call graph for this function:



Here is the caller graph for this function:



6.21.4.4 operator=() [1/2]

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

6.21.4.5 operator=() [2/2]

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

6.21.5 Member Data Documentation

6.21.5.1 color

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

Definition at line 59 of file Object.hpp.

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

6.21.5.2 m_objId

```
unsigned int ven::Object::m_objId  [private]
```

Definition at line 68 of file Object.hpp.

Referenced by getId().

6.21.5.3 model

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

Definition at line 58 of file Object.hpp.

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

6.21.5.4 name

```
std::string ven::Object::name {""}
```

Definition at line 61 of file Object.hpp.

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

6.21.5.5 pointLight

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

Definition at line 62 of file Object.hpp.

Referenced by makePointLight().

6.21.5.6 transform3D

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

Definition at line 60 of file Object.hpp.

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

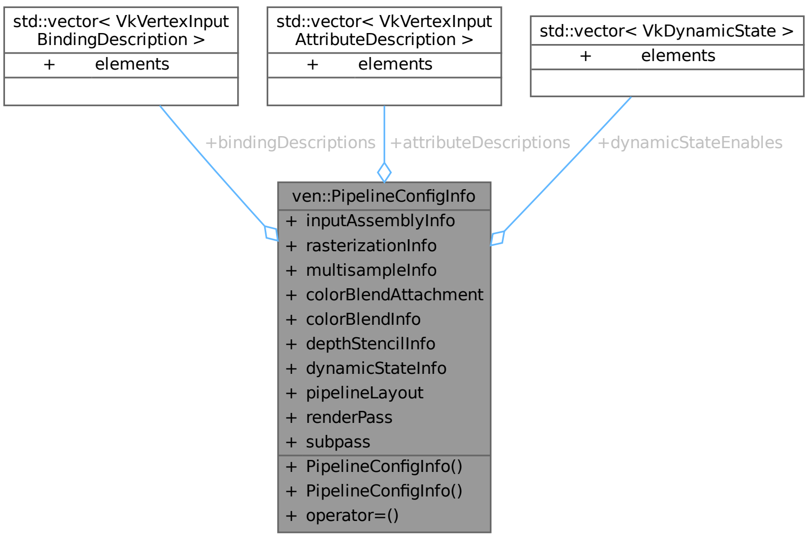
The documentation for this class was generated from the following files:

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

6.22 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

6.22.1 Detailed Description

Definition at line 21 of file Shaders.hpp.

6.22.2 Constructor & Destructor Documentation

6.22.2.1 PipelineConfigInfo() [1/2]

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

6.22.2.2 PipelineConfigInfo() [2/2]

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

6.22.3 Member Function Documentation

6.22.3.1 operator=()

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

6.22.4 Member Data Documentation

6.22.4.1 attributeDescriptions

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

Definition at line 27 of file Shaders.hpp.

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

6.22.4.2 bindingDescriptions

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

Definition at line 26 of file Shaders.hpp.

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

6.22.4.3 colorBlendAttachment

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

Definition at line 31 of file Shaders.hpp.

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

6.22.4.4 colorBlendInfo

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

Definition at line 32 of file Shaders.hpp.

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

6.22.4.5 depthStencilInfo

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

Definition at line 33 of file Shaders.hpp.

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

6.22.4.6 dynamicStateEnables

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

Definition at line 34 of file Shaders.hpp.

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

6.22.4.7 dynamicStateInfo

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

Definition at line 35 of file Shaders.hpp.

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

6.22.4.8 inputAssemblyInfo

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

Definition at line 28 of file Shaders.hpp.

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

6.22.4.9 multisampleInfo

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

Definition at line 30 of file Shaders.hpp.

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

6.22.4.10 pipelineLayout

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

Definition at line 36 of file Shaders.hpp.

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

6.22.4.11 rasterizationInfo

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

Definition at line 29 of file Shaders.hpp.

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

6.22.4.12 renderPass

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

Definition at line 37 of file Shaders.hpp.

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

6.22.4.13 subpass

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

Definition at line 38 of file Shaders.hpp.

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

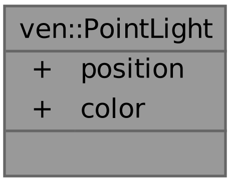
The documentation for this struct was generated from the following file:

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

6.23 ven::PointLight Struct Reference

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

Collaboration diagram for ven::PointLight:



Public Attributes

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

6.23.1 Detailed Description

Definition at line 18 of file FrameInfo.hpp.

6.23.2 Member Data Documentation

6.23.2.1 color

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

Definition at line 21 of file FrameInfo.hpp.

6.23.2.2 position

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

Definition at line 20 of file FrameInfo.hpp.

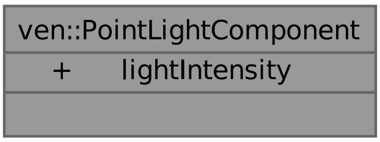
The documentation for this struct was generated from the following file:

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

6.24 ven::PointLightComponent Struct Reference

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

Collaboration diagram for ven::PointLightComponent:



Public Attributes

- float lightIntensity = DEFAULT_LIGHT_INTENSITY

6.24.1 Detailed Description

Definition at line 31 of file Object.hpp.

6.24.2 Member Data Documentation

6.24.2.1 lightIntensity

```
float ven::PointLightComponent::lightIntensity = DEFAULT_LIGHT_INTENSITY
```

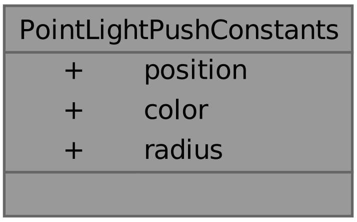
Definition at line 32 of file Object.hpp.

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

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

6.25 PointLightPushConstants Struct Reference

Collaboration diagram for PointLightPushConstants:



Public Attributes

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

6.25.1 Detailed Description

Definition at line 5 of file pointLightSystem.cpp.

6.25.2 Member Data Documentation

6.25.2.1 color

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

Definition at line 7 of file pointLightSystem.cpp.

6.25.2.2 position

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

Definition at line 6 of file pointLightSystem.cpp.

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

6.25.2.3 radius

```
float PointLightPushConstants::radius
```

Definition at line 8 of file pointLightSystem.cpp.

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

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

6.26 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}

6.26.1 Detailed Description

Class for point light system.

Definition at line 22 of file PointLightSystem.hpp.

6.26.2 Constructor & Destructor Documentation

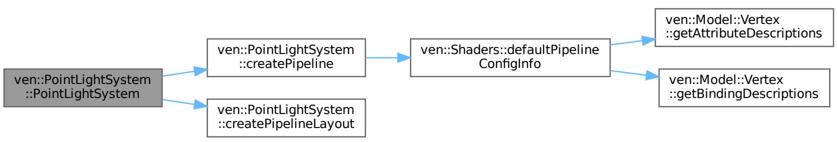
6.26.2.1 PointLightSystem() [1/2]

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

Definition at line 11 of file pointLightSystem.cpp.

References createPipeline(), and createPipelineLayout().

Here is the call graph for this function:



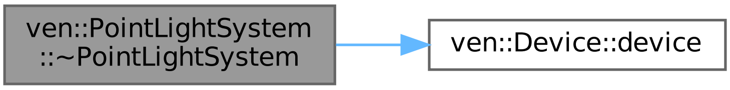
6.26.2.2 ∼PointLightSystem()

```
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:



6.26.2.3 PointLightSystem() [2/2]

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

6.26.3 Member Function Documentation

6.26.3.1 createPipeline()

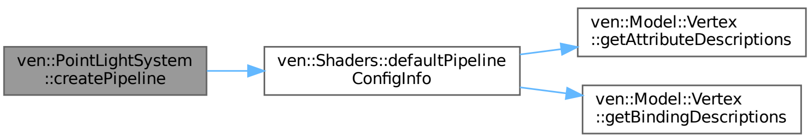
```
void ven::PointLightSystem::createPipeline (
            VkRenderPass renderPass) [private]
```

Definition at line 38 of file pointLightSystem.cpp.

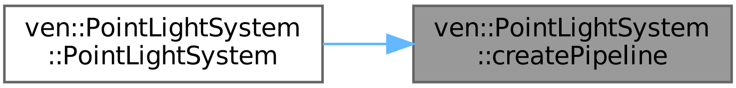
References ven::Shaders::defaultPipelineConfigInfo(), and ven::SHADERS_BIN_PATH.

Referenced by PointLightSystem().

Here is the call graph for this function:



Here is the caller graph for this function:



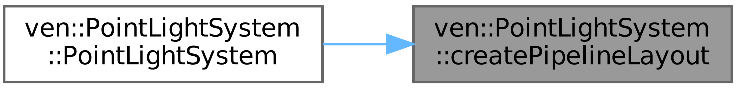
6.26.3.2 createPipelineLayout()

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

Definition at line 17 of file pointLightSystem.cpp.

Referenced by PointLightSystem().

Here is the caller graph for this function:



6.26.3.3 operator=()

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

6.26.3.4 render()

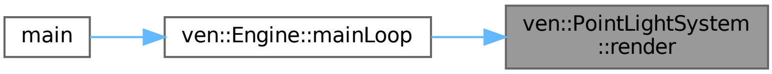
```
void ven::PointLightSystem::render (
            const FrameInfo & frameInfo) const
```

Definition at line 49 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:



6.26.3.5 update()

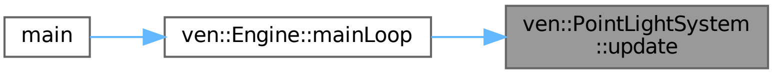
```
void ven::PointLightSystem::update (
            const FrameInfo & frameInfo,
            GlobalUbo & ubo)  [static]
```

Definition at line 69 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:



6.26.4 Member Data Documentation

6.26.4.1 m_device

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

Definition at line 40 of file PointLightSystem.hpp.

Referenced by ∼PointLightSystem().

6.26.4.2 m_pipelineLayout

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

Definition at line 43 of file PointLightSystem.hpp.

Referenced by ∼PointLightSystem().

6.26.4.3 m_shaders

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

Definition at line 42 of file PointLightSystem.hpp.

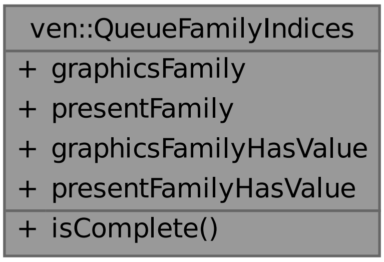
The documentation for this class was generated from the following files:

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

6.27 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

6.27.1 Detailed Description

Definition at line 21 of file Device.hpp.

6.27.2 Member Function Documentation

6.27.2.1 isComplete()

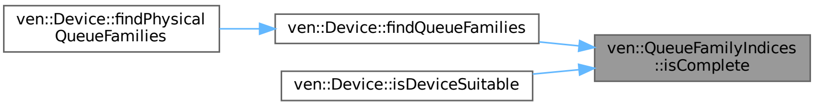
```
bool ven::QueueFamilyIndices::isComplete () const  [inline], [nodiscard]
```

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:



6.27.3 Member Data Documentation

6.27.3.1 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().

6.27.3.2 graphicsFamilyHasValue

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

Definition at line 24 of file Device.hpp.

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

6.27.3.3 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().

6.27.3.4 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:

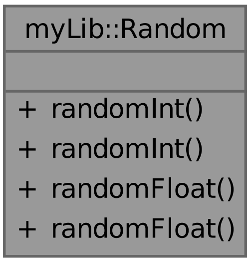
- /home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp

6.28 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 ()

6.28.1 Detailed Description

Class for random number generation.

Definition at line 21 of file Random.hpp.

6.28.2 Member Function Documentation

6.28.2.1 randomFloat() [1/2]

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

Definition at line 40 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:

6.28.2.2 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.

References myLib::RANDOM_FLOAT_MAX, myLib::RANDOM_INT_MAX, and myLib::RANDOM_INT_MIN.

6.28.2.3 randomInt() [1/2]

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

Definition at line 32 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:

6.28.2.4 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.

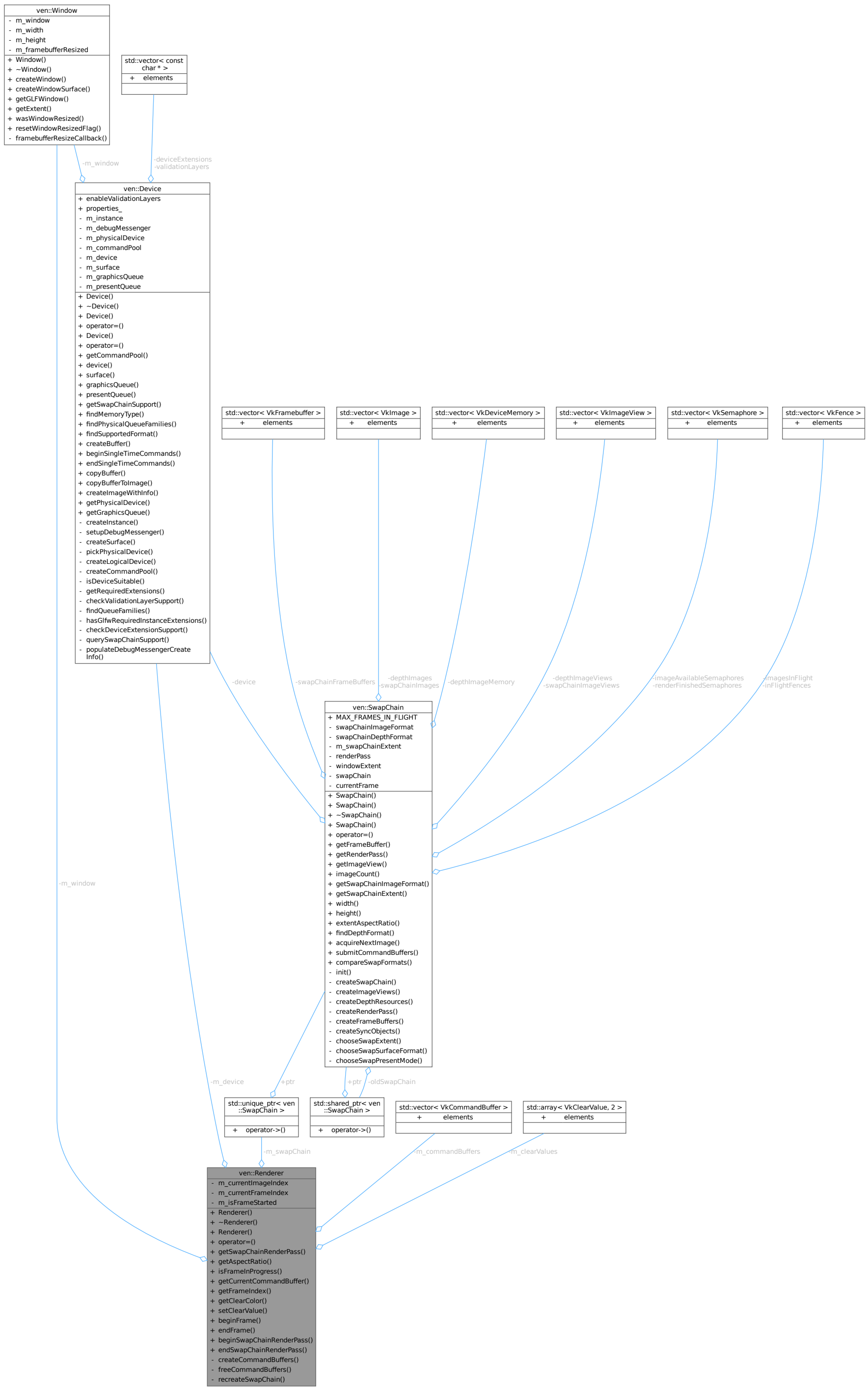
The documentation for this class was generated from the following files:

- /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

6.29 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
- std::array< float, 4 > getClearColor () const
- void setClearValue (VkClearColorValue clearColorValue=DEFAULT_CLEAR_COLOR, VkClearDepth↩ StencilValue clearDepthValue=DEFAULT_CLEAR_DEPTH)
- VkCommandBuffer beginFrame ()
- void endFrame ()
- void beginSwapChainRenderPass (VkCommandBuffer commandBuffer)
- 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
- std::array< VkClearValue, 2 > m_clearValues
- uint32_t m_currentImageIndex {0}
- int m_currentFrameIndex {0}
- bool m_isFrameStarted {false}

6.29.1 Detailed Description

Definition at line 23 of file Renderer.hpp.

6.29.2 Constructor & Destructor Documentation

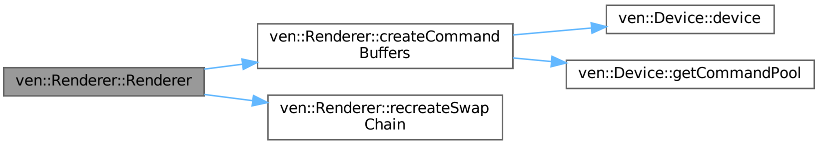
6.29.2.1 Renderer() [1/2]

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

Definition at line 27 of file Renderer.hpp.

References createCommandBuffers(), and recreateSwapChain().

Here is the call graph for this function:



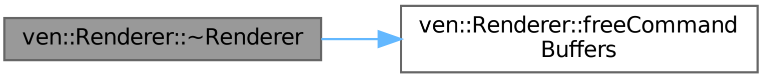
6.29.2.2 ∼Renderer()

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

Definition at line 28 of file Renderer.hpp.

References freeCommandBuffers().

Here is the call graph for this function:



6.29.2.3 Renderer() [2/2]

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

6.29.3 Member Function Documentation

6.29.3.1 beginFrame()

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

Definition at line 43 of file renderer.cpp.

6.29.3.2 beginSwapChainRenderPass()

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

Definition at line 90 of file renderer.cpp.

6.29.3.3 createCommandBuffers()

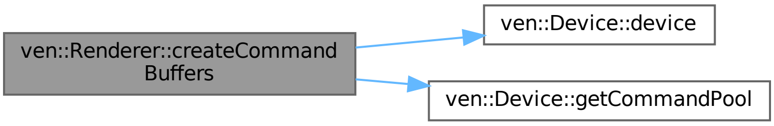
```
void ven::Renderer::createCommandBuffers ()  [private]
```

Definition at line 3 of file renderer.cpp.

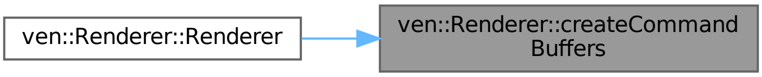
References ven::Device::device(), ven::Device::getCommandPool(), m_commandBuffers, m_device, and ven::SwapChain::MAX_FRAMES_IN_FLIGHT.

Referenced by Renderer().

Here is the call graph for this function:



Here is the caller graph for this function:



6.29.3.4 endFrame()

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

Definition at line 69 of file renderer.cpp.

References ven::SwapChain::MAX_FRAMES_IN_FLIGHT.

6.29.3.5 endSwapChainRenderPass()

```
void ven::Renderer::endSwapChainRenderPass (
            VkCommandBuffer commandBuffer)
```

Definition at line 120 of file renderer.cpp.

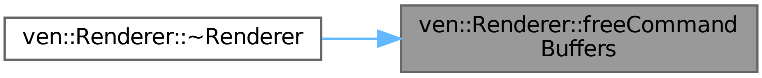
6.29.3.6 freeCommandBuffers()

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

Definition at line 17 of file renderer.cpp.

Referenced by ∼Renderer().

Here is the caller graph for this function:



6.29.3.7 getAspectRatio()

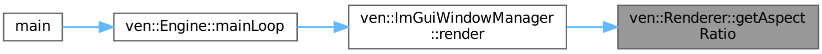
```
float ven::Renderer::getAspectRatio ( ) const [inline], [nodiscard]
```

Definition at line 34 of file Renderer.hpp.

References m_swapChain.

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

Here is the caller graph for this function:



6.29.3.8 getClearColor()

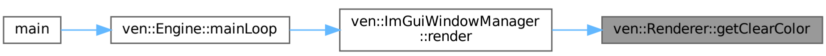
```
std::array< float, 4 > ven::Renderer::getClearColor ( ) const [inline], [nodiscard]
```

Definition at line 39 of file Renderer.hpp.

References m_clearValues.

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

Here is the caller graph for this function:



6.29.3.9 getCurrentCommandBuffer()

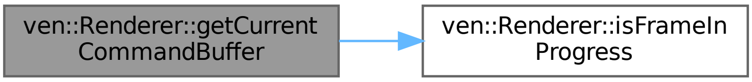
```
VkCommandBuffer ven::Renderer::getCurrentCommandBuffer () const  [inline], [nodiscard]
```

Definition at line 36 of file Renderer.hpp.

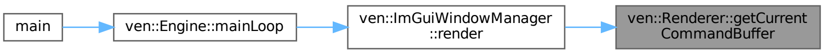
References isFrameInProgress(), m_commandBuffers, and m_currentFrameIndex.

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

Here is the call graph for this function:



Here is the caller graph for this function:



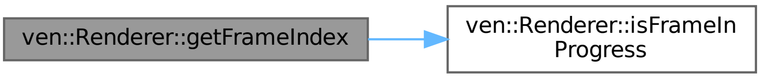
6.29.3.10 getFrameIndex()

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

Definition at line 38 of file Renderer.hpp.

References isFrameInProgress(), and m_currentFrameIndex.

Here is the call graph for this function:



6.29.3.11 getSwapChainRenderPass()

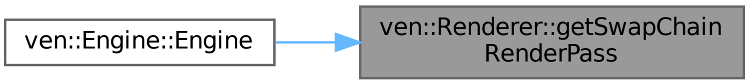
```
VkRenderPass ven::Renderer::getSwapChainRenderPass () const [inline], [nodiscard]
```

Definition at line 33 of file Renderer.hpp.

References m_swapChain.

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

Here is the caller graph for this function:



6.29.3.12 isFrameInProgress()

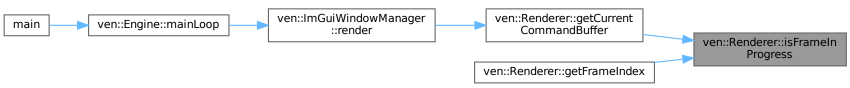
```
bool ven::Renderer::isFrameInProgress () const [inline], [nodiscard]
```

Definition at line 35 of file Renderer.hpp.

References m_isFrameStarted.

Referenced by getCurrentCommandBuffer(), and getFrameIndex().

Here is the caller graph for this function:



6.29.3.13 operator=()

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

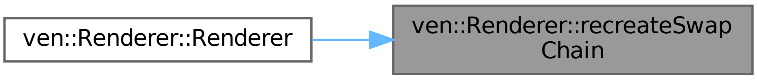
6.29.3.14 recreateSwapChain()

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

Definition at line 23 of file renderer.cpp.

Referenced by Renderer().

Here is the caller graph for this function:



6.29.3.15 setClearValue()

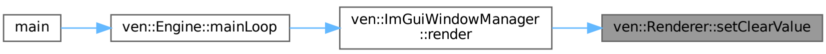
```
void ven::Renderer::setClearValue (
            VkClearColorValue clearColorValue = DEFAULT_CLEAR_COLOR,
            VkClearDepthStencilValue clearDepthValue = DEFAULT_CLEAR_DEPTH)  [inline]
```

Definition at line 46 of file Renderer.hpp.

References m_clearValues.

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

Here is the caller graph for this function:



6.29.4 Member Data Documentation

6.29.4.1 m_clearValues

```
std::array<VkClearValue, 2> ven::Renderer::m_clearValues  [private]
```

Definition at line 62 of file Renderer.hpp.

Referenced by getClearColor(), and setClearValue().

6.29.4.2 m_commandBuffers

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

Definition at line 61 of file Renderer.hpp.

Referenced by createCommandBuffers(), and getCurrentCommandBuffer().

6.29.4.3 m_currentFrameIndex

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

Definition at line 65 of file Renderer.hpp.

Referenced by getCurrentCommandBuffer(), and getFrameIndex().

6.29.4.4 m_currentImageIndex

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

Definition at line 64 of file Renderer.hpp.

6.29.4.5 m_device

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

Definition at line 59 of file Renderer.hpp.

Referenced by createCommandBuffers().

6.29.4.6 m_isFrameStarted

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

Definition at line 66 of file Renderer.hpp.

Referenced by isFrameInProgress().

6.29.4.7 m_swapChain

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

Definition at line 60 of file Renderer.hpp.

Referenced by getAspectRatio(), and getSwapChainRenderPass().

6.29.4.8 m_window

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

Definition at line 58 of file Renderer.hpp.

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

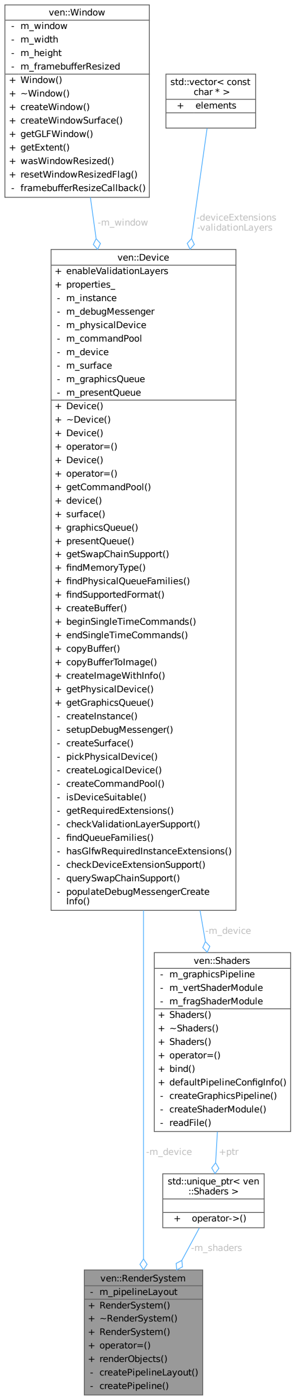
- /home/runner/work/VEngine/VEngine/include/VEngine/Renderer.hpp
- /home/runner/work/VEngine/VEngine/src/renderer.cpp

6.30 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}

6.30.1 Detailed Description

Class for render system.

Definition at line 29 of file RenderSystem.hpp.

6.30.2 Constructor & Destructor Documentation

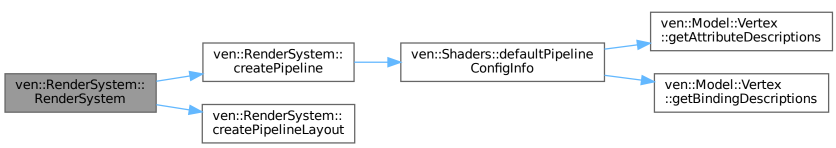
6.30.2.1 RenderSystem() [1/2]

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

Definition at line 3 of file renderSystem.cpp.

References createPipeline(), and createPipelineLayout().

Here is the call graph for this function:



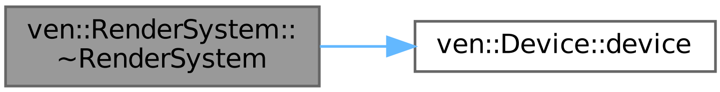
6.30.2.2 ∼RenderSystem()

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

Definition at line 34 of file RenderSystem.hpp.

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

Here is the call graph for this function:



6.30.2.3 RenderSystem() [2/2]

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

6.30.3 Member Function Documentation

6.30.3.1 createPipeline()

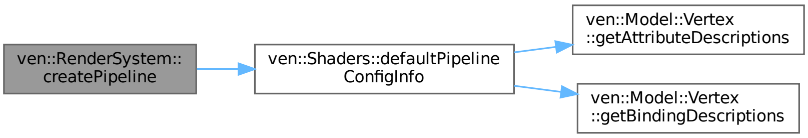
```
void ven::RenderSystem::createPipeline (
            VkRenderPass renderPass) [private]
```

Definition at line 30 of file renderSystem.cpp.

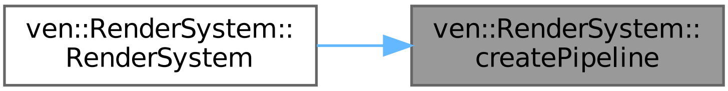
References ven::Shaders::defaultPipelineConfigInfo(), and ven::SHADERS_BIN_PATH.

Referenced by RenderSystem().

Here is the call graph for this function:



Here is the caller graph for this function:



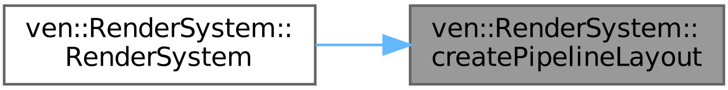
6.30.3.2 createPipelineLayout()

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

Definition at line 9 of file renderSystem.cpp.

Referenced by RenderSystem().

Here is the caller graph for this function:



6.30.3.3 operator=()

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

6.30.3.4 renderObjects()

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

Definition at line 39 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:

6.30.4 Member Data Documentation

6.30.4.1 m_device

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

Definition at line 46 of file RenderSystem.hpp.

Referenced by ∼RenderSystem().

6.30.4.2 m_pipelineLayout

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

Definition at line 48 of file RenderSystem.hpp.

Referenced by ∼RenderSystem().

6.30.4.3 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:

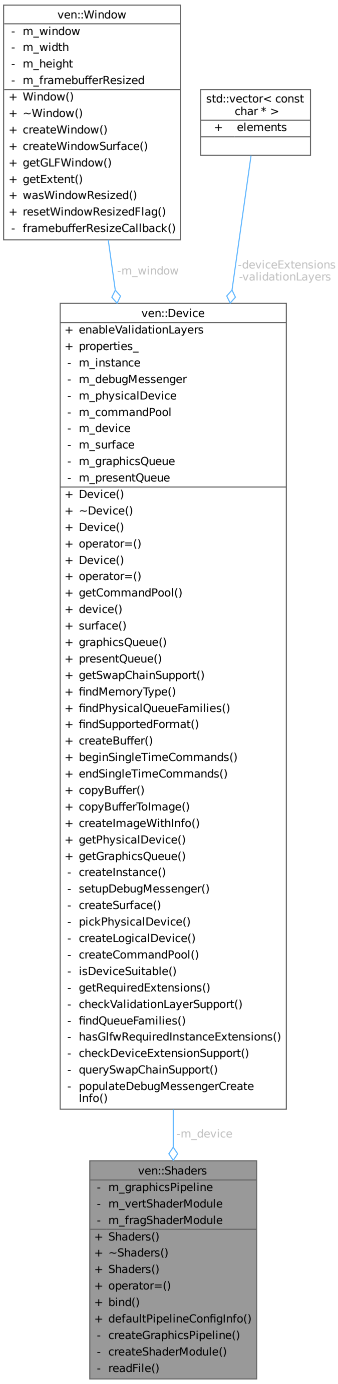
- /home/runner/work/VEngine/VEngine/include/VEngine/System/RenderSystem.hpp
- /home/runner/work/VEngine/VEngine/src/system/renderSystem.cpp

6.31 ven::Shaders Class Reference

Class for shaders.

`#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}

6.31.1 Detailed Description

Class for shaders.

Definition at line 46 of file Shaders.hpp.

6.31.2 Constructor & Destructor Documentation

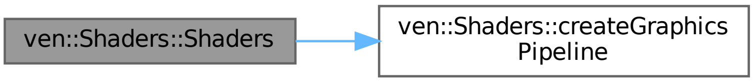
6.31.2.1 Shaders() [1/2]

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

Definition at line 50 of file Shaders.hpp.

References createGraphicsPipeline().

Here is the call graph for this function:



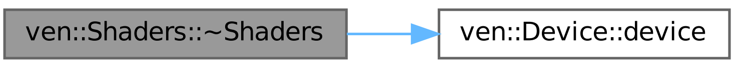
6.31.2.2 ∼Shaders()

```
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:



6.31.2.3 Shaders() [2/2]

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

6.31.3 Member Function Documentation

6.31.3.1 bind()

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

Definition at line 57 of file Shaders.hpp.

References m_graphicsPipeline.

6.31.3.2 createGraphicsPipeline()

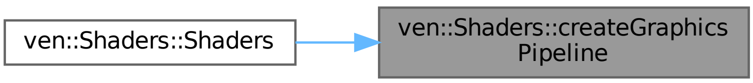
```
void ven::Shaders::createGraphicsPipeline (
            const std::string & vertFilepath,
            const std::string & fragFilepath,
            const PipelineConfigInfo & configInfo) [private]
```

Definition at line 31 of file shaders.cpp.

References ven::PipelineConfigInfo::attributeDescriptions, ven::PipelineConfigInfo::bindingDescriptions, ven::PipelineConfigInfo::colo
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:



6.31.3.3 createShaderModule()

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

Definition at line 100 of file shaders.cpp.

6.31.3.4 defaultPipelineConfigInfo()

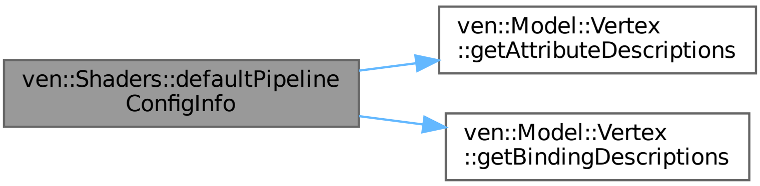
```
void ven::Shaders::defaultPipelineConfigInfo (
            PipelineConfigInfo & configInfo)  [static]
```

Definition at line 112 of file shaders.cpp.

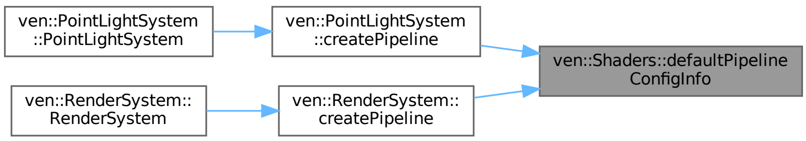
References ven::PipelineConfigInfo::attributeDescriptions, ven::PipelineConfigInfo::bindingDescriptions, ven::PipelineConfigInfo::colo
ven::PipelineConfigInfo::colorBlendInfo, ven::PipelineConfigInfo::depthStencilInfo, ven::PipelineConfigInfo::dynamicStateEnables,
ven::PipelineConfigInfo::dynamicStateInfo, 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:



6.31.3.5 operator=()

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

6.31.3.6 readFile()

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

Definition at line 13 of file shaders.cpp.

6.31.4 Member Data Documentation

6.31.4.1 m_device

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

Definition at line 65 of file Shaders.hpp.

Referenced by ∼Shaders().

6.31.4.2 m_fragShaderModule

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

Definition at line 68 of file Shaders.hpp.

Referenced by ∼Shaders().

6.31.4.3 m_graphicsPipeline

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

Definition at line 66 of file Shaders.hpp.

Referenced by bind(), and ∼Shaders().

6.31.4.4 m_vertShaderModule

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

Definition at line 67 of file Shaders.hpp.

Referenced by ∼Shaders().

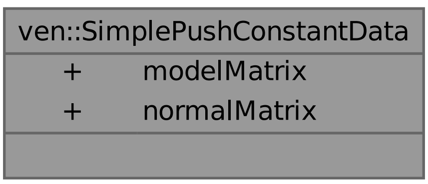
The documentation for this class was generated from the following files:

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

6.32 ven::SimplePushConstantData Struct Reference

`#include <RenderSystem.hpp>`

Collaboration diagram for ven::SimplePushConstantData:



Public Attributes

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

6.32.1 Detailed Description

Definition at line 19 of file RenderSystem.hpp.

6.32.2 Member Data Documentation

6.32.2.1 modelMatrix

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

Definition at line 20 of file RenderSystem.hpp.

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

6.32.2.2 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:

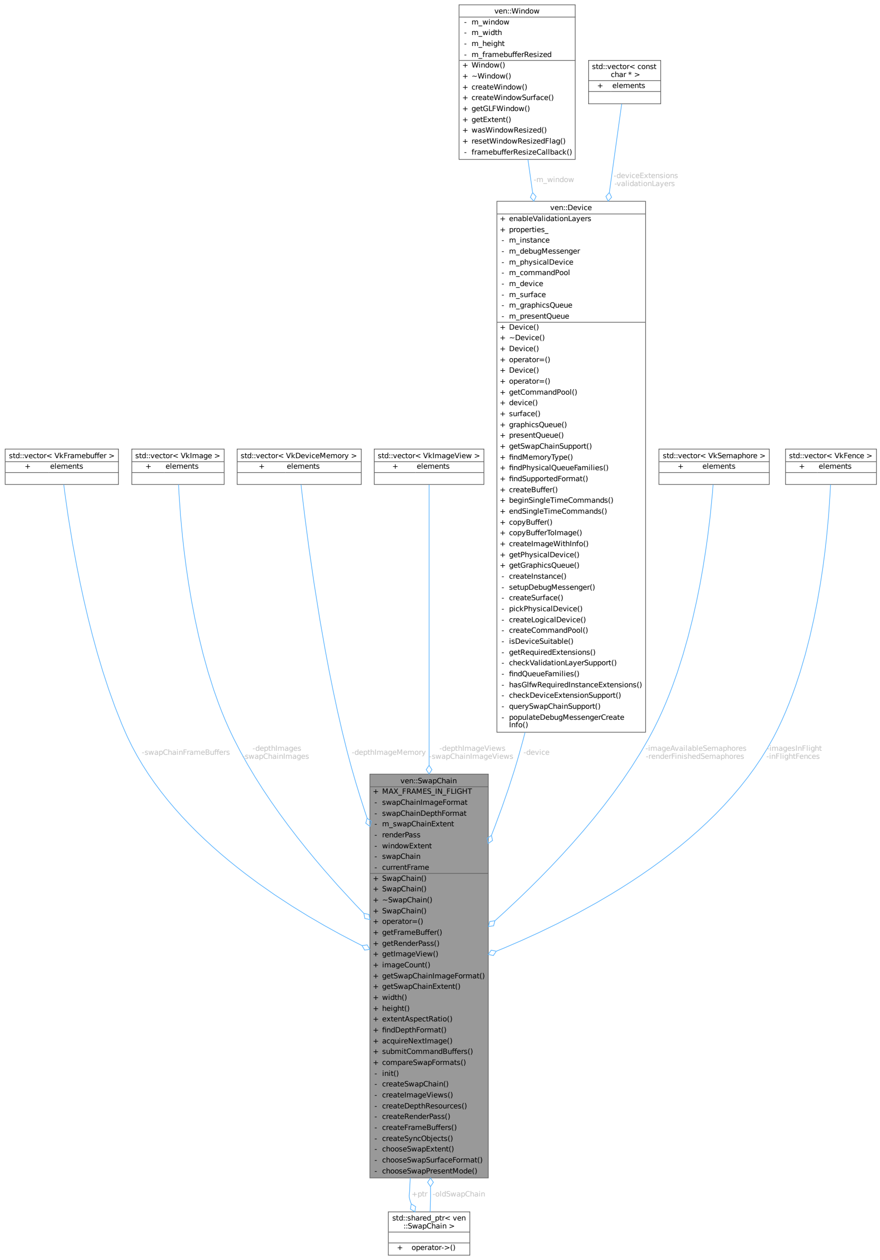
- /home/runner/work/VEngine/VEngine/include/VEngine/System/RenderSystem.hpp

6.33 ven::SwapChain Class Reference

Class for swap chain.

```
#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 > depthImageMemory
- 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

6.33.1 Detailed Description

Class for swap chain.

Definition at line 21 of file SwapChain.hpp.

6.33.2 Constructor & Destructor Documentation

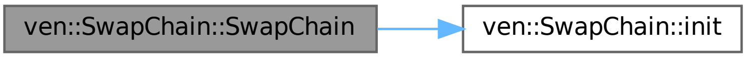
6.33.2.1 SwapChain() [1/3]

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

Definition at line 27 of file SwapChain.hpp.

References init().

Here is the call graph for this function:



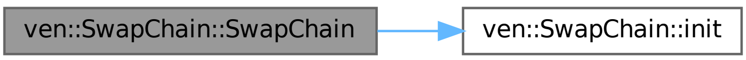
6.33.2.2 SwapChain() [2/3]

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

Definition at line 28 of file SwapChain.hpp.

References init(), and oldSwapChain.

Here is the call graph for this function:



6.33.2.3 ∼SwapChain()

```
ven::SwapChain::∼SwapChain ()
```

Definition at line 7 of file swapChain.cpp.

References depthImageMemory, 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:

6.33.2.4 SwapChain() [3/3]

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

6.33.3 Member Function Documentation

6.33.3.1 acquireNextImage()

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

Definition at line 49 of file swapChain.cpp.

6.33.3.2 chooseSwapExtent()

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

Definition at line 362 of file swapChain.cpp.

6.33.3.3 chooseSwapPresentMode()

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

Definition at line 342 of file swapChain.cpp.

6.33.3.4 chooseSwapSurfaceFormat()

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

Definition at line 331 of file swapChain.cpp.

6.33.3.5 compareSwapFormats()

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

Definition at line 49 of file SwapChain.hpp.

References swapChainDepthFormat, and swapChainImageFormat.

6.33.3.6 createDepthResources()

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

Definition at line 262 of file swapChain.cpp.

6.33.3.7 createFrameBuffers()

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

Definition at line 240 of file swapChain.cpp.

6.33.3.8 createImageViews()

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

Definition at line 160 of file swapChain.cpp.

6.33.3.9 createRenderPass()

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

Definition at line 181 of file swapChain.cpp.

6.33.3.10 createSwapChain()

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

Definition at line 103 of file swapChain.cpp.

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

6.33.3.11 createSyncObjects()

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

Definition at line 308 of file swapChain.cpp.

6.33.3.12 extentAspectRatio()

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

Definition at line 43 of file SwapChain.hpp.

References m_swapChainExtent.

6.33.3.13 findDepthFormat()

```
VkFormat ven::SwapChain::findDepthFormat () const  [nodiscard]
```

Definition at line 374 of file swapChain.cpp.

6.33.3.14 getFrameBuffer()

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

Definition at line 34 of file SwapChain.hpp.

References swapChainFrameBuffers.

6.33.3.15 getImageView()

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

Definition at line 36 of file SwapChain.hpp.

References swapChainImageViews.

6.33.3.16 getRenderPass()

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

Definition at line 35 of file SwapChain.hpp.

References renderPass.

6.33.3.17 getSwapChainExtent()

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

Definition at line 39 of file SwapChain.hpp.

References m_swapChainExtent.

6.33.3.18 getSwapChainImageFormat()

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

Definition at line 38 of file SwapChain.hpp.

References swapChainImageFormat.

6.33.3.19 height()

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

Definition at line 41 of file SwapChain.hpp.

References m_swapChainExtent.

6.33.3.20 imageCount()

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

Definition at line 37 of file SwapChain.hpp.

References swapChainImages.

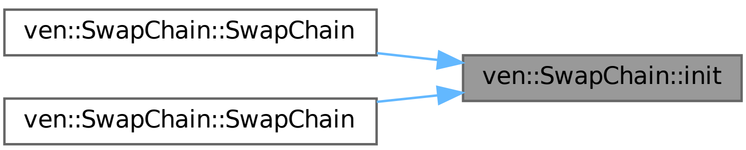
6.33.3.21 init()

`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:



6.33.3.22 operator=()

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

6.33.3.23 submitCommandBuffers()

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

Definition at line 56 of file swapChain.cpp.

6.33.3.24 width()

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

Definition at line 40 of file SwapChain.hpp.

References m_swapChainExtent.

6.33.4 Member Data Documentation

6.33.4.1 currentFrame

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

Definition at line 90 of file SwapChain.hpp.

6.33.4.2 depthImageMemory

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

Definition at line 75 of file SwapChain.hpp.

Referenced by ~SwapChain().

6.33.4.3 depthImages

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

Definition at line 74 of file SwapChain.hpp.

Referenced by ~SwapChain().

6.33.4.4 depthImageViews

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

Definition at line 76 of file SwapChain.hpp.

Referenced by ~SwapChain().

6.33.4.5 device

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

Definition at line 80 of file SwapChain.hpp.

Referenced by ~SwapChain().

6.33.4.6 imageAvailableSemaphores

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

Definition at line 86 of file SwapChain.hpp.

Referenced by ~SwapChain().

6.33.4.7 imagesInFlight

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

Definition at line 89 of file SwapChain.hpp.

6.33.4.8 inFlightFences

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

Definition at line 88 of file SwapChain.hpp.

Referenced by ~SwapChain().

6.33.4.9 m_swapChainExtent

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

Definition at line 69 of file SwapChain.hpp.

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

6.33.4.10 MAX_FRAMES_IN_FLIGHT

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

Definition at line 25 of file SwapChain.hpp.

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

6.33.4.11 oldSwapChain

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

Definition at line 84 of file SwapChain.hpp.

Referenced by SwapChain().

6.33.4.12 renderFinishedSemaphores

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

Definition at line 87 of file SwapChain.hpp.

Referenced by ∼SwapChain().

6.33.4.13 renderPass

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

Definition at line 72 of file SwapChain.hpp.

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

6.33.4.14 swapChain

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

Definition at line 83 of file SwapChain.hpp.

Referenced by ∼SwapChain().

6.33.4.15 swapChainDepthFormat

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

Definition at line 68 of file SwapChain.hpp.

Referenced by compareSwapFormats().

6.33.4.16 swapChainFrameBuffers

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

Definition at line 71 of file SwapChain.hpp.

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

6.33.4.17 swapChainImageFormat

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

Definition at line 67 of file SwapChain.hpp.

Referenced by compareSwapFormats(), and getSwapChainImageFormat().

6.33.4.18 swapChainImages

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

Definition at line 77 of file SwapChain.hpp.

Referenced by imageCount().

6.33.4.19 swapChainImageViews

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

Definition at line 78 of file SwapChain.hpp.

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

6.33.4.20 windowExtent

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

Definition at line 81 of file SwapChain.hpp.

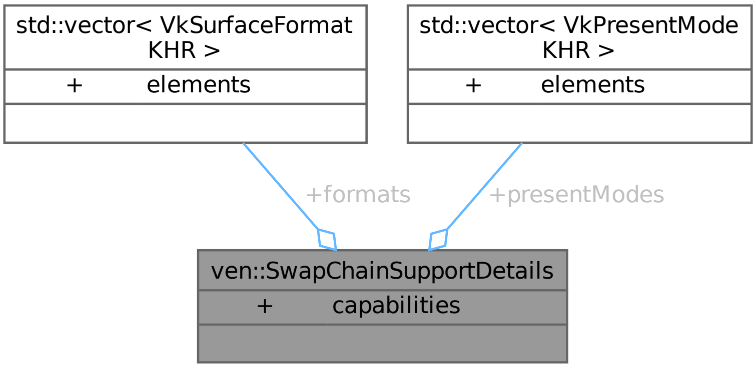
The documentation for this class was generated from the following files:

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

6.34 ven::SwapChainSupportDetails Struct Reference

`#include <Device.hpp>`

Collaboration diagram for ven::SwapChainSupportDetails:



Public Attributes

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

6.34.1 Detailed Description

Definition at line 15 of file Device.hpp.

6.34.2 Member Data Documentation

6.34.2.1 capabilities

```
VkSurfaceCapabilitiesKHR ven::SwapChainSupportDetails::capabilities
```

Definition at line 16 of file Device.hpp.

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

6.34.2.2 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().

6.34.2.3 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:

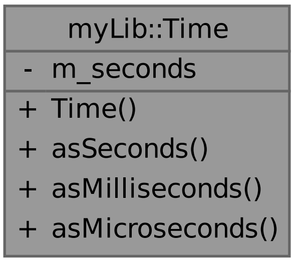
- /home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp

6.35 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}

6.35.1 Detailed Description

Class used for time management.

Definition at line 18 of file Time.hpp.

6.35.2 Constructor & Destructor Documentation

6.35.2.1 Time()

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

Construct a new Time object.

Definition at line 25 of file Time.hpp.

6.35.3 Member Function Documentation

6.35.3.1 asMicroseconds()

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

Transform the time to microseconds.

Returns

int The time in microseconds

Definition at line 43 of file Time.hpp.

References m_seconds, and myLib::MICROSECONDS_PER_SECOND.

6.35.3.2 asMilliseconds()

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

Transform the time to milliseconds.

Returns

int The time in milliseconds

Definition at line 37 of file Time.hpp.

References m_seconds, and myLib::MILLISECONDS_PER_SECOND.

6.35.3.3 asSeconds()

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

Transform the time to seconds.

Returns

int The time in seconds

Definition at line 31 of file Time.hpp.

References m_seconds.

6.35.4 Member Data Documentation

6.35.4.1 m_seconds

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

Definition at line 50 of file Time.hpp.

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

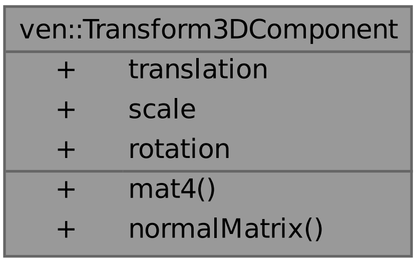
The documentation for this class was generated from the following file:

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

6.36 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 {}

6.36.1 Detailed Description

Definition at line 22 of file Object.hpp.

6.36.2 Member Function Documentation

6.36.2.1 mat4()

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

Definition at line 3 of file object.cpp.

References rotation, scale, and translation.

6.36.2.2 normalMatrix()

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

Definition at line 38 of file object.cpp.

6.36.3 Member Data Documentation

6.36.3.1 rotation

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

Definition at line 25 of file Object.hpp.

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

6.36.3.2 scale

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

Definition at line 24 of file Object.hpp.

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

6.36.3.3 translation

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

Definition at line 23 of file Object.hpp.

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

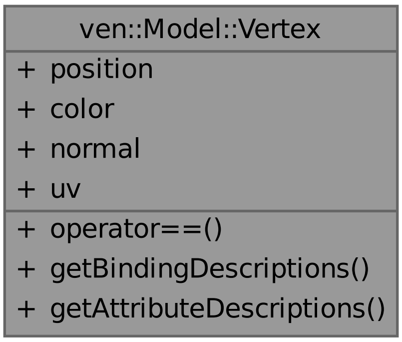
The documentation for this struct was generated from the following files:

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

6.37 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 {}

6.37.1 Detailed Description

Definition at line 25 of file Model.hpp.

6.37.2 Member Function Documentation

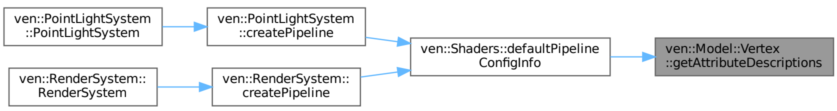
6.37.2.1 getAttributeDescriptions()

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

Definition at line 107 of file model.cpp.

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

Here is the caller graph for this function:



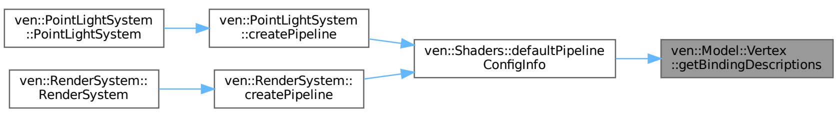
6.37.2.2 getBindingDescriptions()

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

Definition at line 98 of file model.cpp.

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

Here is the caller graph for this function:



6.37.2.3 operator==()

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

Definition at line 34 of file Model.hpp.

References color, normal, position, and uv.

6.37.3 Member Data Documentation

6.37.3.1 color

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

Definition at line 27 of file Model.hpp.

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

6.37.3.2 normal

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

Definition at line 28 of file Model.hpp.

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

6.37.3.3 position

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

Definition at line 26 of file Model.hpp.

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

6.37.3.4 uv

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

Definition at line 29 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:

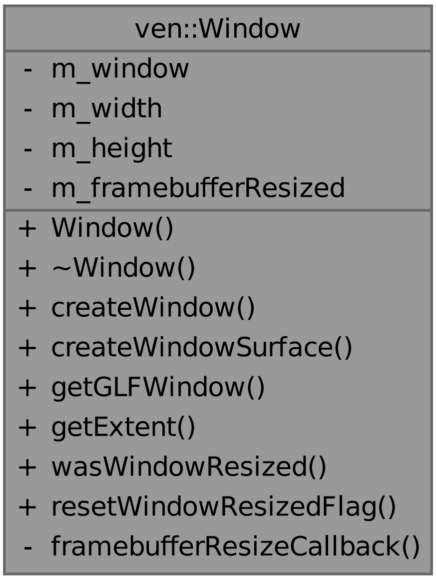
- /home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp
- /home/runner/work/VEngine/VEngine/src/model.cpp

6.38 ven::Window Class Reference

Class for window.

`#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

6.38.1 Detailed Description

Class for window.

Definition at line 26 of file Window.hpp.

6.38.2 Constructor & Destructor Documentation

6.38.2.1 Window()

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

Definition at line 30 of file Window.hpp.

6.38.2.2 ∼Window()

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

Definition at line 31 of file Window.hpp.

References m_window.

6.38.3 Member Function Documentation

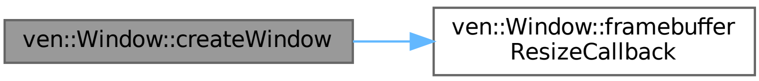
6.38.3.1 createWindow()

```
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:



6.38.3.2 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:

6.38.3.3 framebufferResizeCallback()

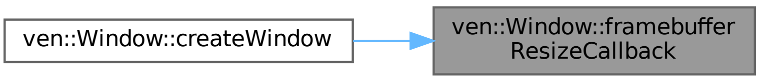
```
void ven::Window::framebufferResizeCallback (
            GLFWwindow * window,
            int width,
            int height) [static], [private]
```

Definition at line 31 of file window.cpp.

References m_framebufferResized.

Referenced by createWindow().

Here is the caller graph for this function:



6.38.3.4 getExtent()

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

Definition at line 38 of file Window.hpp.

References m_height, and m_width.

6.38.3.5 getGLFWindow()

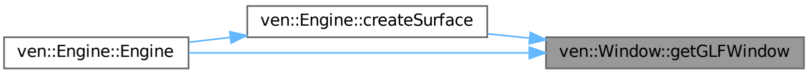
```
GLFWwindow * ven::Window::getGLFWindow () const  [inline], [nodiscard]
```

Definition at line 36 of file Window.hpp.

References m_window.

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

Here is the caller graph for this function:



6.38.3.6 resetWindowResizedFlag()

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

Definition at line 40 of file Window.hpp.

References m_framebufferResized.

6.38.3.7 wasWindowResized()

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

Definition at line 39 of file Window.hpp.

References m_framebufferResized.

6.38.4 Member Data Documentation

6.38.4.1 m_framebufferResized

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

Definition at line 50 of file Window.hpp.

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

6.38.4.2 m_height

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

Definition at line 48 of file Window.hpp.

Referenced by getExtent().

6.38.4.3 m_width

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

Definition at line 47 of file Window.hpp.

Referenced by getExtent().

6.38.4.4 m_window

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

Definition at line 46 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

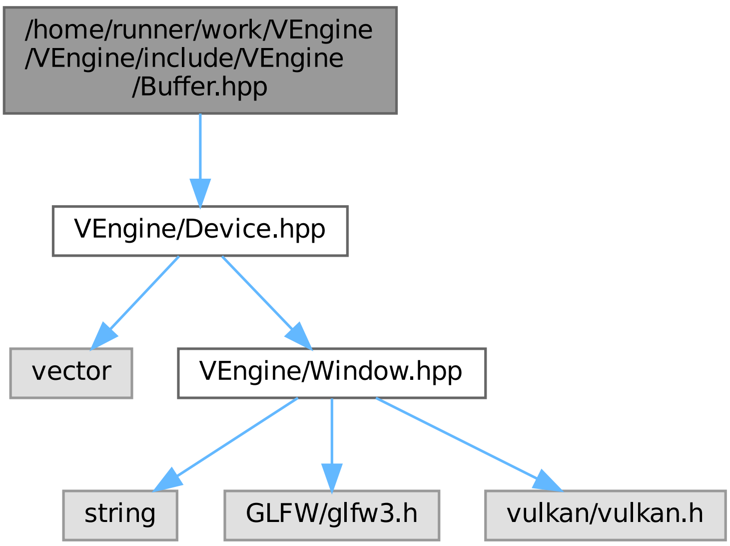
Chapter 7

File Documentation

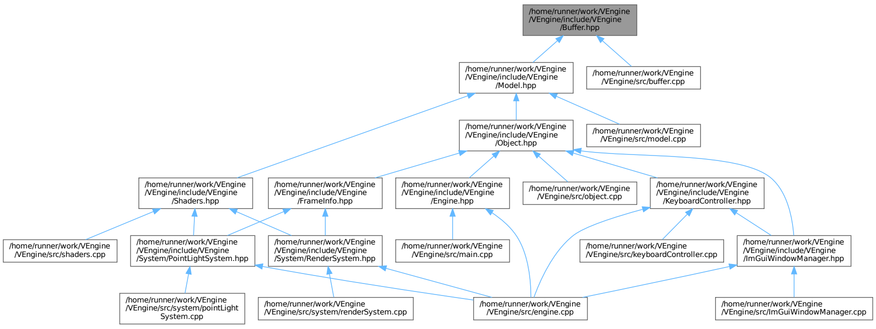
7.1 /home/runner/work/VEngine/VEngine/include/VEngine/Buffer.hpp File Reference

This file contains the Buffer class.

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



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



Classes

- class ven::Buffer

    Class for buffer.

Namespaces

- namespace ven

7.1.1 Detailed Description

This file contains the Buffer class.

Definition in file Buffer.hpp.

7.2 Buffer.hpp

Go to the documentation of this file.

```
///
/// @file Buffer.hpp
/// @brief This file contains the Buffer class
/// @namespace ven
///

#pragma once

#include "VEngine/Device.hpp"

namespace ven {

    ///
    /// @class Buffer
    /// @brief Class for buffer
    /// @namespace ven
    ///
    class Buffer {

        public:

            Buffer(Device& device, VkDeviceSize instanceSize, uint32_t instanceCount,
VkBufferUsageFlags usageFlags, VkMemoryPropertyFlags memoryPropertyFlags, VkDeviceSize
minOffsetAlignment = 1);
            ~Buffer();

```

```
            Buffer(const Buffer&) = delete;
            Buffer& operator=(const Buffer&) = delete;

            ///
            /// @brief Map a memory range of this buffer. If successful, mapped points to the specified buffer range.
            ///
            /// @param size (Optional) Size of the memory range to map. Pass VK_WHOLE_SIZE to map the complete buffer range.
            /// @param offset (Optional) Byte offset from beginning
            ///
            /// @return VkResult of the buffer mapping call
            ///
            VkResult map(VkDeviceSize size = VK_WHOLE_SIZE, VkDeviceSize offset = 0);

            ///
            /// @brief Unmap a mapped memory range
            ///
            /// @note Does not return a result as vkUnmapMemory can't fail
            ///
            void unmap();

            ///
            /// @brief Copies the specified data to the mapped buffer. Default value writes whole buffer range
            ///
            /// @param data Pointer to the data to copy
            /// @param size (Optional) Size of the data to copy. Pass VK_WHOLE_SIZE to flush the complete buffer range.
            /// @param offset (Optional) Byte offset from beginning of mapped region
            ///
            void writeToBuffer(const void* data, VkDeviceSize size = VK_WHOLE_SIZE, VkDeviceSize offset = 0) const;

            ///
            /// @brief Flush a memory range of the buffer to make it visible to the device
            ///
            ///  @note Only required for non-coherent memory
            ///
            /// @param size (Optional) Size of the memory range to flush. Pass VK_WHOLE_SIZE to flush the complete buffer range.
            /// @param offset (Optional) Byte offset from beginning
            ///
            /// @return VkResult of the flush call
            ///
            [[nodiscard]] VkResult flush(VkDeviceSize size = VK_WHOLE_SIZE, VkDeviceSize offset = 0) const;

            ///
            /// @brief Create a buffer info descriptor
            ///
            /// @param size (Optional) Size of the memory range of the descriptor
            /// @param offset (Optional) Byte offset from beginning
            ///
            /// @return VkDescriptorBufferInfo of specified offset and range
            ///
            [[nodiscard]] VkDescriptorBufferInfo descriptorInfo(const VkDeviceSize size = VK_WHOLE_SIZE, const VkDeviceSize offset = 0) const { return VkDescriptorBufferInfo{m_buffer, offset, size, }; }

            ///
            /// @brief Invalidate a memory range of the buffer to make it visible to the host
            ///
            /// @note Only required for non-coherent memory
            ///
            /// @param size (Optional) Size of the memory range to invalidate. Pass VK_WHOLE_SIZE to invalidate
            /// the complete buffer range.
            /// @param offset (Optional) Byte offset from beginning
            ///
            /// @return VkResult of the invalidate call
            ///
            [[nodiscard]] VkResult invalidate(VkDeviceSize size = VK_WHOLE_SIZE, VkDeviceSize offset = 0) const;

            ///
            /// Copies "instanceSize" bytes of data to the mapped buffer at an offset of index * alignmentSize
            ///
            /// @param data Pointer to the data to copy
            /// @param index Used in offset calculation
            ///
            ///
            void writeToIndex(const void* data, const VkDeviceSize index) const { writeToBuffer(data, m_instanceSize, index * m_alignmentSize); }

            ///
```

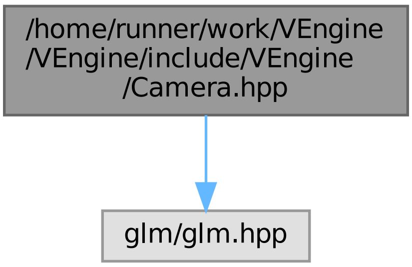
```
            ///  Flush the memory range at index * alignmentSize of the buffer to make it visible to the device
            ///
            /// @param index Used in offset calculation
            ///
            [[nodiscard]] VkResult flushIndex(const VkDeviceSize index) const { return flush(m_alignmentSize, index * m_alignmentSize); }

            ///
            ///
            /// Create a buffer info descriptor
            ///
            /// @param index Specifies the region given by index * alignmentSize
            ///
            /// @return VkDescriptorBufferInfo for instance at index
            ///
            [[nodiscard]] VkDescriptorBufferInfo descriptorInfoForIndex(const VkDeviceSize index) const { return descriptorInfo(m_alignmentSize, index * m_alignmentSize); }

            ///
            /// Invalidate a memory range of the buffer to make it visible to the host
            ///
            /// @note Only required for non-coherent memory
            ///
            /// @param index Specifies the region to invalidate: index * alignmentSize
            ///
            /// @return VkResult of the invalidate call
            ///
            [[nodiscard]] VkResult invalidateIndex(const VkDeviceSize index) const { return invalidate(m_alignmentSize, index * m_alignmentSize); }

            [[nodiscard]] VkBuffer getBuffer() const { return m_buffer; }
            [[nodiscard]] void* getMappedMemory() const { return m_mapped; }
            [[nodiscard]] uint32_t getInstanceCount() const { return m_instanceCount; }
            [[nodiscard]] VkDeviceSize getInstanceSize() const { return m_instanceSize; }
            [[nodiscard]] VkDeviceSize getAlignmentSize() const { return m_instanceSize; }
            [[nodiscard]] VkBufferUsageFlags getUsageFlags() const { return m_usageFlags; }
            [[nodiscard]] VkMemoryPropertyFlags getMemoryPropertyFlags() const { return m_memoryPropertyFlags; }
            [[nodiscard]] VkDeviceSize getBufferSize() const { return m_bufferSize; }

        private:
            ///
            /// Returns the minimum instance size required to be compatible with devices minOffsetAlignment
            ///
            /// @param instanceSize The size of an instance
            /// @param minOffsetAlignment The minimum required alignment, in bytes, for the offset member (eg
            /// minUniformBufferOffsetAlignment)
            ///
            /// @return VkResult of the buffer mapping call
            ///
            static VkDeviceSize getAlignment(VkDeviceSize instanceSize, VkDeviceSize minOffsetAlignment);

            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;

    }; // class Buffer

} // namespace ven
```

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

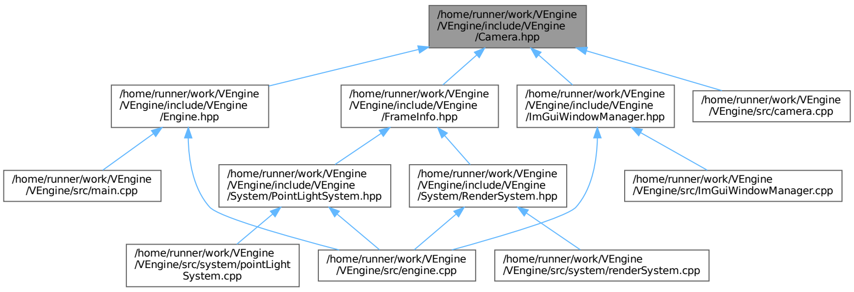
This file contains the Camera class.

`#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

  Class for camera.

Namespaces

- namespace ven

Variables

- static constexpr glm::vec3 ven::DEFAULT_POSITION {0.F, 0.F, -2.5F}
- static constexpr glm::vec3 ven::DEFAULT_ROTATION {0.F, 0.F, 0.F}
- static constexpr float ven::DEFAULT_FOV = glm::radians(50.0F)
- static constexpr float ven::DEFAULT_NEAR = 0.1F
- static constexpr float ven::DEFAULT_FAR = 100.F

7.3.1 Detailed Description

This file contains the Camera class.

This file contains the KeyboardController class.

Definition in file Camera.hpp.

7.4 Camera.hpp

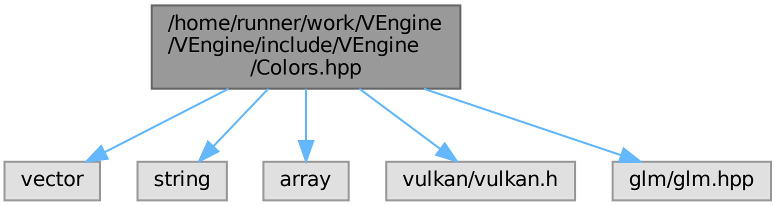
Go to the documentation of this file.

```
///
/// @file Camera.hpp
/// @brief This file contains the Camera class
/// @namespace ven
///

#pragma once

#include <glm/glm.hpp>

namespace ven {

    static constexpr glm::vec3 DEFAULT_POSITION{0.F, 0.F, -2.5F};
    static constexpr glm::vec3 DEFAULT_ROTATION{0.F, 0.F, 0.F};

    static constexpr float DEFAULT_FOV = glm::radians(50.0F);
    static constexpr float DEFAULT_NEAR = 0.1F;
    static constexpr float DEFAULT_FAR = 100.F;

    ///
    /// @class Camera
    /// @brief Class for camera
    /// @namespace ven
    ///
    class Camera {

        public:

            void setOrthographicProjection(float left, float right, float top, float bottom, float near, float far);
            void setPerspectiveProjection(float aspect);
            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}) { setViewDirection(position, target - position, up); }
            void setViewYXZ(glm::vec3 position, glm::vec3 rotation);
            void setFov(float fov) { m_fov = fov; }
            void setNear(float near) { m_near = near; }
            void setFar(float far) { m_far = far; }

            [[nodiscard]] const glm::mat4& getProjection() const { return m_projectionMatrix; }
            [[nodiscard]] const glm::mat4& getView() const { return m_viewMatrix; }
            [[nodiscard]] const glm::mat4& getInverseView() const { return m_inverseViewMatrix; }
            [[nodiscard]] float getFov() const { return m_fov; }
            [[nodiscard]] float getNear() const { return m_near; }
            [[nodiscard]] float getFar() const { return m_far; }

        private:

            float m_fov{DEFAULT_FOV};
            float m_near{DEFAULT_NEAR};
            float m_far{DEFAULT_FAR};
            glm::mat4 m_projectionMatrix{1.F};
            glm::mat4 m_viewMatrix{1.F};
            glm::mat4 m_inverseViewMatrix{1.F};

    }; // class Camera

} // namespace ven
```

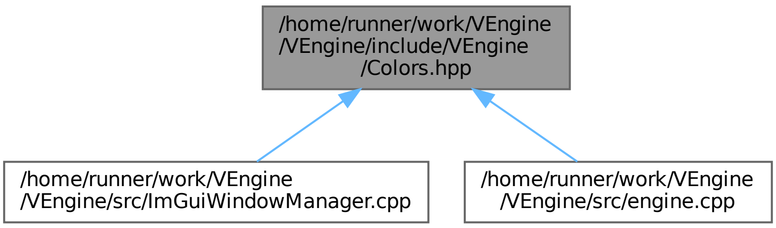
7.5 /home/runner/work/VEngine/VEngine/include/VEngine/Colors.hpp File Reference

```
#include <vector>
#include <string>
#include <array>
#include <vulkan/vulkan.h>
#include <glm/glm.hpp>
```

Include dependency graph for Colors.hpp:



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



Classes

- class ven::Colors

    Class for colors.

Namespaces

- namespace ven

7.6 Colors.hpp

Go to the documentation of this file.

```
///
/// @file Colors.hpp
/// @brief
/// @namespace ven
///

#pragma once

#include <vector>
#include <string>
#include <array>

#include <vulkan/vulkan.h>

#include <glm/glm.hpp>

namespace ven {

    ///
    /// @class Colors
    /// @brief Class for colors
    /// @namespace ven
    ///
    class Colors {

        static constexpr float COLOR_MAX = 255.0F;

        public:

            static constexpr glm::vec3 WHITE = glm::vec3(COLOR_MAX, COLOR_MAX, COLOR_MAX) / COLOR_MAX;
            static constexpr glm::vec3 BLACK = glm::vec3(0.0F);
            static constexpr glm::vec3 RED = glm::vec3(COLOR_MAX, 0.0F, 0.0F) / COLOR_MAX;
            static constexpr glm::vec3 GREEN = glm::vec3(0.0F, COLOR_MAX, 0.0F) / COLOR_MAX;
            static constexpr glm::vec3 BLUE = glm::vec3(0.0F, 0.0F, COLOR_MAX) / COLOR_MAX;
            static constexpr glm::vec3 YELLOW = glm::vec3(COLOR_MAX, COLOR_MAX, 0.0F) / COLOR_MAX;
            static constexpr glm::vec3 CYAN = glm::vec3(0.0F, COLOR_MAX, COLOR_MAX) / COLOR_MAX;
            static constexpr glm::vec3 MAGENTA = glm::vec3(COLOR_MAX, 0.0F, COLOR_MAX) / COLOR_MAX;
            static constexpr glm::vec3 SILVER = glm::vec3(192.0F, 192.0F, 192.0F) / COLOR_MAX;
            static constexpr glm::vec3 GRAY = glm::vec3(128.0F, 128.0F, 128.0F) / COLOR_MAX;
            static constexpr glm::vec3 MAROON = glm::vec3(128.0F, 0.0F, 0.0F) / COLOR_MAX;
            static constexpr glm::vec3 OLIVE = glm::vec3(128.0F, 128.0F, 0.0F) / COLOR_MAX;
            static constexpr glm::vec3 LIME = glm::vec3(0.0F, COLOR_MAX, 0.0F) / COLOR_MAX;
            static constexpr glm::vec3 AQUA = glm::vec3(0.0F, COLOR_MAX, COLOR_MAX) / COLOR_MAX;
            static constexpr glm::vec3 TEAL = glm::vec3(0.0F, 128.0F, 128.0F) / COLOR_MAX;
            static constexpr glm::vec3 NAVY = glm::vec3(0.0F, 0.0F, 128.0F) / COLOR_MAX;
            static constexpr glm::vec3 FUCHSIA = glm::vec3(COLOR_MAX, 0.0F, COLOR_MAX) / COLOR_MAX;
            static constexpr glm::vec3 NIGHT_BLUE = glm::vec3(25.0F, 25.0F, 112.0F) / COLOR_MAX;
            static constexpr glm::vec3 SKY_BLUE = glm::vec3(102.0F, 178.0F, 255.0F) / COLOR_MAX;
            static constexpr glm::vec3 SUNSET = glm::vec3(255.0F, 128.0F, 0.0F) / COLOR_MAX;

            static constexpr VkClearColorValue WHITE_V = {{1.0F, 1.0F, 1.0F, 1.0F}};
            static constexpr VkClearColorValue BLACK_V = {{0.0F, 0.0F, 0.0F, 1.0F}};
            static constexpr VkClearColorValue RED_V = {{1.0F, 0.0F, 0.0F, 1.0F}};
            static constexpr VkClearColorValue GREEN_V = {{0.0F, 1.0F, 0.0F, 1.0F}};
            static constexpr VkClearColorValue BLUE_V = {{0.0F, 0.0F, 1.0F, 1.0F}};
            static constexpr VkClearColorValue YELLOW_V = {{1.0F, 1.0F, 0.0F, 1.0F}};
            static constexpr VkClearColorValue CYAN_V = {{0.0F, 1.0F, 1.0F, 1.0F}};
            static constexpr VkClearColorValue MAGENTA_V = {{1.0F, 0.0F, 1.0F, 1.0F}};
            static constexpr VkClearColorValue SILVER_V = {{0.75F, 0.75F, 0.75F, 1.0F}};
            static constexpr VkClearColorValue GRAY_V = {{0.5F, 0.5F, 0.5F, 1.0F}};
            static constexpr VkClearColorValue MAROON_V = {{0.5F, 0.0F, 0.0F, 1.0F}};
            static constexpr VkClearColorValue OLIVE_V = {{0.5F, 0.5F, 0.0F, 1.0F}};
            static constexpr VkClearColorValue LIME_V = {{0.0F, 1.0F, 0.0F, 1.0F}};
            static constexpr VkClearColorValue AQUA_V = {{0.0F, 1.0F, 1.0F, 1.0F}};
            static constexpr VkClearColorValue TEAL_V = {{0.0F, 0.5F, 0.5F, 1.0F}};
            static constexpr VkClearColorValue NAVY_V = {{0.0F, 0.0F, 0.5F, 1.0F}};
            static constexpr VkClearColorValue FUCHSIA_V = {{1.0F, 0.0F, 1.0F, 1.0F}};
            static constexpr VkClearColorValue NIGHT_BLUE_V = {{0.1F, 0.1F, 0.44F, 1.0F}};
            static constexpr VkClearColorValue SKY_BLUE_V = {{0.4F, 0.6F, 0.9F, 1.0F}};
            static constexpr VkClearColorValue SUNSET_V = {{1.0F, 0.5F, 0.0F, 1.0F}};
            static constexpr VkClearColorValue NIGHT_MODE_V = {{0.0F, 0.0F, 0.0F, 1.0F}};

        static constexpr std::array<std::pair<const char*, glm::vec3>, 20> COLORS = {{
                {"White", Colors::WHITE},
                {"Black", Colors::BLACK},
                {"Red", Colors::RED},
                {"Green", Colors::GREEN},
                {"Blue", Colors::BLUE},
                {"Yellow", Colors::YELLOW},
                {"Cyan", Colors::CYAN},
                {"Magenta", Colors::MAGENTA},
                {"Silver", Colors::SILVER},
```

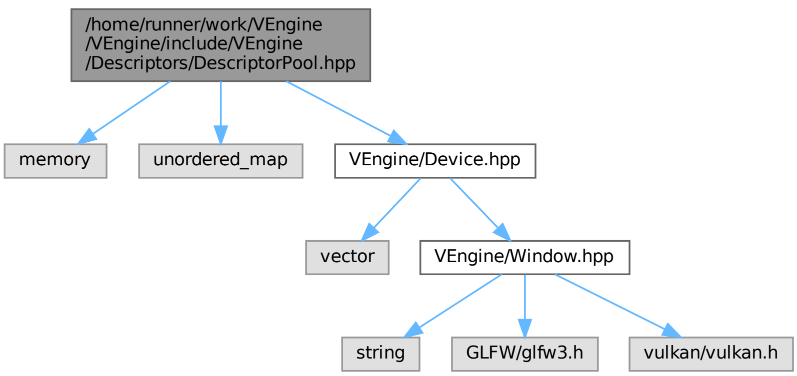
```
                {"Gray", Colors::GRAY},
                {"Maroon", Colors::MAROON},
                {"Olive", Colors::OLIVE},
                {"Lime", Colors::LIME},
                {"Aqua", Colors::AQUA},
                {"Teal", Colors::TEAL},
                {"Navy", Colors::NAVY},
                {"Fuchsia", Colors::FUCHSIA},
                {"Night Blue", ven::Colors::NIGHT_BLUE},
                {"Sky Blue", Colors::SKY_BLUE},
                {"Sunset", Colors::SUNSET}
        }};

        static constexpr std::array<std::pair<const char*, VkClearColorValue>, 20> CLEAR_COLORS = {{
                {"White", Colors::WHITE_V},
                {"Black", Colors::BLACK_V},
                {"Red", Colors::RED_V},
                {"Green", Colors::GREEN_V},
                {"Blue", Colors::BLUE_V},
                {"Yellow", Colors::YELLOW_V},
                {"Cyan", Colors::CYAN_V},
                {"Magenta", Colors::MAGENTA_V},
                {"Silver", Colors::SILVER_V},
                {"Gray", Colors::GRAY_V},
                {"Maroon", Colors::MAROON_V},
                {"Olive", Colors::OLIVE_V},
                {"Lime", Colors::LIME_V},
                {"Aqua", Colors::AQUA_V},
                {"Teal", Colors::TEAL_V},
                {"Navy", Colors::NAVY_V},
                {"Fuchsia", Colors::FUCHSIA_V},
                {"Night Blue", Colors::NIGHT_BLUE_V},
                {"Sky Blue", Colors::SKY_BLUE_V},
                {"Sunset", Colors::SUNSET_V}
        }};

    }; // class Colors

} // namespace ven
```

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

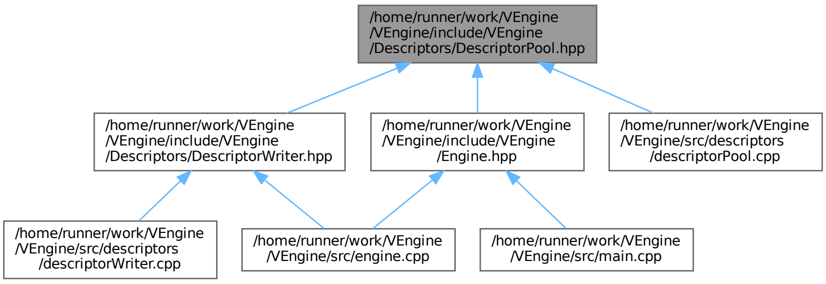
This file contains the DescriptorPool class.

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

Include dependency graph for DescriptorPool.hpp:



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



Classes

- class ven::DescriptorPool

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

Namespaces

- namespace ven

7.7.1 Detailed Description

This file contains the DescriptorPool class.

Definition in file DescriptorPool.hpp.

7.8 DescriptorPool.hpp

Go to the documentation of this file.

```
///
/// @file DescriptorPool.hpp
/// @brief This file contains the DescriptorPool class
/// @namespace ven
///

#pragma once

#include <memory>
#include <unordered_map>

#include "VEngine/Device.hpp"

namespace ven {

    ///
    /// @class DescriptorPool
    /// @brief Class for descriptor pool
    /// @namespace ven
    ///
    class DescriptorPool {

        public:

```

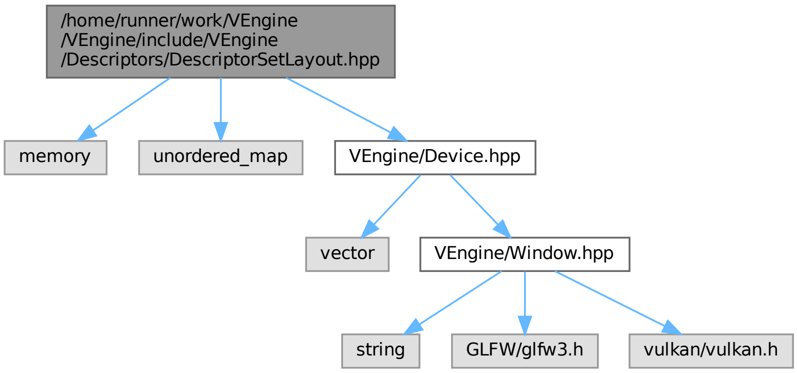
```
            class Builder {

                public:

                    explicit Builder(Device &device) : m_device{device} {}

                    Builder &addPoolSize(VkDescriptorType descriptorType, uint32_t count);
                    Builder &setPoolFlags(VkDescriptorPoolCreateFlags flags);
                    Builder &setMaxSets(uint32_t count);
                    [[nodiscard]] std::unique_ptr<DescriptorPool> build() const { return
      std::make_unique<DescriptorPool>(m_device, m_maxSets, m_poolFlags, m_poolSizes); }

                private:

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

            }; // class Builder

            DescriptorPool(Device &device, uint32_t maxSets, VkDescriptorPoolCreateFlags poolFlags,
      const std::vector<VkDescriptorPoolSize> &poolSizes);
            ~DescriptorPool() { vkDestroyDescriptorPool(m_device.device(), m_descriptorPool, nullptr);
      }
            DescriptorPool(const DescriptorPool &) = delete;
            DescriptorPool &operator=(const DescriptorPool &) = delete;

            bool allocateDescriptor(VkDescriptorSetLayout descriptorSetLayout, VkDescriptorSet
      &descriptor) const;
            void freeDescriptors(const std::vector<VkDescriptorSet> &descriptors) const {
      vkFreeDescriptorSets(m_device.device(), m_descriptorPool, static_cast<uint32_t>(descriptors.size()),
      descriptors.data()); }
            void resetPool() const { vkResetDescriptorPool(m_device.device(), m_descriptorPool, 0); }

            [[nodiscard]] VkDescriptorPool getDescriptorPool() const { return m_descriptorPool; }

        private:

            Device &m_device;
            VkDescriptorPool m_descriptorPool;
            friend class DescriptorWriter;

    }; // class DescriptorPool

} // namespace ven
```

7.9 /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorSetLayout.hpp File Reference

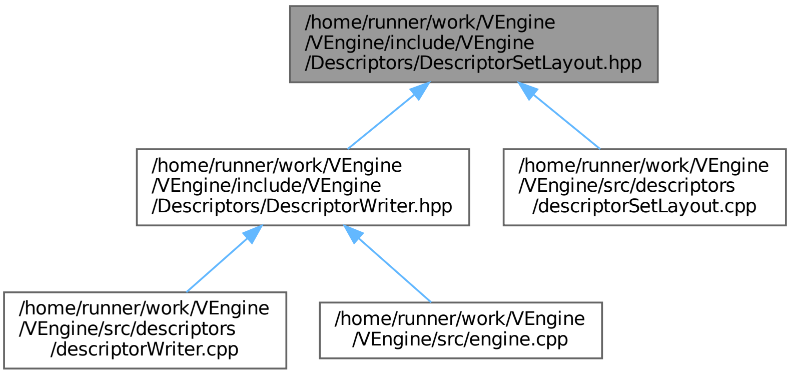
This file contains the DescriptorSetLayout class.

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

Include dependency graph for DescriptorSetLayout.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

Namespaces

- namespace ven

7.9.1 Detailed Description

This file contains the DescriptorSetLayout class.

Definition in file DescriptorSetLayout.hpp.

7.10 DescriptorSetLayout.hpp

Go to the documentation of this file.

```
///
/// @file DescriptorSetLayout.hpp
/// @brief This file contains the DescriptorSetLayout class
/// @namespace ven
///

#pragma once

#include <memory>
#include <unordered_map>

#include "VEngine/Device.hpp"

namespace ven {

    ///
    /// @class DescriptorSetLayout
    /// @brief Class for descriptor set layout
    /// @namespace ven
    ///
    class DescriptorSetLayout {

        public:

            class Builder {

                public:

                    explicit Builder(Device &device) : m_device{device} {}

                    Builder &addBinding(uint32_t binding, VkDescriptorType descriptorType, VkShaderStageFlags stageFlags, uint32_t count = 1);
                    std::unique_ptr<DescriptorSetLayout> build() const { return std::make_unique<DescriptorSetLayout>(m_device, m_bindings); }

                private:

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

            }; // class Builder

            DescriptorSetLayout(Device &device, const std::unordered_map<uint32_t, VkDescriptorSetLayoutBinding>& bindings);
            ~DescriptorSetLayout() { vkDestroyDescriptorSetLayout(m_device.device(), m_descriptorSetLayout, nullptr); }
            DescriptorSetLayout(const DescriptorSetLayout &) = delete;
            DescriptorSetLayout &operator=(const DescriptorSetLayout &) = delete;

            VkDescriptorSetLayout getDescriptorSetLayout() const { return m_descriptorSetLayout; }

        private:

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

            friend class DescriptorWriter;

    }; // class DescriptorSetLayout

} // namespace ven
```

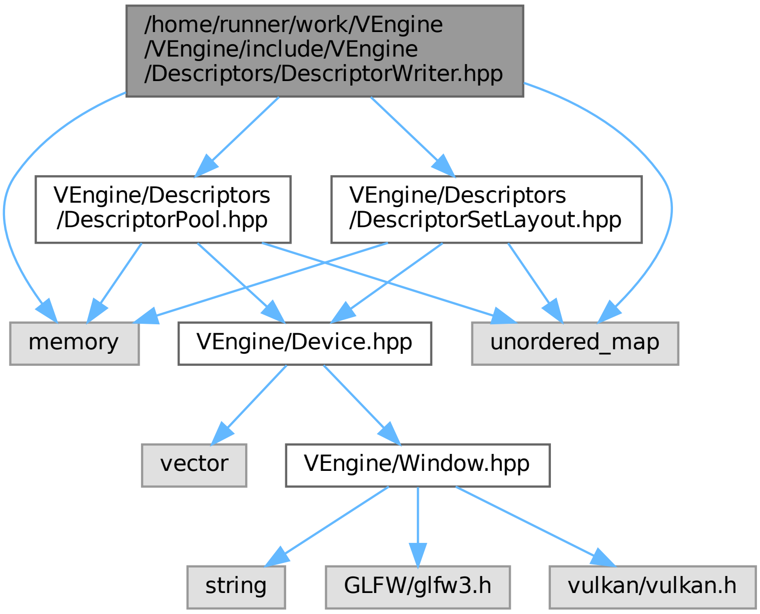
7.11 /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorWriter.hpp File Reference

This file contains the DescriptorsWriter class.

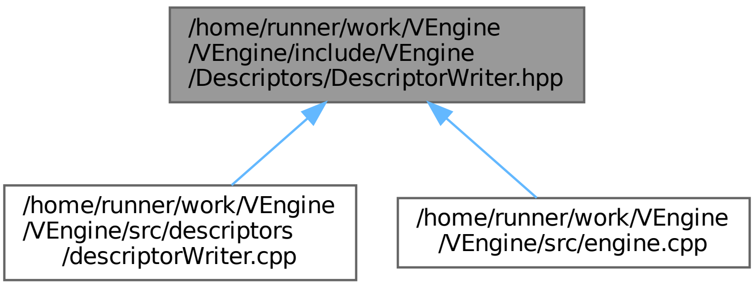
```
#include <memory>
#include <unordered_map>
#include "VEngine/Descriptors/DescriptorPool.hpp"
```

`#include "VEngine/Descriptors/DescriptorSetLayout.hpp"`

Include dependency graph for DescriptorWriter.hpp:



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



Classes

- class ven::DescriptorWriter

    Class for descriptor writer.

Namespaces

- namespace ven

7.11.1 Detailed Description

This file contains the DescriptorsWriter class.

Definition in file DescriptorWriter.hpp.

7.12 DescriptorWriter.hpp

Go to the documentation of this file.

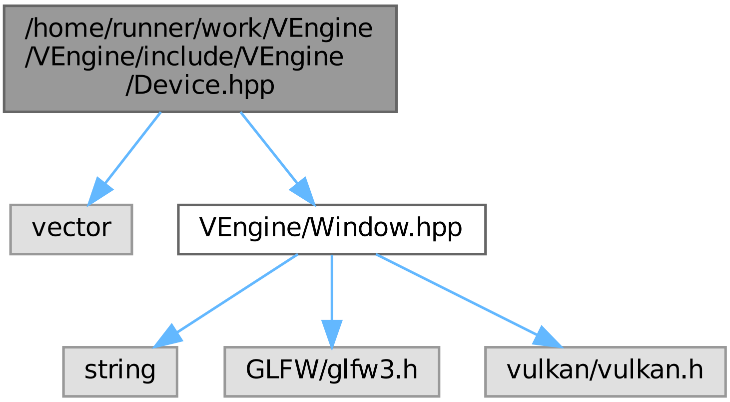
```
///
/// @file DescriptorWriter.hpp
/// @brief This file contains the DescriptorsWriter class
/// @namespace ven
///

#pragma once

#include <memory>
#include <unordered_map>

#include "VEngine/Descriptors/DescriptorPool.hpp"
#include "VEngine/Descriptors/DescriptorSetLayout.hpp"

namespace ven {

    ///
    /// @class DescriptorWriter
    /// @brief Class for descriptor writer
    /// @namespace ven
    ///
    class DescriptorWriter {

        public:

            DescriptorWriter(DescriptorSetLayout &setLayout, DescriptorPool &pool) :
      m_setLayout{setLayout}, m_pool{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:

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

    }; // class DescriptorWriter

} // namespace ven
```

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

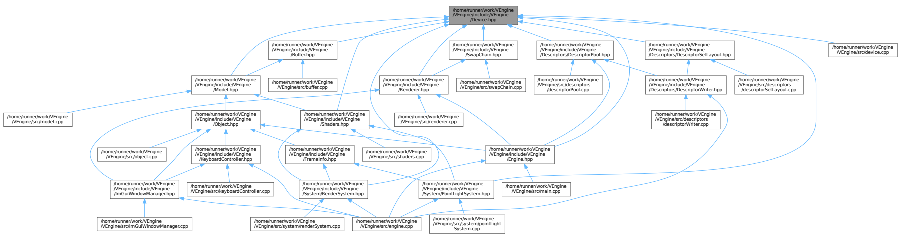
This file contains the Device class.

```
#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

7.13.1 Detailed Description

This file contains the Device class.

Definition in file Device.hpp.

7.14 Device.hpp

Go to the documentation of this file.

```cpp
///
/// @file Device.hpp
/// @brief This file contains the Device class
/// @namespace ven
///

#pragma once

#include <vector>

#include "VEngine/Window.hpp"

namespace ven {

    struct SwapChainSupportDetails {
        VkSurfaceCapabilitiesKHR capabilities;
        std::vector<VkSurfaceFormatKHR> formats;
        std::vector<VkPresentModeKHR> presentModes;
    };

    struct QueueFamilyIndices {
        uint32_t graphicsFamily{};
        uint32_t presentFamily{};
        bool graphicsFamilyHasValue = false;
        bool presentFamilyHasValue = false;
        [[nodiscard]] bool isComplete() const { return graphicsFamilyHasValue && presentFamilyHasValue; }
    };

    class Device {

        public:

            #ifdef NDEBUG
                const bool enableValidationLayers = false;
            #else
                const bool enableValidationLayers = true;
            #endif

            explicit Device(Window &window);
            ~Device();

            Device(const Device &) = delete;
            Device& operator=(const Device &) = delete;
            Device(Device &&) = delete;
            Device &operator=(Device &&) = delete;

            [[nodiscard]] VkCommandPool getCommandPool() const { return m_commandPool; }
            [[nodiscard]] VkDevice device() const { return m_device; }
            [[nodiscard]] VkSurfaceKHR surface() const { return m_surface; }
            [[nodiscard]] VkQueue graphicsQueue() const { return m_graphicsQueue; }
            [[nodiscard]] VkQueue presentQueue() const { return m_presentQueue; }

            [[nodiscard]] SwapChainSupportDetails getSwapChainSupport() const { return querySwapChainSupport(m_physicalDevice); }
            [[nodiscard]] uint32_t findMemoryType(uint32_t typeFilter, VkMemoryPropertyFlags propertiesp) const;
            [[nodiscard]] QueueFamilyIndices findPhysicalQueueFamilies() const { return findQueueFamilies(m_physicalDevice); }
            [[nodiscard]] VkFormat findSupportedFormat(const std::vector<VkFormat> &candidates, VkImageTiling tiling, VkFormatFeatureFlags features) const;

            // Buffer Helper Functions
            void createBuffer(VkDeviceSize size, VkBufferUsageFlags usage, VkMemoryPropertyFlags propertiesp, VkBuffer &buffer, VkDeviceMemory &bufferMemory) const;
            [[nodiscard]] 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;

            [[nodiscard]] VkPhysicalDevice getPhysicalDevice() const { return m_physicalDevice; }
            [[nodiscard]] VkQueue getGraphicsQueue() const { return m_graphicsQueue; }

            VkPhysicalDeviceProperties properties_;

        private:

            void createInstance();
```

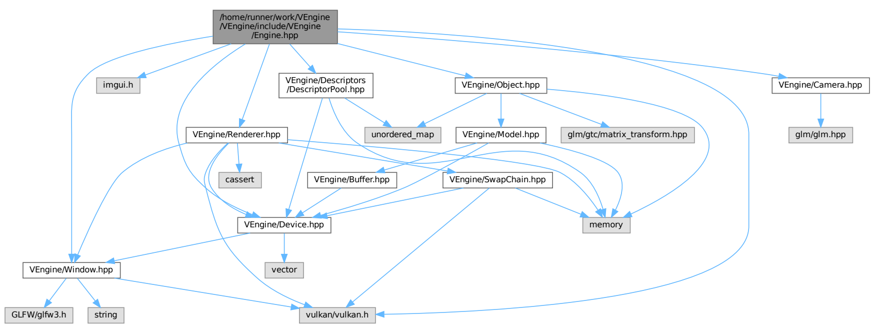
```
            void setupDebugMessenger();
            void createSurface() { m_window.createWindowSurface(m_instance, &m_surface); };
            void pickPhysicalDevice();
            void createLogicalDevice();
            void createCommandPool();

            // helper functions
            bool isDeviceSuitable(VkPhysicalDevice device) const;
            [[nodiscard]] std::vector<const char *> getRequiredExtensions() const;
            [[nodiscard]] bool checkValidationLayerSupport() const;
            QueueFamilyIndices findQueueFamilies(VkPhysicalDevice device) const;
            static void populateDebugMessengerCreateInfo(VkDebugUtilsMessengerCreateInfoEXT &createInfo);
            void hasGlfwRequiredInstanceExtensions() const;
            bool checkDeviceExtensionSupport(VkPhysicalDevice device) const;
            SwapChainSupportDetails querySwapChainSupport(VkPhysicalDevice device) const;

            VkInstance m_instance;
            VkDebugUtilsMessengerEXT m_debugMessenger;
            VkPhysicalDevice m_physicalDevice = VK_NULL_HANDLE;
            Window &m_window;
            VkCommandPool m_commandPool;

            VkDevice m_device;
            VkSurfaceKHR m_surface;
            VkQueue m_graphicsQueue;
            VkQueue m_presentQueue;

            const std::vector<const char *> validationLayers = {"VK_LAYER_KHRONOS_validation"};
            const std::vector<const char *> deviceExtensions = {VK_KHR_SWAPCHAIN_EXTENSION_NAME};

    }; // class Device

} // namespace ven
```

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

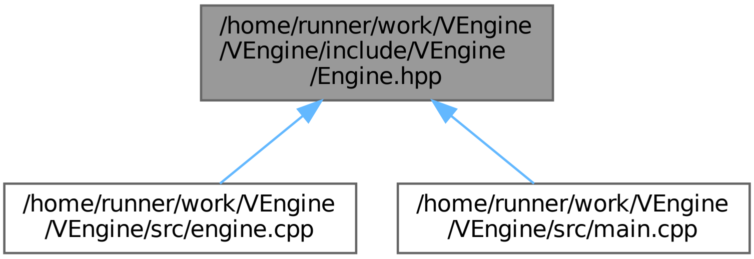
This file contains the Engine class.

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

7.15.1 Detailed Description

This file contains the Engine class.

Definition in file Engine.hpp.

7.16 Engine.hpp

Go to the documentation of this file.

```
///
/// @file Engine.hpp
/// @brief This file contains the Engine class
/// @namespace ven
///

#pragma once

#include <vulkan/vulkan.h>

#include <imgui.h>

#include "VEngine/Window.hpp"
#include "VEngine/Device.hpp"
#include "VEngine/Object.hpp"
#include "VEngine/Renderer.hpp"
#include "VEngine/Descriptors/DescriptorPool.hpp"
#include "VEngine/Camera.hpp"

namespace ven {

    class Engine {

    public:

```

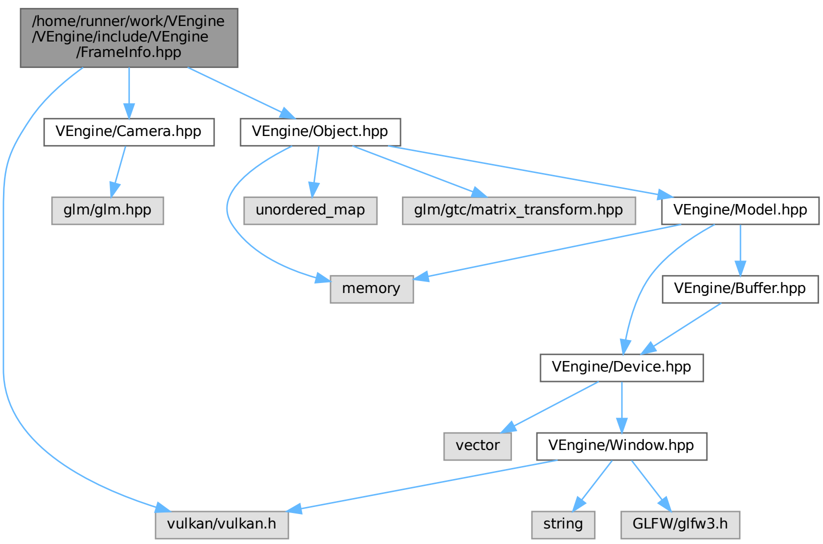
```
        explicit Engine(uint32_t = DEFAULT_WIDTH, uint32_t = DEFAULT_HEIGHT, const std::string &title = DEFAULT_TITLE.data());
        ~Engine() = default;

        Engine(const Engine &) = delete;
        Engine operator=(const Engine &) = delete;

        void mainLoop();

    private:

        void loadObjects();

        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};

        void createInstance();
        void createSurface() { if (glfwCreateWindowSurface(m_instance, m_window.getGLFWindow(), nullptr, &m_surface) != VK_SUCCESS) { throw std::runtime_error("Failed to create window surface"); } }

    }; // class Engine

} // namespace ven
```

7.17 /home/runner/work/VEngine/VEngine/include/VEngine/Frame↩ Info.hpp File Reference

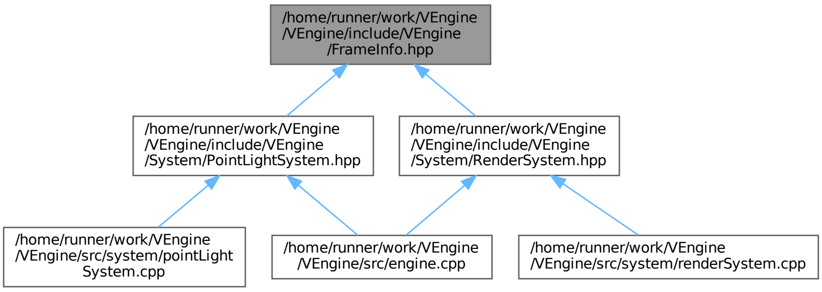
This file contains the FrameInfo class.

```
#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

7.17.1 Detailed Description

This file contains the FrameInfo class.

Definition in file FrameInfo.hpp.

7.18 FrameInfo.hpp

Go to the documentation of this file.

```
///
/// @file FrameInfo.hpp
/// @brief This file contains the FrameInfo class
/// @namespace ven
///

#pragma once

#include <vulkan/vulkan.h>

#include "VEngine/Camera.hpp"
#include "VEngine/Object.hpp"

namespace ven {

static constexpr std::size_t MAX_LIGHTS = 10;
```

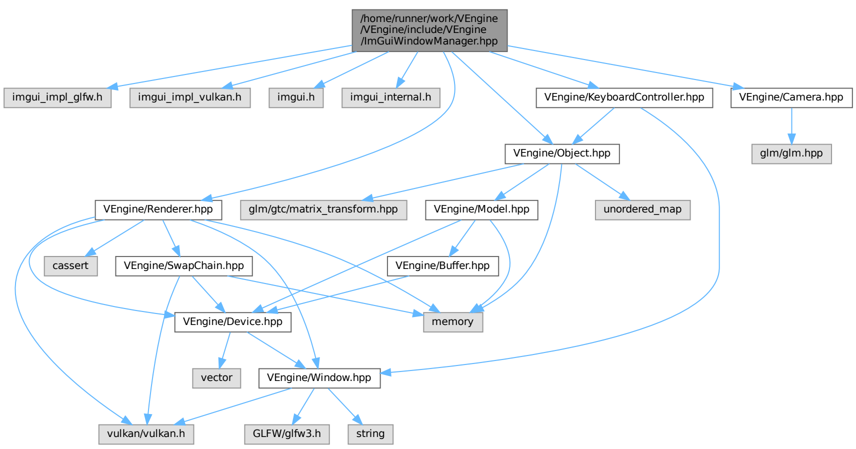
```

    struct PointLight
    {
        glm::vec4 position{};
        glm::vec4 color{};
    };

    struct GlobalUbo
    {
        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;
    };

    struct FrameInfo
    {
        int frameIndex;
        float frameTime;
        VkCommandBuffer commandBuffer;
        Camera &camera;
        VkDescriptorSet globalDescriptorSet;
        Object::Map &objects;
    };

} // namespace ven
```

7.19 /home/runner/work/VEngine/VEngine/include/VEngine/ImGuiWindowManager.hpp File Reference

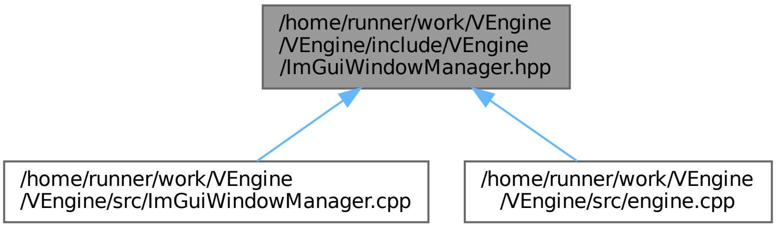
This file contains the ImGuiWindowManager class.

```
#include <imgui_impl_glfw.h>
#include <imgui_impl_vulkan.h>
#include <imgui.h>
#include <imgui_internal.h>
#include "VEngine/Object.hpp"
#include "VEngine/Renderer.hpp"
#include "VEngine/Camera.hpp"
#include "VEngine/KeyboardController.hpp"
```

Include dependency graph for ImGuiWindowManager.hpp:



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



Classes

- class ven::ImGuiWindowManager

  Class for ImGui window manager.
- struct ven::ImGuiWindowManager::funcs

Namespaces

- namespace ven

7.19.1 Detailed Description

This file contains the ImGuiWindowManager class.

Definition in file ImGuiWindowManager.hpp.

7.20 ImGuiWindowManager.hpp

Go to the documentation of this file.

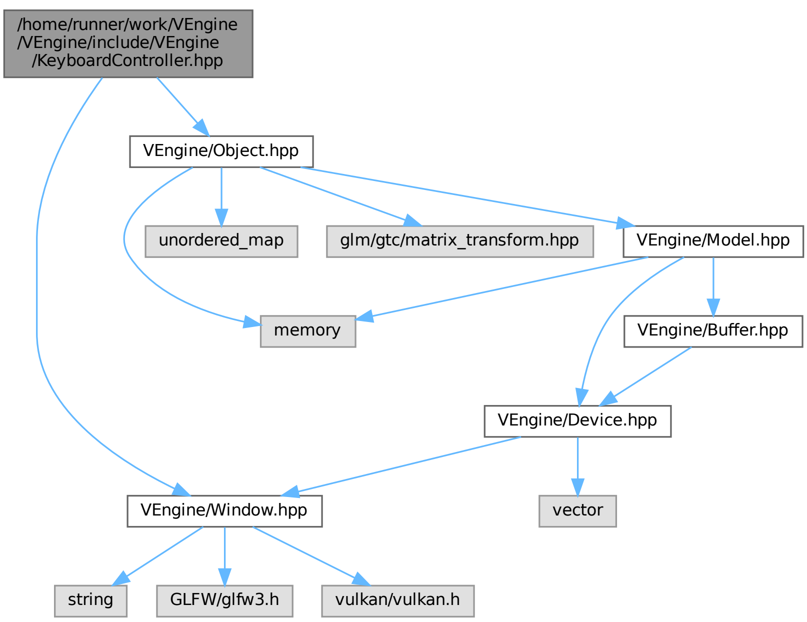
```
///
/// @file ImGuiWindowManager.hpp
/// @brief This file contains the ImGuiWindowManager class
/// @namespace ven
///

#pragma once

#include <imgui_impl_glfw.h>
#include <imgui_impl_vulkan.h>
#include <imgui.h>
#include <imgui_internal.h>

#include "VEngine/Object.hpp"
#include "VEngine/Renderer.hpp"
#include "VEngine/Camera.hpp"
#include "VEngine/KeyboardController.hpp"

namespace ven {

    ///
    /// @class ImGuiWindowManager
    /// @brief Class for ImGui window manager
    /// @namespace ven
```

```
    ///
    class ImGuiWindowManager {

        public:

            ImGuiWindowManager() = default;
            ~ImGuiWindowManager() = default;

            ImGuiWindowManager(const ImGuiWindowManager&) = delete;
            ImGuiWindowManager& operator=(const ImGuiWindowManager&) = delete;

            static void init(GLFWwindow* window, VkInstance instance, Device* device, VkRenderPass renderPass);
            static void render(Renderer *renderer, std::unordered_map<unsigned int, Object>& objects, ImGuiIO& io, Object& cameraObj, Camera& camera, KeyboardController& cameraController, VkPhysicalDevice physicalDevice);
            static void cleanup();

        private:

            struct funcs { static bool IsLegacyNativeDupe(ImGuiKey key) { return key >= 0 && key < 512 && ImGui::GetIO().KeyMap[key] != -1; } }; // Hide Native<>ImGuiKey duplicates when both exist

    }; // class ImGuiWindowManager

} // namespace ven
```

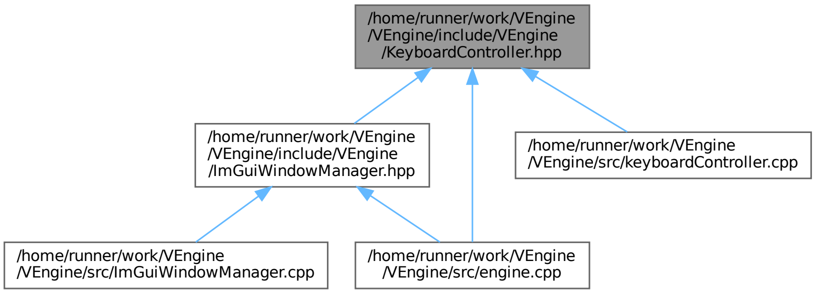
7.21 /home/runner/work/VEngine/VEngine/include/VEngine/KeyboardController.hpp File Reference

```
#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

  Class for keyboard controller.
- struct ven::KeyboardController::KeyMappings

Namespaces

- namespace ven

Variables

- static constexpr float ven::DEFAULT_MOVE_SPEED = 3.F
- static constexpr float ven::DEFAULT_LOOK_SPEED = 1.5F

7.22 KeyboardController.hpp

Go to the documentation of this file.

```
///
/// @file Camera.hpp
/// @brief This file contains the KeyboardController class
/// @namespace ven
///

#pragma once

#include "VEngine/Window.hpp"
#include "VEngine/Object.hpp"

namespace ven {

    static constexpr float DEFAULT_MOVE_SPEED = 3.F;
    static constexpr float DEFAULT_LOOK_SPEED = 1.5F;

    ///
    /// @class KeyboardController
    /// @brief Class for keyboard controller
    /// @namespace ven
    ///
    class KeyboardController {

        public:

            struct KeyMappings {
```

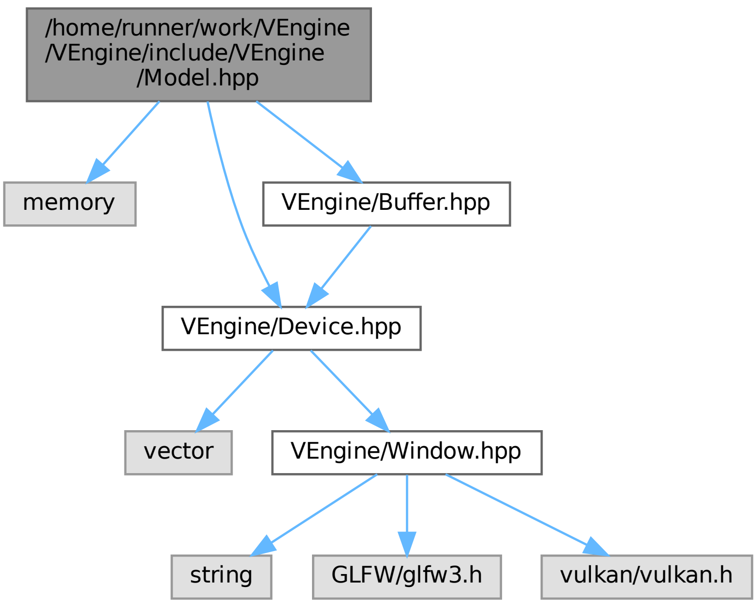
```
            int moveLeft = GLFW_KEY_A;
            int moveRight = GLFW_KEY_D;
            int moveForward = GLFW_KEY_W;
            int moveBackward = GLFW_KEY_S;
            int moveUp = GLFW_KEY_SPACE;
            int moveDown = GLFW_KEY_LEFT_SHIFT;
            int lookLeft = GLFW_KEY_LEFT;
 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

        void moveInPlaneXZ(GLFWwindow* window, float dt, Object& object, bool* showDebugWindow) const;

        KeyMappings m_keys{};
        float m_moveSpeed{DEFAULT_MOVE_SPEED};
        float m_lookSpeed{DEFAULT_LOOK_SPEED};

    }; // class KeyboardController

} // namespace ven
```

7.23 /home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp File Reference

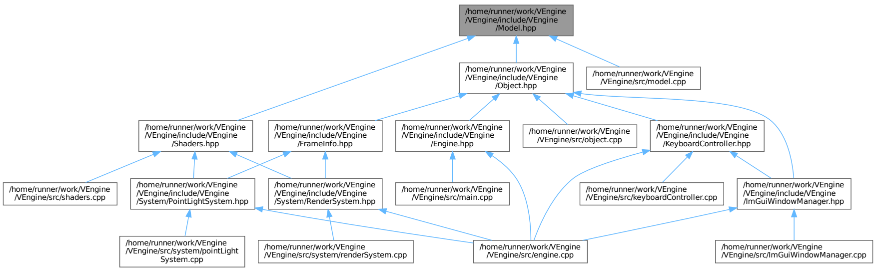
This file contains the Model class.

```
#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

    Class for model.
- struct ven::Model::Vertex
- struct ven::Model::Builder

Namespaces

- namespace ven

7.23.1 Detailed Description

This file contains the Model class.

Definition in file Model.hpp.

7.24 Model.hpp

Go to the documentation of this file.

```
///
/// @file Model.hpp
/// @brief This file contains the Model class
/// @namespace ven
///

#pragma once

#include <memory>

#include "VEngine/Device.hpp"
#include "VEngine/Buffer.hpp"

namespace ven {

    ///
    /// @class Model
    /// @brief Class for model
    /// @namespace ven
    ///
    class Model {

        public:

            struct Vertex {
                glm::vec3 position{};
```

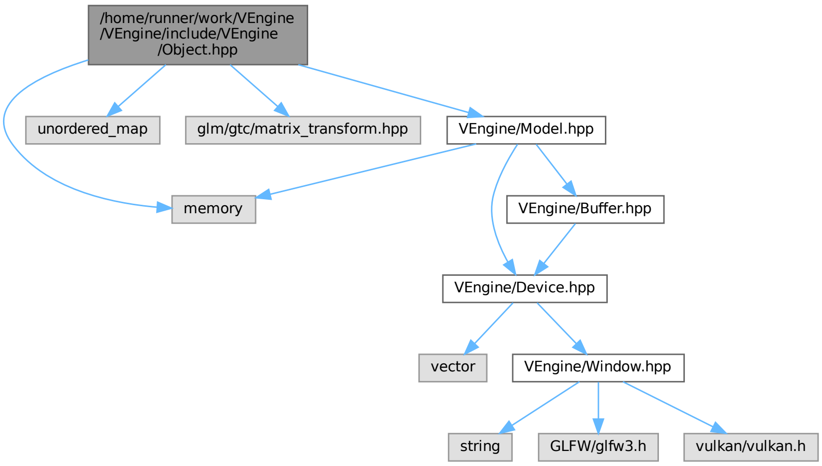
```
                glm::vec3 color{};
                glm::vec3 normal{};
                glm::vec2 uv{};

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

                bool operator==(const Vertex& other) const {
                    return position == other.position && color == other.color && normal == other.normal && uv == other.uv;
                }
            };

            struct Builder {
                std::vector<Vertex> vertices;
                std::vector<uint32_t> indices;

                void loadModel(const std::string &filename);
            };

            Model(Device &device, const Builder &builder);
            ~Model();

            Model(const Model&) = delete;
            void operator=(const Model&) = delete;

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

            void bind(VkCommandBuffer commandBuffer) const;
            void draw(VkCommandBuffer commandBuffer) const;

        private:

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

            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;

    }; // class Model

} // namespace ven
```

7.25 /home/runner/work/VEngine/VEngine/include/VEngine/Object.hpp File Reference

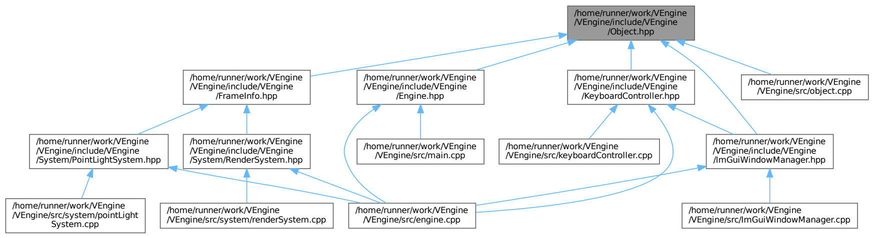
This file contains the Object class.

```
#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

  Class for object.

Namespaces

- namespace ven

Variables

- static constexpr float ven::DEFAULT_LIGHT_INTENSITY = .2F
- static constexpr float ven::DEFAULT_LIGHT_RADIUS = 0.1F
- static constexpr glm::vec3 ven::DEFAULT_LIGHT_COLOR = glm::vec3(1.F)

7.25.1 Detailed Description

This file contains the Object class.

Definition in file Object.hpp.

7.26 Object.hpp

Go to the documentation of this file.

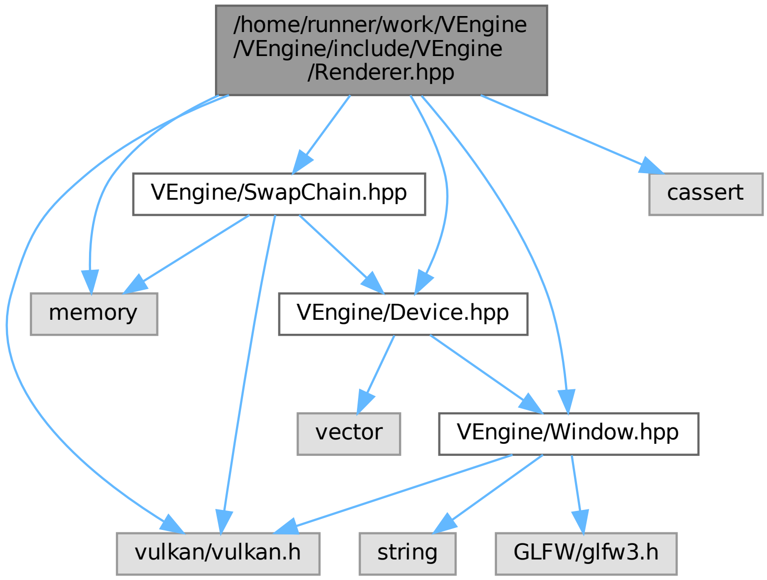
```
///
/// @file Object.hpp
/// @brief This file contains the Object class
/// @namespace ven
///

#pragma once

#include <memory>
#include <unordered_map>

#include <glm/gtc/matrix_transform.hpp>

#include "VEngine/Model.hpp"

namespace ven {

    static constexpr float DEFAULT_LIGHT_INTENSITY = .2F;
    static constexpr float DEFAULT_LIGHT_RADIUS = 0.1F;
    static constexpr glm::vec3 DEFAULT_LIGHT_COLOR = glm::vec3(1.F);

    struct Transform3DComponent {
        glm::vec3 translation{};
        glm::vec3 scale{1.F, 1.F, 1.F};
        glm::vec3 rotation{};

        [[nodiscard]] glm::mat4 mat4() const;
        [[nodiscard]] glm::mat3 normalMatrix() const;
    };

    struct PointLightComponent {
        float lightIntensity = DEFAULT_LIGHT_INTENSITY;
    };

    ///
    /// @class Object
    /// @brief Class for object
    /// @namespace ven
    ///
    class Object {

        public:

            using Map = std::unordered_map<unsigned int, Object>;

            ~Object() = default;

            Object(const Object&) = delete;
            Object& operator=(const Object&) = delete;
            Object(Object&&) = default;
            Object& operator=(Object&&) = default;

            static Object createObject() { static unsigned int objId = 0; return Object(objId++); }
            static Object makePointLight(float intensity = DEFAULT_LIGHT_INTENSITY, float radius = DEFAULT_LIGHT_RADIUS, glm::vec3 color = DEFAULT_LIGHT_COLOR);

            [[nodiscard]] unsigned int getId() const { return m_objId; }

            std::shared_ptr<Model> model{};
            glm::vec3 color{};
            Transform3DComponent transform3D{};
            std::string name{""};
            std::unique_ptr<PointLightComponent> pointLight = nullptr;

        private:

            explicit Object(const unsigned int objId) : m_objId(objId) {}

            unsigned int m_objId;

    }; // class Object

} // namespace ven
```

7.27 /home/runner/work/VEngine/VEngine/include/VEngine/Renderer.hpp File Reference

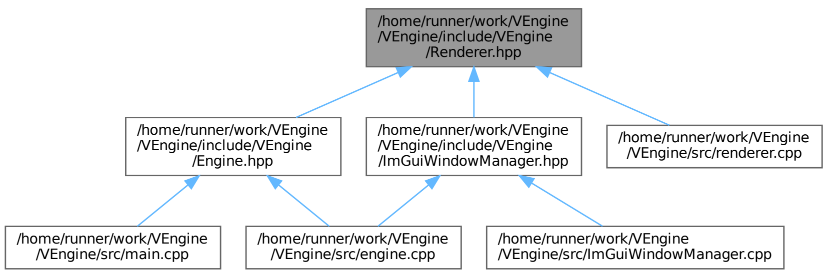
This file contains the Renderer class.

```
#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

Variables

- static constexpr VkClearColorValue ven::DEFAULT_CLEAR_COLOR = {{0.0F, 0.0F, 0.0F, 1.0F}}
- static constexpr VkClearDepthStencilValue ven::DEFAULT_CLEAR_DEPTH = {1.0F, 0}

7.27.1 Detailed Description

This file contains the Renderer class.

Definition in file Renderer.hpp.

7.28 Renderer.hpp

Go to the documentation of this file.

```
///
/// @file Renderer.hpp
/// @brief This file contains the Renderer class
/// @namespace ven
///

#pragma once

#include <memory>
#include <cassert>

#include <vulkan/vulkan.h>

#include "VEngine/Window.hpp"
#include "VEngine/Device.hpp"
#include "VEngine/SwapChain.hpp"

namespace ven {

    static constexpr VkClearColorValue DEFAULT_CLEAR_COLOR = {{0.0F, 0.0F, 0.0F, 1.0F}};
    static constexpr VkClearDepthStencilValue DEFAULT_CLEAR_DEPTH = {1.0F, 0};

    class Renderer {

        public:

            Renderer(Window &window, Device &device) : m_window{window}, m_device{device} { recreateSwapChain(); createCommandBuffers(); }
            ~Renderer() { freeCommandBuffers(); }

            Renderer(const Renderer &) = delete;
            Renderer& operator=(const Renderer &) = delete;

            [[nodiscard]] VkRenderPass getSwapChainRenderPass() const { return m_swapChain->getRenderPass(); }
            [[nodiscard]] float getAspectRatio() const { return m_swapChain->extentAspectRatio(); }
            [[nodiscard]] bool isFrameInProgress() const { return m_isFrameStarted; }
            [[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)]; }

            [[nodiscard]] int getFrameIndex() const { assert(isFrameInProgress() && "cannot get frame index when frame not in progress"); return m_currentFrameIndex; }
            [[nodiscard]] std::array<float, 4> getClearColor() const { return {
                m_clearValues[0].color.float32[0],
                m_clearValues[0].color.float32[1],
                m_clearValues[0].color.float32[2],
                m_clearValues[0].color.float32[3]
            };}

```

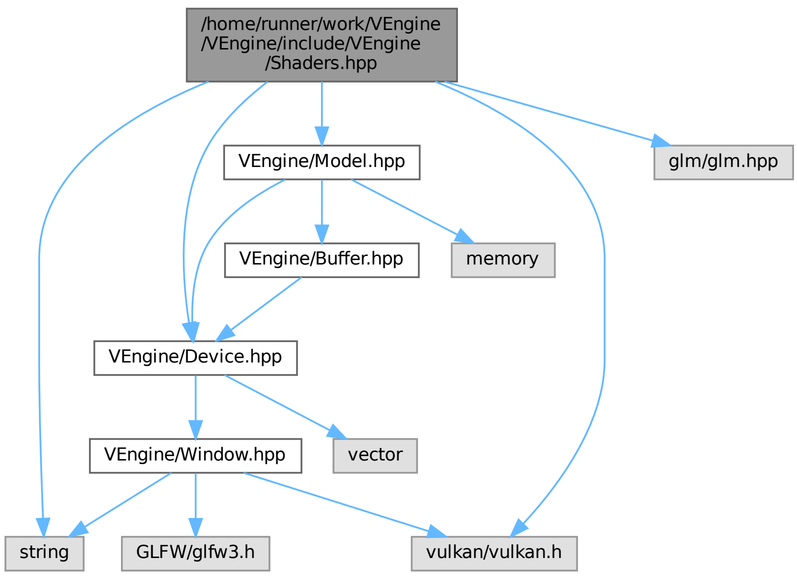
```
            void setClearValue(VkClearColorValue clearColorValue = DEFAULT_CLEAR_COLOR,
      VkClearDepthStencilValue clearDepthValue = DEFAULT_CLEAR_DEPTH) { m_clearValues[0].color =
      clearColorValue; m_clearValues[1].depthStencil = clearDepthValue; }
            VkCommandBuffer beginFrame();
            void endFrame();
            void beginSwapChainRenderPass(VkCommandBuffer commandBuffer);
            void endSwapChainRenderPass(VkCommandBuffer commandBuffer);

        private:

            void createCommandBuffers();
            void freeCommandBuffers();
            void recreateSwapChain();

            Window &m_window;
            Device &m_device;
            std::unique_ptr<SwapChain> m_swapChain;
            std::vector<VkCommandBuffer> m_commandBuffers;
            std::array<VkClearValue, 2> m_clearValues;

            uint32_t m_currentImageIndex{0};
            int m_currentFrameIndex{0};
            bool m_isFrameStarted{false};

    }; // class Renderer

} // namespace ven
```

7.29 /home/runner/work/VEngine/VEngine/include/VEngine/Shaders.hpp File Reference

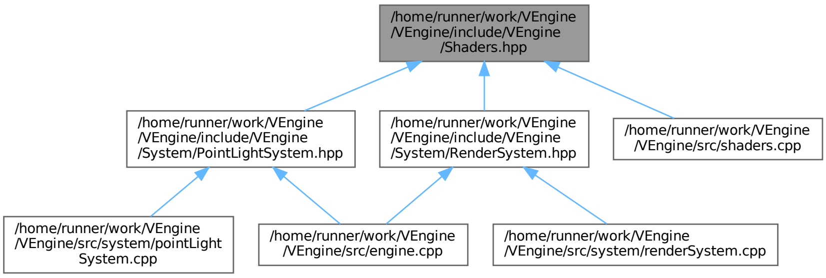
This file contains the Shader class.

```
#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

    Class for shaders.

Namespaces

- namespace ven

Variables

- static constexpr std::string_view ven::SHADERS_BIN_PATH = "shaders/bin/"

7.29.1 Detailed Description

This file contains the Shader class.

Definition in file Shaders.hpp.

7.30 Shaders.hpp

Go to the documentation of this file.

```
///
/// @file Shaders.hpp
/// @brief This file contains the Shader class
/// @namespace ven
///

#pragma once

#include <string>

#include <vulkan/vulkan.h>
#include <glm/glm.hpp>

#include "VEngine/Device.hpp"
#include "VEngine/Model.hpp"

namespace ven {
```

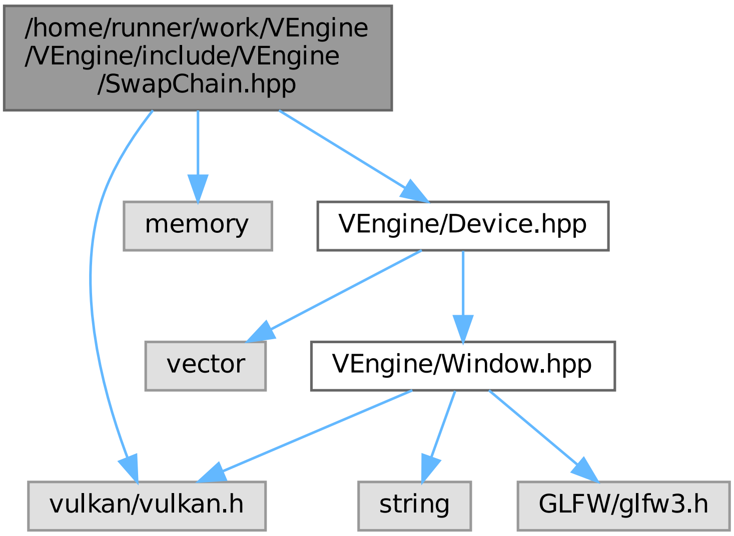
```cpp

    static constexpr std::string_view SHADERS_BIN_PATH = "shaders/bin/";

    struct PipelineConfigInfo {
        PipelineConfigInfo() = default;
        PipelineConfigInfo(const PipelineConfigInfo&) = delete;
        PipelineConfigInfo& operator=(const PipelineConfigInfo&) = delete;

        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;
    };

    ///
    /// @class Shaders
    /// @brief Class for shaders
    /// @namespace ven
    ///
    class Shaders {

        public:

            Shaders(Device &device, const std::string& vertFilepath, const std::string& fragFilepath, const PipelineConfigInfo& configInfo) : m_device{device} { createGraphicsPipeline(vertFilepath, fragFilepath, configInfo); };
            ~Shaders();

            Shaders(const Shaders&) = delete;
            Shaders& operator=(const Shaders&) = delete;

            static void defaultPipelineConfigInfo(PipelineConfigInfo& configInfo);
            void bind(const VkCommandBuffer commandBuffer) const { vkCmdBindPipeline(commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, m_graphicsPipeline); }

        private:

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

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

    }; // class Shaders

} // namespace ven
```

7.31 /home/runner/work/VEngine/VEngine/include/VEngine/SwapChain.hpp File Reference

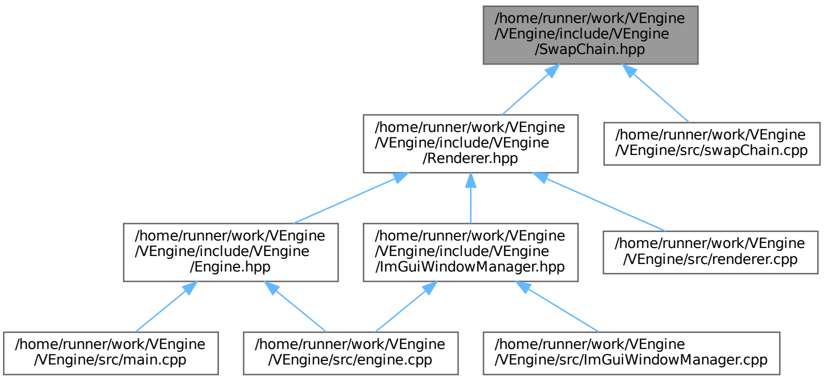
This file contains the Shader class.

```cpp
#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

    Class for swap chain.

Namespaces

- namespace ven

7.31.1 Detailed Description

This file contains the Shader class.

Definition in file SwapChain.hpp.

7.32 SwapChain.hpp

Go to the documentation of this file.

```cpp
///
/// @file SwapChain.hpp
/// @brief This file contains the Shader class
/// @namespace ven
///

#pragma once

#include <vulkan/vulkan.h>
#include <memory>

#include "VEngine/Device.hpp"

namespace ven {

    ///
    /// @class SwapChain
    /// @brief Class for swap chain
    /// @namespace ven
    ///
    class SwapChain {

        public:

            static constexpr int MAX_FRAMES_IN_FLIGHT = 2;

            SwapChain(Device &deviceRef, const VkExtent2D windowExtentRef) : device{deviceRef}, windowExtent{windowExtentRef} { init(); }
            SwapChain(Device &deviceRef, const VkExtent2D windowExtentRef, std::shared_ptr<SwapChain> previous) : device{deviceRef}, windowExtent{windowExtentRef}, oldSwapChain{std::move(previous)} { init(); oldSwapChain = nullptr; }
            ~SwapChain();

            SwapChain(const SwapChain &) = delete;
            SwapChain& operator=(const SwapChain &) = delete;

            [[nodiscard]] VkFramebuffer getFrameBuffer(const unsigned long index) const { return swapChainFrameBuffers[index]; }
            [[nodiscard]] VkRenderPass getRenderPass() const { return renderPass; }
            [[nodiscard]] VkImageView getImageView(const int index) const { return swapChainImageViews[static_cast<unsigned long>(index)]; }
            [[nodiscard]] size_t imageCount() const { return swapChainImages.size(); }
            [[nodiscard]] VkFormat getSwapChainImageFormat() const { return swapChainImageFormat; }
            [[nodiscard]] VkExtent2D getSwapChainExtent() const { return m_swapChainExtent; }
            [[nodiscard]] uint32_t width() const { return m_swapChainExtent.width; }
            [[nodiscard]] uint32_t height() const { return m_swapChainExtent.height; }

            [[nodiscard]] float extentAspectRatio() const { return static_cast<float>(m_swapChainExtent.width) / static_cast<float>(m_swapChainExtent.height); }
            [[nodiscard]] VkFormat findDepthFormat() const;

            VkResult acquireNextImage(uint32_t *imageIndex) const;
            VkResult submitCommandBuffers(const VkCommandBuffer *buffers, const uint32_t *imageIndex);

            [[nodiscard]] bool compareSwapFormats(const SwapChain &swapChainp) const {
                return swapChainImageFormat == swapChainp.swapChainImageFormat && swapChainDepthFormat == swapChainp.swapChainDepthFormat;
            }

        private:

            void init();
            void createSwapChain();
            void createImageViews();
            void createDepthResources();
            void createRenderPass();
            void createFrameBuffers();
            void createSyncObjects();

```

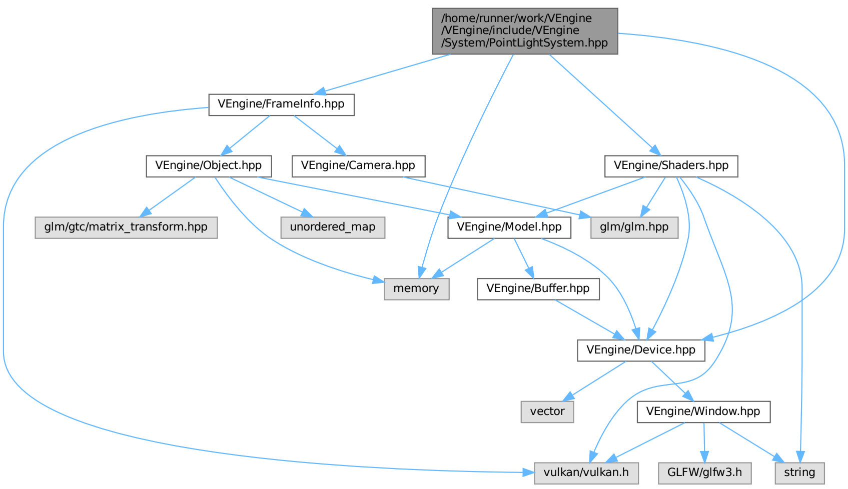
```
            static VkSurfaceFormatKHR chooseSwapSurfaceFormat(const std::vector<VkSurfaceFormatKHR>
    &availableFormats);
            static VkPresentModeKHR chooseSwapPresentMode(const std::vector<VkPresentModeKHR>
    &availablePresentModes);
            [[nodiscard]] VkExtent2D chooseSwapExtent(const VkSurfaceCapabilitiesKHR &capabilities)
    const;

            VkFormat swapChainImageFormat{};
            VkFormat swapChainDepthFormat{};
            VkExtent2D m_swapChainExtent{};

            std::vector<VkFramebuffer> swapChainFrameBuffers;
            VkRenderPass renderPass{};

            std::vector<VkImage> depthImages;
            std::vector<VkDeviceMemory> depthImageMemory;
            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;

    }; // class SwapChain

} // namespace ven
```

7.33 /home/runner/work/VEngine/VEngine/include/VEngine/System/↩ PointLightSystem.hpp File Reference

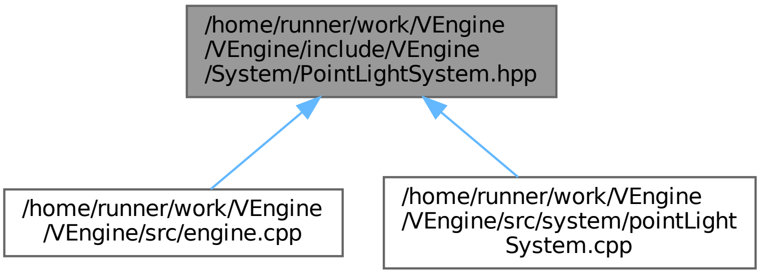
This file contains the PointLightSystem class.

```
#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

7.33.1 Detailed Description

This file contains the PointLightSystem class.

Definition in file PointLightSystem.hpp.

7.34 PointLightSystem.hpp

Go to the documentation of this file.

```
///
/// @file PointLightSystem.hpp
/// @brief This file contains the PointLightSystem class
/// @namespace ven
///

#pragma once

#include <memory>

#include "VEngine/Device.hpp"
#include "VEngine/Shaders.hpp"
#include "VEngine/FrameInfo.hpp"

namespace ven {

    ///
    /// @class PointLightSystem
    /// @brief Class for point light system
    /// @namespace ven
    ///
    class PointLightSystem {

```

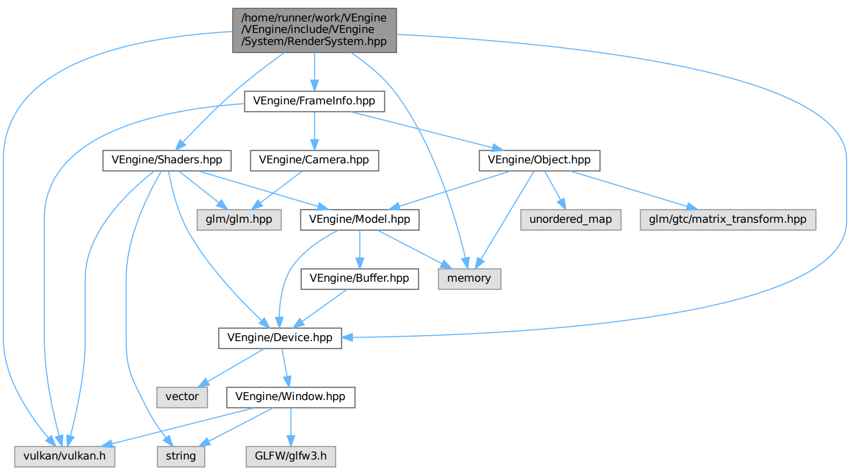
```cpp
    public:

        explicit PointLightSystem(Device& device, VkRenderPass renderPass, VkDescriptorSetLayout globalSetLayout);
        ~PointLightSystem() { vkDestroyPipelineLayout(m_device.device(), m_pipelineLayout, nullptr); }

        PointLightSystem(const PointLightSystem&) = delete;
        PointLightSystem& operator=(const PointLightSystem&) = delete;

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

    private:

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

        Device &m_device;

        std::unique_ptr<Shaders> m_shaders;
        VkPipelineLayout m_pipelineLayout{nullptr};

    }; // class PointLightSystem

} // namespace ven
```

7.35 /home/runner/work/VEngine/VEngine/include/VEngine/System/RenderSystem.hpp File Reference

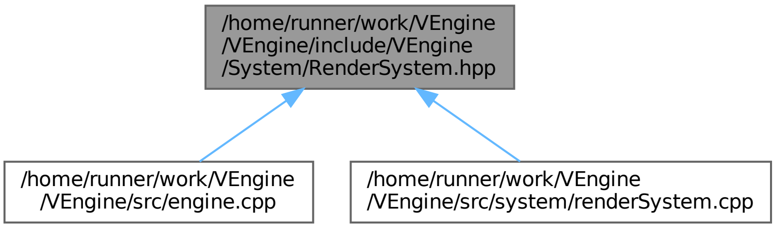
This file contains the RenderSystem class.

```cpp
#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

7.35.1 Detailed Description

This file contains the RenderSystem class.

Definition in file RenderSystem.hpp.

7.36 RenderSystem.hpp

Go to the documentation of this file.

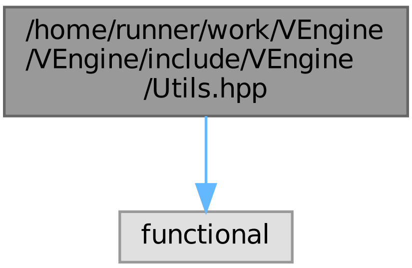
```
///
/// @file RenderSystem.hpp
/// @brief This file contains the RenderSystem class
/// @namespace ven
///

#pragma once

#include <memory>

#include <vulkan/vulkan.h>

#include "VEngine/Device.hpp"
#include "VEngine/Shaders.hpp"
#include "VEngine/FrameInfo.hpp"

namespace ven {

    struct SimplePushConstantData {
        glm::mat4 modelMatrix{1.F};
        glm::mat4 normalMatrix{1.F};
    };

    ///
```

```
    /// @class RenderSystem
    /// @brief Class for render system
    /// @namespace ven
    ///
    class RenderSystem {

        public:

            explicit RenderSystem(Device& device, VkRenderPass renderPass, VkDescriptorSetLayout globalSetLayout);
            ~RenderSystem() { vkDestroyPipelineLayout(m_device.device(), m_pipelineLayout, nullptr); }

            RenderSystem(const RenderSystem&) = delete;
            RenderSystem& operator=(const RenderSystem&) = delete;

            void renderObjects(const FrameInfo &frameInfo) const;

        private:

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

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

    }; // class RenderSystem

} // namespace ven
```

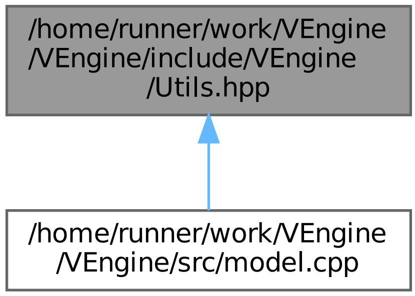
7.37 /home/runner/work/VEngine/VEngine/include/VEngine/Utils.hpp File Reference

`#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)

7.38 Utils.hpp

Go to the documentation of this file.

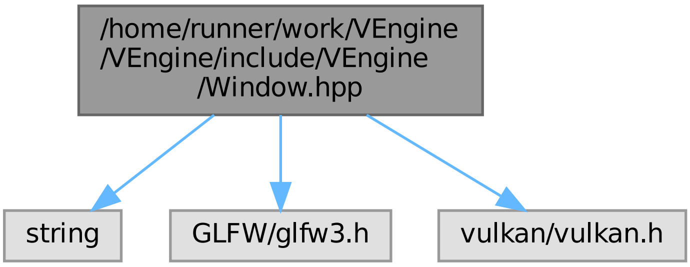
```
///
/// @file Utils.hpp
/// @brief
/// @namespace ven
///

#pragma once

#include <functional>

namespace ven {

    template<typename T, typename... Rest>
    void hashCombine(std::size_t& seed, const T& v, const Rest&... rest) {
        seed ^= std::hash<T>{}(v) + 0x9e3779b9 + (seed « 6) + (seed » 2);
        (hashCombine(seed, rest), ...);
    }

} // namespace ven
```

7.39 /home/runner/work/VEngine/VEngine/include/VEngine/Window.hpp File Reference

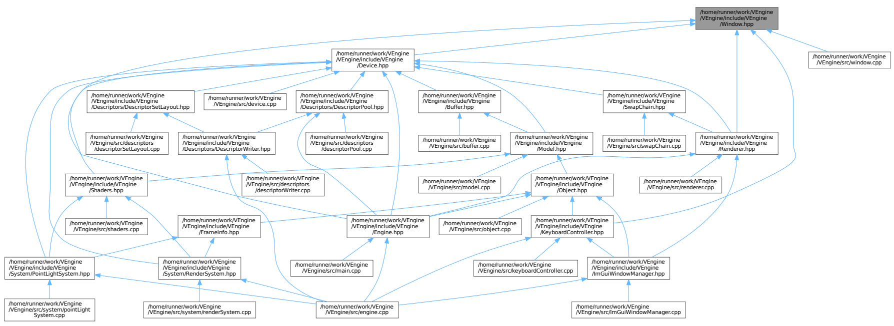
This file contains the Window class.

```
#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

  Class for window.

Namespaces

- namespace ven

Macros

- #define GLFW_INCLUDE_VULKAN

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"

7.39.1 Detailed Description

This file contains the Window class.

Definition in file Window.hpp.

7.39.2 Macro Definition Documentation

7.39.2.1 GLFW_INCLUDE_VULKAN

```
#define GLFW_INCLUDE_VULKAN
```

Definition at line 11 of file Window.hpp.

7.40 Window.hpp

Go to the documentation of this file.

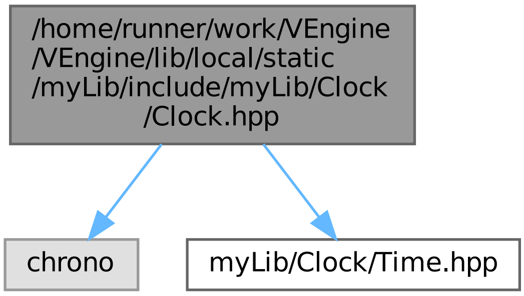
```
00001 ///
00002 /// @file Window.hpp
00003 /// @brief This file contains the Window class
00004 /// @namespace ven
00005 ///
00006
00007 #pragma once
00008
00009 #include <string>
00010
00011 #define GLFW_INCLUDE_VULKAN
00012 #include <GLFW/glfw3.h>
00013 #include <vulkan/vulkan.h>
00014
00015 namespace ven {
00016
00017     static constexpr uint32_t DEFAULT_WIDTH = 1920;
00018     static constexpr uint32_t DEFAULT_HEIGHT = 1080;
00019     static constexpr std::string_view DEFAULT_TITLE = "VEngine";
00020
00021     ///
00022     /// @class Window
00023     /// @brief Class for window
00024     /// @namespace ven
00025     ///
00026     class Window {
00027
00028         public:
00029
00030             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) {};
00031             ~Window() { glfwDestroyWindow(m_window); glfwTerminate(); m_window = nullptr;};
00032
00033             [[nodiscard]] GLFWwindow* createWindow(uint32_t width, uint32_t height, const std::string
      &title);
00034             void createWindowSurface(VkInstance instance, VkSurfaceKHR* surface) const;
00035
00036             [[nodiscard]] GLFWwindow* getGLFWindow() const { return m_window; };
00037
00038             [[nodiscard]] VkExtent2D getExtent() const { return {m_width, m_height}; };
00039             [[nodiscard]] bool wasWindowResized() const { return m_framebufferResized; }
00040             void resetWindowResizedFlag() { m_framebufferResized = false; }
00041
00042         private:
00043
00044             static void framebufferResizeCallback(GLFWwindow* window, int width, int height);
00045
00046             GLFWwindow* m_window{nullptr};
00047             uint32_t m_width;
00048             uint32_t m_height;
00049
00050             bool m_framebufferResized = false;
00051
00052     }; // class Window
00053
00054 } // namespace ven
```

7.41 /home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/Clock/Clock.hpp File Reference

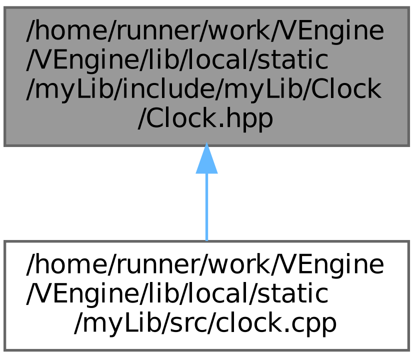
Clock class for time management.

```
#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.

7.41.1 Detailed Description

Clock class for time management.

Definition in file Clock.hpp.

7.41.2 Typedef Documentation

7.41.2.1 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.

7.42 Clock.hpp

Go to the documentation of this file.

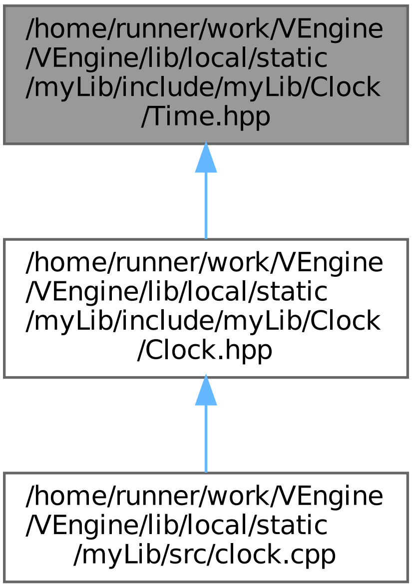
```
00001 ///
00002 /// @file Clock.hpp
00003 /// @brief Clock class for time management
00004 /// @namespace myLib
00005 ///
00006
00007 #pragma once
00008
00009 #include <chrono>
00010
00011 #include "myLib/Clock/Time.hpp"
00012
00013 ///
00014 /// @brief TimePoint is a type alias for a time point 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
```

```
            ///
            void restart() { m_start = std::chrono::high_resolution_clock::now(); };

            ///
            /// @brief Pause the clock
            ///
            void pause();

            ///
            /// @brief Resume the clock
            ///
            void resume();

            ///
            /// @brief Get the elapsed time since the last restart
            /// @return Time The elapsed time
            ///
            [[nodiscard]] Time getElapsedTime() const;

        private:

            ///
            /// @property The start time
            ///
            TimePoint m_start;

            ///
            /// @property The pause time
            ///
            TimePoint m_pause;

            ///
            /// @property The "is in pause" boolean variable
            ///
            bool m_paused{false};

    }; // Clock

} // namespace myLib
```

7.43 /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

Variables

- static constexpr unsigned int myLib::MICROSECONDS_PER_SECOND = 1000000
- static constexpr unsigned int myLib::MILLISECONDS_PER_SECOND = 1000

7.43.1 Detailed Description

Class for time management.

Definition in file Time.hpp.

7.44 Time.hpp

Go to the documentation of this file.

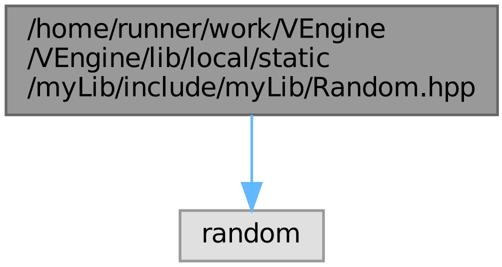
```
///
/// @file Time.hpp
/// @brief Class for time management
/// @namespace myLib
///

#pragma once

namespace myLib {

    static constexpr unsigned int MICROSECONDS_PER_SECOND = 1000000;
    static constexpr unsigned int MILLISECONDS_PER_SECOND = 1000;

    ///
    /// @class Time
    /// @brief Class used for time management
    ///
    class Time {

        public:

            ///
            /// @brief Construct a new Time object
            ///
            explicit Time(const double seconds) : m_seconds(seconds) {};

            ///
            /// @brief Transform the time to seconds
            /// @return int The time in seconds
            ///
            [[nodiscard]] int asSeconds() const { return static_cast<int>(m_seconds); };

            ///
            /// @brief Transform the time to milliseconds
            /// @return int The time in milliseconds
            ///
            [[nodiscard]] int asMilliseconds() const { return static_cast<int>(m_seconds * MILLISECONDS_PER_SECOND); }

            ///
            /// @brief Transform the time to microseconds
            /// @return int The time in microseconds
            ///
            [[nodiscard]] int asMicroseconds() const { return static_cast<int>(m_seconds * MICROSECONDS_PER_SECOND); };

        private:

            ///
            /// @property The time in seconds
            ///
            double m_seconds{0.0F};

    }; // Time

} // namespace myLib
```

7.45 /home/runner/work/VEngine/VEngine/lib/local/static/myLib/include/myLib/Random.hpp File Reference

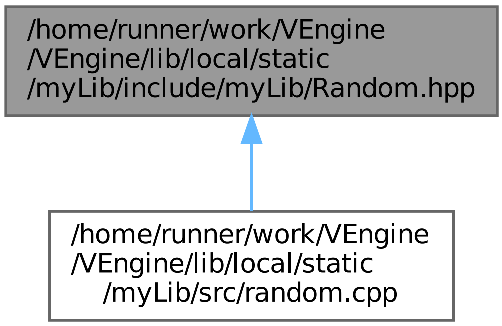
Class for random number generation.

`#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

Variables

- static constexpr int myLib::RANDOM_INT_MIN = -1000
- static constexpr int myLib::RANDOM_INT_MAX = 1000
- static constexpr float myLib::RANDOM_FLOAT_MAX = 1000.0F

7.45.1 Detailed Description

Class for random number generation.

Definition in file Random.hpp.

7.46 Random.hpp

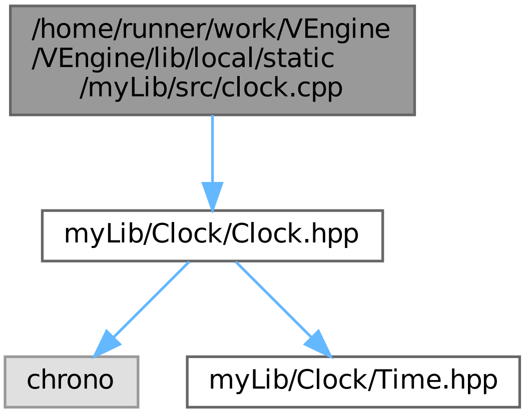
Go to the documentation of this file.

```
///
/// @file Random.hpp
/// @brief Class for random number generation
/// @namespace myLib
///

#pragma once

#include <random>

namespace myLib {

    static constexpr int RANDOM_INT_MIN = -1000;
    static constexpr int RANDOM_INT_MAX = 1000;
    static constexpr float RANDOM_FLOAT_MAX = 1000.0F;

    ///
    /// @class Random
    /// @brief Class for random number generation
    ///
    class Random {

        public:

            ///
            /// @brief Generate a random integer between min and max
            /// @param min The minimum value
            /// @param max The maximum value
            /// @return int The random integer
            ///
            static int randomInt(int min, int max);
            static int randomInt() { return randomInt(-1000, 1000); };

            ///
            /// @param min The minimum value
            /// @param max The maximum value
            /// @return float The random float
            ///
            static float randomFloat(float min, float max);
            static float randomFloat() { return randomFloat(-1.0F, 1.0F); };

    }; // class Random

} // namespace myLib
```

7.47 /home/runner/work/VEngine/VEngine/lib/local/static/myLib/src/clock.cpp File Reference

`#include "myLib/Clock/Clock.hpp"`

Include dependency graph for clock.cpp:



7.48 clock.cpp

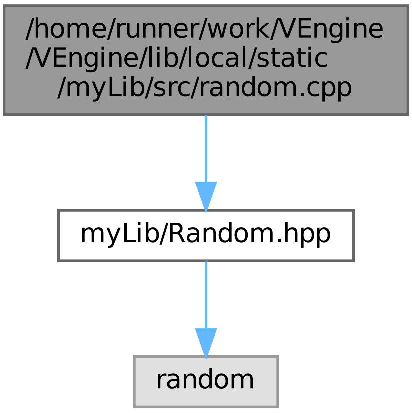
Go to the documentation of this file.

```
#include "myLib/Clock/Clock.hpp"

void myLib::Clock::pause()
{
    if (m_paused) {
        return;
    }
    m_pause = std::chrono::high_resolution_clock::now();
    m_paused = true;
}

void myLib::Clock::resume()
{
    if (!m_paused) {
        return;
    }

    m_start += std::chrono::high_resolution_clock::now() - m_pause;
    m_paused = false;
}

myLib::Time myLib::Clock::getElapsedTime() const
{
    TimePoint now = std::chrono::high_resolution_clock::now();
    std::chrono::duration<float> elapsed_time{};
    if (m_paused) {
        elapsed_time = m_pause - m_start;
    } else {
        elapsed_time = now - m_start;
    }
    return Time(elapsed_time.count());
}
```

7.49 /home/runner/work/VEngine/VEngine/lib/local/static/myLib/src/random.cpp File Reference

```
#include "myLib/Random.hpp"
```

Include dependency graph for random.cpp:



7.50 random.cpp

Go to the documentation of this file.

```
#include "myLib/Random.hpp"

int myLib::Random::randomInt(const int min, const int max)
{
    std::mt19937 gen(std::random_device{}());
    std::uniform_int_distribution<> dis(min, max);
    return dis(gen);
}

float myLib::Random::randomFloat(const float min, const float max)
{
    return min + (static_cast<float>(randomInt(RANDOM_INT_MIN, RANDOM_INT_MAX)) / RANDOM_FLOAT_MAX *
  (max - min));
}
```

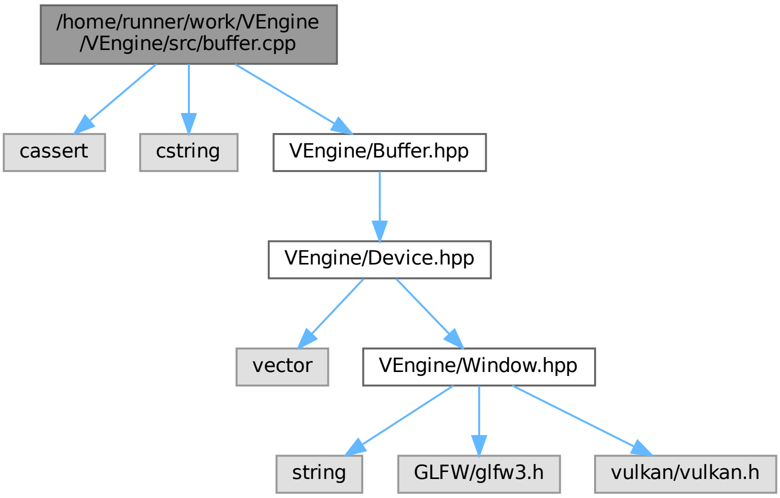
7.51 /home/runner/work/VEngine/VEngine/README.md File Reference

7.52 /home/runner/work/VEngine/VEngine/src/buffer.cpp File Reference

```
#include <cassert>
#include <cstring>
```

```
#include "VEngine/Buffer.hpp"
```

Include dependency graph for buffer.cpp:



7.53 buffer.cpp

Go to the documentation of this file.

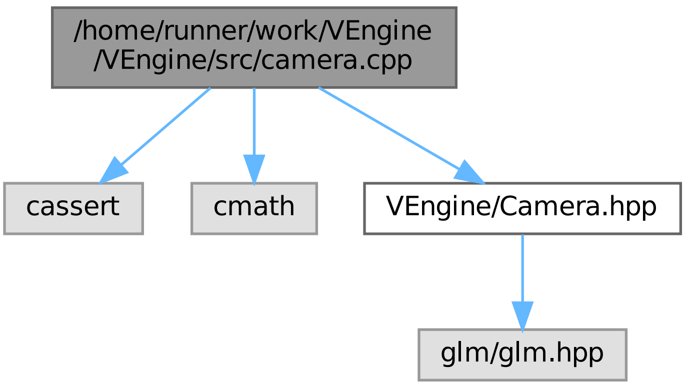
```
#include <cassert>
#include <cstring>

#include "VEngine/Buffer.hpp"

VkDeviceSize ven::Buffer::getAlignment(const VkDeviceSize instanceSize, const VkDeviceSize
      minOffsetAlignment) {
    if (minOffsetAlignment > 0) {
        return (instanceSize + minOffsetAlignment - 1) & ~(minOffsetAlignment - 1);
    }
    return instanceSize;
}

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}
{
    m_bufferSize = m_alignmentSize * m_instanceCount;
    device.createBuffer(m_bufferSize, m_usageFlags, m_memoryPropertyFlags, m_buffer, m_memory);
}

ven::Buffer::~Buffer()
{
    unmap();
    vkDestroyBuffer(m_device.device(), m_buffer, nullptr);
    vkFreeMemory(m_device.device(), m_memory, nullptr);
}

VkResult ven::Buffer::map(const VkDeviceSize size, const VkDeviceSize offset)
{
    assert(m_buffer && m_memory && "Called map on m_buffer before create");
    return vkMapMemory(m_device.device(), m_memory, offset, size, 0, &m_mapped);
}

void ven::Buffer::unmap()
{
    if (m_mapped != nullptr) {
        vkUnmapMemory(m_device.device(), m_memory);
        m_mapped = nullptr;
```

```
    }
}

void ven::Buffer::writeToBuffer(const void *data, const VkDeviceSize size, const VkDeviceSize offset)
const
{
    assert(m_mapped && "Cannot copy to unmapped m_buffer");

    if (size == VK_WHOLE_SIZE) {
        memcpy(m_mapped, data, m_bufferSize);
    } else {
        char *memOffset = static_cast<char *>(m_mapped);
        memOffset += offset;
        memcpy(memOffset, data, size);
    }
}

VkResult ven::Buffer::flush(const VkDeviceSize size, const VkDeviceSize offset) const
{
    VkMappedMemoryRange mappedRange = {};
    mappedRange.sType = VK_STRUCTURE_TYPE_MAPPED_MEMORY_RANGE;
    mappedRange.memory = m_memory;
    mappedRange.offset = offset;
    mappedRange.size = size;
    return vkFlushMappedMemoryRanges(m_device.device(), 1, &mappedRange);
}

VkResult ven::Buffer::invalidate(const VkDeviceSize size, const VkDeviceSize offset) const
{
    VkMappedMemoryRange mappedRange = {};
    mappedRange.sType = VK_STRUCTURE_TYPE_MAPPED_MEMORY_RANGE;
    mappedRange.memory = m_memory;
    mappedRange.offset = offset;
    mappedRange.size = size;
    return vkInvalidateMappedMemoryRanges(m_device.device(), 1, &mappedRange);
}
```

7.54 /home/runner/work/VEngine/VEngine/src/camera.cpp File Reference

```
#include <cassert>
#include <cmath>
#include "VEngine/Camera.hpp"
```

Include dependency graph for camera.cpp:



7.55 camera.cpp

Go to the documentation of this file.

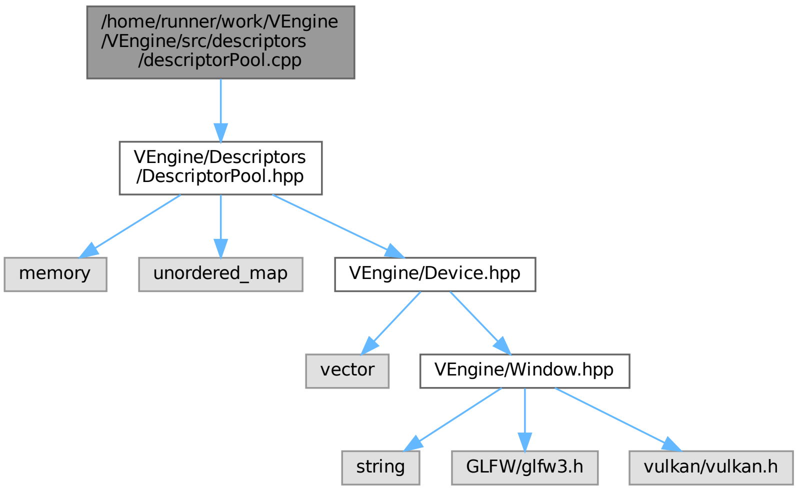
```cpp
#include <cassert>
#include <cmath>

#include "VEngine/Camera.hpp"

void ven::Camera::setOrthographicProjection(const float left, const float right, const float top, const float bottom, const float near, const float far)
{
    m_projectionMatrix = glm::mat4{1.0F};
    m_projectionMatrix[0][0] = 2.F / (right - left);
    m_projectionMatrix[1][1] = 2.F / (bottom - top);
    m_projectionMatrix[2][2] = 1.F / (far - near);
    m_projectionMatrix[3][0] = -(right + left) / (right - left);
    m_projectionMatrix[3][1] = -(bottom + top) / (bottom - top);
    m_projectionMatrix[3][2] = -near / (far - near);
}

void ven::Camera::setPerspectiveProjection(const float aspect)
{
    assert(glm::abs(aspect - std::numeric_limits<float>::epsilon()) > 0.0F);
    const float tanHalfFov = std::tan(m_fov / 2.F);
    m_projectionMatrix = glm::mat4{0.0F};
    m_projectionMatrix[0][0] = 1.F / (aspect * tanHalfFov);
    m_projectionMatrix[1][1] = 1.F / (tanHalfFov);
    m_projectionMatrix[2][2] = m_far / (m_far - m_near);
    m_projectionMatrix[2][3] = 1.F;
    m_projectionMatrix[3][2] = -(m_far * m_near) / (m_far - m_near);
}

void ven::Camera::setViewDirection(const glm::vec3 position, const glm::vec3 direction, const glm::vec3 up)
{
    const glm::vec3 w{normalize(direction)};
    const glm::vec3 u{normalize(cross(w, up))};
    const glm::vec3 v{cross(w, u)};

    m_viewMatrix = glm::mat4{1.F};
    m_viewMatrix[0][0] = u.x;
    m_viewMatrix[1][0] = u.y;
    m_viewMatrix[2][0] = u.z;
    m_viewMatrix[0][1] = v.x;
    m_viewMatrix[1][1] = v.y;
    m_viewMatrix[2][1] = v.z;
    m_viewMatrix[0][2] = w.x;
    m_viewMatrix[1][2] = w.y;
    m_viewMatrix[2][2] = w.z;
    m_viewMatrix[3][0] = -dot(u, position);
    m_viewMatrix[3][1] = -dot(v, position);
    m_viewMatrix[3][2] = -dot(w, position);

    m_inverseViewMatrix = glm::mat4{1.F};
    m_inverseViewMatrix[0][0] = u.x;
    m_inverseViewMatrix[0][1] = u.y;
    m_inverseViewMatrix[0][2] = u.z;
    m_inverseViewMatrix[1][0] = v.x;
    m_inverseViewMatrix[1][1] = v.y;
    m_inverseViewMatrix[1][2] = v.z;
    m_inverseViewMatrix[2][0] = w.x;
    m_inverseViewMatrix[2][1] = w.y;
    m_inverseViewMatrix[2][2] = w.z;
    m_inverseViewMatrix[3][0] = position.x;
    m_inverseViewMatrix[3][1] = position.y;
    m_inverseViewMatrix[3][2] = position.z;
}

void ven::Camera::setViewYXZ(const glm::vec3 position, const glm::vec3 rotation)
{
    const float c3 = glm::cos(rotation.z);
    const float s3 = glm::sin(rotation.z);
    const float c2 = glm::cos(rotation.x);
    const float s2 = glm::sin(rotation.x);
    const float c1 = glm::cos(rotation.y);
    const float s1 = glm::sin(rotation.y);
    const glm::vec3 u{(c1 * c3 + s1 * s2 * s3), (c2 * s3), (c1 * s2 * s3 - c3 * s1)};
    const glm::vec3 v{(c3 * s1 * s2 - c1 * s3), (c2 * c3), (c1 * c3 * s2 + s1 * s3)};
    const glm::vec3 w{(c2 * s1), (-s2), (c1 * c2)};
    m_viewMatrix = glm::mat4{1.F};
    m_viewMatrix[0][0] = u.x;
    m_viewMatrix[1][0] = u.y;
    m_viewMatrix[2][0] = u.z;
    m_viewMatrix[0][1] = v.x;
    m_viewMatrix[1][1] = v.y;
```

```
    m_viewMatrix[2][1] = v.z;
    m_viewMatrix[0][2] = w.x;
    m_viewMatrix[1][2] = w.y;
    m_viewMatrix[2][2] = w.z;
    m_viewMatrix[3][0] = -dot(u, position);
    m_viewMatrix[3][1] = -dot(v, position);
    m_viewMatrix[3][2] = -dot(w, position);

    m_inverseViewMatrix = glm::mat4{1.F};
    m_inverseViewMatrix[0][0] = u.x;
    m_inverseViewMatrix[0][1] = u.y;
    m_inverseViewMatrix[0][2] = u.z;
    m_inverseViewMatrix[1][0] = v.x;
    m_inverseViewMatrix[1][1] = v.y;
    m_inverseViewMatrix[1][2] = v.z;
    m_inverseViewMatrix[2][0] = w.x;
    m_inverseViewMatrix[2][1] = w.y;
    m_inverseViewMatrix[2][2] = w.z;
    m_inverseViewMatrix[3][0] = position.x;
    m_inverseViewMatrix[3][1] = position.y;
    m_inverseViewMatrix[3][2] = position.z;
}
```

7.56 /home/runner/work/VEngine/VEngine/src/descriptors/descriptor←Pool.cpp File Reference

`#include "VEngine/Descriptors/DescriptorPool.hpp"`

Include dependency graph for descriptorPool.cpp:



7.57 descriptorPool.cpp

Go to the documentation of this file.

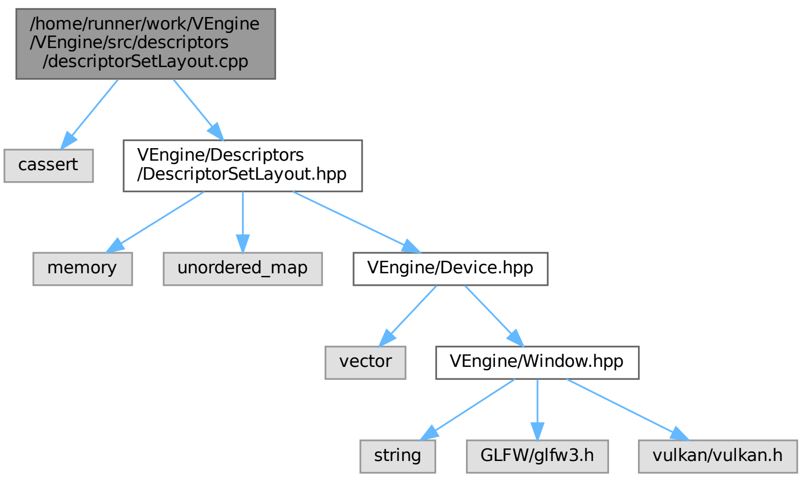
```
#include "VEngine/Descriptors/DescriptorPool.hpp"

ven::DescriptorPool::Builder &ven::DescriptorPool::Builder::addPoolSize(const VkDescriptorType
    descriptorType, const uint32_t count)
{
    m_poolSizes.push_back({descriptorType, count});
    return *this;
}

```

```
ven::DescriptorPool::Builder &ven::DescriptorPool::Builder::setPoolFlags(const VkDescriptorPoolCreateFlags flags)
{
    m_poolFlags = flags;
    return *this;
}
ven::DescriptorPool::Builder &ven::DescriptorPool::Builder::setMaxSets(const uint32_t count)
{
    m_maxSets = count;
    return *this;
}

ven::DescriptorPool::DescriptorPool(Device &device, const uint32_t maxSets, const VkDescriptorPoolCreateFlags poolFlags, const std::vector<VkDescriptorPoolSize> &poolSizes) : m_device{device}
{
    VkDescriptorPoolCreateInfo descriptorPoolInfo{};
    descriptorPoolInfo.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_POOL_CREATE_INFO;
    descriptorPoolInfo.poolSizeCount = static_cast<uint32_t>(poolSizes.size());
    descriptorPoolInfo.pPoolSizes = poolSizes.data();
    descriptorPoolInfo.maxSets = maxSets;
    descriptorPoolInfo.flags = poolFlags;

    if (vkCreateDescriptorPool(m_device.device(), &descriptorPoolInfo, nullptr, &m_descriptorPool) !=
        VK_SUCCESS) {
        throw std::runtime_error("failed to create descriptor pool!");
    }
}

bool ven::DescriptorPool::allocateDescriptor(const VkDescriptorSetLayout descriptorSetLayout, VkDescriptorSet &descriptor) const
{
    VkDescriptorSetAllocateInfo allocInfo{};
    allocInfo.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_ALLOCATE_INFO;
    allocInfo.descriptorPool = m_descriptorPool;
    allocInfo.pSetLayouts = &descriptorSetLayout;
    allocInfo.descriptorSetCount = 1;

    // Might want to create a "DescriptorPoolManager" class that handles this case, and builds
    // a new pool whenever an old pool fills up. But this is beyond our current scope
    return vkAllocateDescriptorSets(m_device.device(), &allocInfo, &descriptor) == VK_SUCCESS;
}
```

7.58 /home/runner/work/VEngine/VEngine/src/descriptors/descriptorSetLayout.cpp File Reference

```
#include <cassert>
#include "VEngine/Descriptors/DescriptorSetLayout.hpp"
```

Include dependency graph for descriptorSetLayout.cpp:



7.59 descriptorSetLayout.cpp

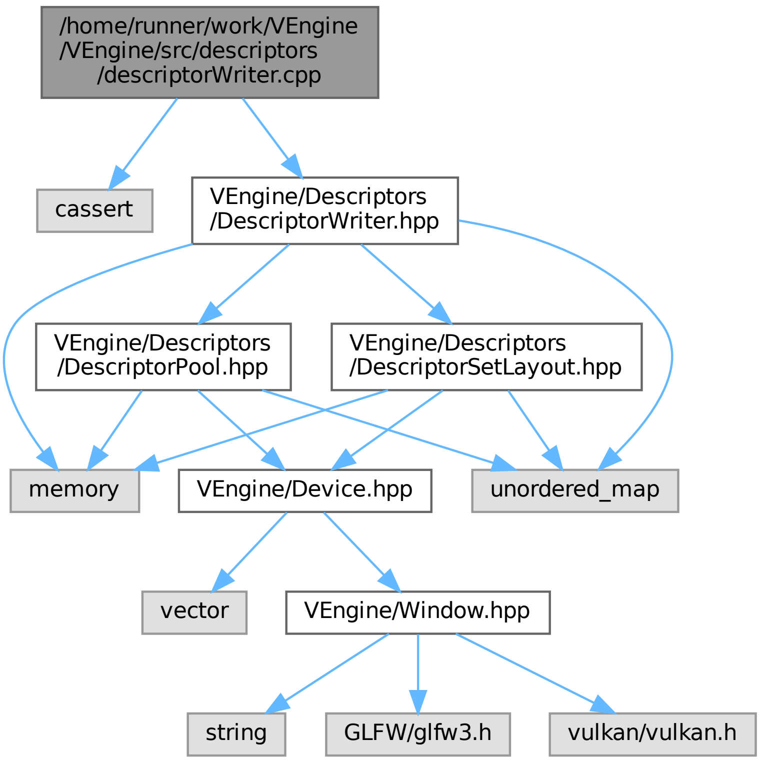
Go to the documentation of this file.

```cpp
#include <cassert>

#include "VEngine/Descriptors/DescriptorSetLayout.hpp"

ven::DescriptorSetLayout::Builder &ven::DescriptorSetLayout::Builder::addBinding(const uint32_t binding, const VkDescriptorType descriptorType, const VkShaderStageFlags stageFlags, const uint32_t count)
{
    assert(m_bindings.contains(binding) == 0 && "Binding already exists in layout");
    VkDescriptorSetLayoutBinding layoutBinding{};
    layoutBinding.binding = binding;
    layoutBinding.descriptorType = descriptorType;
    layoutBinding.descriptorCount = count;
    layoutBinding.stageFlags = stageFlags;
    m_bindings[binding] = layoutBinding;
    return *this;
}

ven::DescriptorSetLayout::DescriptorSetLayout(Device &device, const std::unordered_map<uint32_t, VkDescriptorSetLayoutBinding>& bindings) : m_device{device}, m_bindings{bindings}
{
    std::vector<VkDescriptorSetLayoutBinding> setLayoutBindings{};
    setLayoutBindings.reserve(bindings.size());
for (auto kv : bindings) {
        setLayoutBindings.push_back(kv.second);
    }

    VkDescriptorSetLayoutCreateInfo descriptorSetLayoutInfo{};
    descriptorSetLayoutInfo.sType = VK_STRUCTURE_TYPE_DESCRIPTOR_SET_LAYOUT_CREATE_INFO;
    descriptorSetLayoutInfo.bindingCount = static_cast<uint32_t>(setLayoutBindings.size());
    descriptorSetLayoutInfo.pBindings = setLayoutBindings.data();

    if (vkCreateDescriptorSetLayout(
            m_device.device(),
            &descriptorSetLayoutInfo,
            nullptr,
            &m_descriptorSetLayout) != VK_SUCCESS) {
        throw std::runtime_error("failed to create descriptor set layout!");
    }
}
```

7.60 /home/runner/work/VEngine/VEngine/src/descriptors/descriptorWriter.cpp File Reference

```cpp
#include <cassert>
#include "VEngine/Descriptors/DescriptorWriter.hpp"
```

Include dependency graph for descriptorWriter.cpp:



7.61 descriptorWriter.cpp

Go to the documentation of this file.

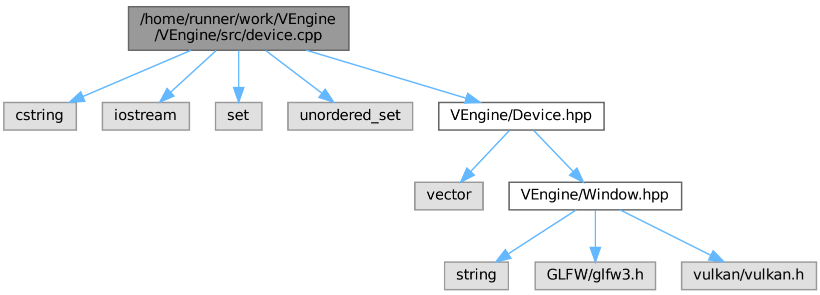
```
#include <cassert>

#include "VEngine/Descriptors/DescriptorWriter.hpp"

ven::DescriptorWriter &ven::DescriptorWriter::writeBuffer(const uint32_t binding, const VkDescriptorBufferInfo *bufferInfo)
{
    assert(m_setLayout.m_bindings.count(binding) == 1 && "Layout does not contain specified binding");

    const auto &bindingDescription = m_setLayout.m_bindings[binding];

    assert(bindingDescription.descriptorCount == 1 && "Binding single descriptor info, but binding expects multiple");

    VkWriteDescriptorSet write{};
    write.sType = VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET;
    write.descriptorType = bindingDescription.descriptorType;
    write.dstBinding = binding;
    write.pBufferInfo = bufferInfo;
    write.descriptorCount = 1;

    m_writes.push_back(write);
    return *this;
}

ven::DescriptorWriter &ven::DescriptorWriter::writeImage(const uint32_t binding, const VkDescriptorImageInfo *imageInfo)
{
    assert(m_setLayout.m_bindings.count(binding) == 1 && "Layout does not contain specified binding");

```

```
    const VkDescriptorSetLayoutBinding &bindingDescription = m_setLayout.m_bindings[binding];

    assert(bindingDescription.descriptorCount == 1 && "Binding single descriptor info, but binding expects multiple");

    VkWriteDescriptorSet write{};
    write.sType = VK_STRUCTURE_TYPE_WRITE_DESCRIPTOR_SET;
    write.descriptorType = bindingDescription.descriptorType;
    write.dstBinding = binding;
    write.pImageInfo = imageInfo;
    write.descriptorCount = 1;

    m_writes.push_back(write);
    return *this;
}

bool ven::DescriptorWriter::build(VkDescriptorSet &set)
{
    if (!m_pool.allocateDescriptor(m_setLayout.getDescriptorSetLayout(), set)) {
        return false;
    }
    overwrite(set);
    return true;
}

void ven::DescriptorWriter::overwrite(const VkDescriptorSet &set)
{
    for (auto &write : m_writes) {
        write.dstSet = set;
    }
    vkUpdateDescriptorSets(m_pool.m_device.device(), static_cast<unsigned int>(m_writes.size()), m_writes.data(), 0, nullptr);
}
```

7.62 /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 VkDebugUtilsMessageSeverityFlagBitsEXT messageSeverity, const VkDebugUtilsMessageTypeFlagsEXT messageType, const VkDebugUtilsMessengerCallbackDataEXT *pCallbackData, void *pUserData)
- VkResult CreateDebugUtilsMessengerEXT (const VkInstance instance, const VkDebugUtilsMessengerCreateInfoEXT *pCreateInfo, const VkAllocationCallbacks *pAllocator, VkDebugUtilsMessengerEXT *pDebugMessenger)
- void DestroyDebugUtilsMessengerEXT (const VkInstance instance, const VkDebugUtilsMessengerEXT debugMessenger, const VkAllocationCallbacks *pAllocator)

7.62.1 Function Documentation

7.62.1.1 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:

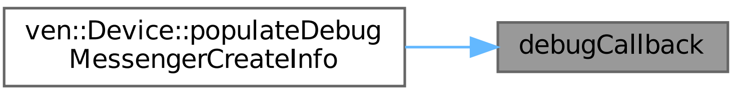
7.62.1.2 debugCallback()

```
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:



7.62.1.3 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:

7.63 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();
```

```
    createLogicalDevice();
    createCommandPool();
}

ven::Device::~Device()
{
    vkDestroyCommandPool(m_device, m_commandPool, nullptr);
    vkDestroyDevice(m_device, nullptr);

    if (enableValidationLayers) {
        DestroyDebugUtilsMessengerEXT(m_instance, m_debugMessenger, nullptr);
    }

    vkDestroySurfaceKHR(m_instance, m_surface, nullptr);
    vkDestroyInstance(m_instance, nullptr);
}

void ven::Device::createInstance()
{
    if (enableValidationLayers && !checkValidationLayerSupport()) {
        throw std::runtime_error("validation layers requested, but not available!");
    }

    VkApplicationInfo appInfo = {};
    appInfo.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
    appInfo.pApplicationName = "LittleVulkanEngine App";
    appInfo.applicationVersion = VK_MAKE_VERSION(1, 0, 0);
    appInfo.pEngineName = "No Engine";
    appInfo.engineVersion = VK_MAKE_VERSION(1, 0, 0);
    appInfo.apiVersion = VK_API_VERSION_1_0;

    VkInstanceCreateInfo createInfo = {};
    createInfo.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
    createInfo.pApplicationInfo = &appInfo;

    std::vector<const char *> extensions = getRequiredExtensions();
    createInfo.enabledExtensionCount = static_cast<uint32_t>(extensions.size());
    createInfo.ppEnabledExtensionNames = extensions.data();

    VkDebugUtilsMessengerCreateInfoEXT debugCreateInfo;
    if (enableValidationLayers) {
        createInfo.enabledLayerCount = static_cast<uint32_t>(validationLayers.size());
        createInfo.ppEnabledLayerNames = validationLayers.data();

        populateDebugMessengerCreateInfo(debugCreateInfo);
        createInfo.pNext = &debugCreateInfo;
    } else {
        createInfo.enabledLayerCount = 0;
        createInfo.pNext = nullptr;
    }

    if (vkCreateInstance(&createInfo, nullptr, &m_instance) != VK_SUCCESS) {
        throw std::runtime_error("failed to create instance!");
    }

    hasGlfwRequiredInstanceExtensions();
}

void ven::Device::pickPhysicalDevice()
{
    uint32_t deviceCount = 0;
    vkEnumeratePhysicalDevices(m_instance, &deviceCount, nullptr);
    if (deviceCount == 0) {
        throw std::runtime_error("failed to find GPUs with Vulkan support!");
    }
    std::cout « "Device count: " « deviceCount « '\n';
    std::vector<VkPhysicalDevice> devices(deviceCount);
    vkEnumeratePhysicalDevices(m_instance, &deviceCount, devices.data());

    for (const auto &device : devices) {
        if (isDeviceSuitable(device)) {
            m_physicalDevice = device;
            break;
        }
    }

    if (m_physicalDevice == VK_NULL_HANDLE) {
        throw std::runtime_error("failed to find a suitable GPU!");
    }

    vkGetPhysicalDeviceProperties(m_physicalDevice, &properties_);
    std::cout « "physical device: " « properties_.deviceName « '\n';
}

void ven::Device::createLogicalDevice()
{
    const QueueFamilyIndices indices = findQueueFamilies(m_physicalDevice);
```

```

    std::vector<VkDeviceQueueCreateInfo> queueCreateInfos;
    const std::set<uint32_t> uniqueQueueFamilies = {indices.graphicsFamily, indices.presentFamily};
    float queuePriority = 1.0F;

    for (const uint32_t queueFamily : uniqueQueueFamilies) {
        VkDeviceQueueCreateInfo queueCreateInfo = {};
        queueCreateInfo.sType = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO;
        queueCreateInfo.queueFamilyIndex = queueFamily;
        queueCreateInfo.queueCount = 1;
        queueCreateInfo.pQueuePriorities = &queuePriority;
        queueCreateInfos.push_back(queueCreateInfo);
    }

    VkPhysicalDeviceFeatures deviceFeatures = {};
    deviceFeatures.samplerAnisotropy = VK_TRUE;

    VkDeviceCreateInfo createInfo = {};
    createInfo.sType = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;

    createInfo.queueCreateInfoCount = static_cast<uint32_t>(queueCreateInfos.size());
    createInfo.pQueueCreateInfos = queueCreateInfos.data();

    createInfo.pEnabledFeatures = &deviceFeatures;
    createInfo.enabledExtensionCount = static_cast<uint32_t>(deviceExtensions.size());
    createInfo.ppEnabledExtensionNames = deviceExtensions.data();

        // might not really be necessary anymore because device specific validation layers
        // have been deprecated
    if (enableValidationLayers) {
        createInfo.enabledLayerCount = static_cast<uint32_t>(validationLayers.size());
        createInfo.ppEnabledLayerNames = validationLayers.data();
    } else {
        createInfo.enabledLayerCount = 0;
    }

    if (vkCreateDevice(m_physicalDevice, &createInfo, nullptr, &m_device) != VK_SUCCESS) {
        throw std::runtime_error("failed to create logical device!");
    }

    vkGetDeviceQueue(m_device, indices.graphicsFamily, 0, &m_graphicsQueue);
    vkGetDeviceQueue(m_device, indices.presentFamily, 0, &m_presentQueue);
}

void ven::Device::createCommandPool()
{
    const QueueFamilyIndices queueFamilyIndices = findPhysicalQueueFamilies();

    VkCommandPoolCreateInfo poolInfo = {};
    poolInfo.sType = VK_STRUCTURE_TYPE_COMMAND_POOL_CREATE_INFO;
    poolInfo.queueFamilyIndex = queueFamilyIndices.graphicsFamily;
    poolInfo.flags = VK_COMMAND_POOL_CREATE_TRANSIENT_BIT |
VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT;

    if (vkCreateCommandPool(m_device, &poolInfo, nullptr, &m_commandPool) != VK_SUCCESS) {
        throw std::runtime_error("failed to create command pool!");
    }
}

bool ven::Device::isDeviceSuitable(const VkPhysicalDevice device) const
{
    const QueueFamilyIndices indices = findQueueFamilies(device);
    const bool extensionsSupported = checkDeviceExtensionSupport(device);
    bool swapChainAdequate = false;

    if (extensionsSupported) {
        SwapChainSupportDetails swapChainSupport = querySwapChainSupport(device);
        swapChainAdequate = !swapChainSupport.formats.empty() &&
!swapChainSupport.presentModes.empty();
    }

    VkPhysicalDeviceFeatures supportedFeatures;
    vkGetPhysicalDeviceFeatures(device, &supportedFeatures);

    return indices.isComplete() && extensionsSupported && swapChainAdequate &&
(supportedFeatures.samplerAnisotropy != 0U);
}

void ven::Device::populateDebugMessengerCreateInfo(VkDebugUtilsMessengerCreateInfoEXT &createInfo)
{
    createInfo = {};
    createInfo.sType = VK_STRUCTURE_TYPE_DEBUG_UTILS_MESSENGER_CREATE_INFO_EXT;
    createInfo.messageSeverity = VK_DEBUG_UTILS_MESSAGE_SEVERITY_WARNING_BIT_EXT |
                                 VK_DEBUG_UTILS_MESSAGE_SEVERITY_ERROR_BIT_EXT;
    createInfo.messageType = VK_DEBUG_UTILS_MESSAGE_TYPE_GENERAL_BIT_EXT |
                             VK_DEBUG_UTILS_MESSAGE_TYPE_VALIDATION_BIT_EXT |
                             VK_DEBUG_UTILS_MESSAGE_TYPE_PERFORMANCE_BIT_EXT;
```

```
    createInfo.pfnUserCallback = debugCallback;
    createInfo.pUserData = nullptr;  // Optional
}

void ven::Device::setupDebugMessenger()
{
    if (!enableValidationLayers) { return; }
    VkDebugUtilsMessengerCreateInfoEXT createInfo;
    populateDebugMessengerCreateInfo(createInfo);
    if (CreateDebugUtilsMessengerEXT(m_instance, &createInfo, nullptr, &m_debugMessenger) != VK_SUCCESS) {
        throw std::runtime_error("failed to set up debug messenger!");
    }
}

bool ven::Device::checkValidationLayerSupport() const
{
    uint32_t layerCount = 0;
    vkEnumerateInstanceLayerProperties(&layerCount, nullptr);

    std::vector<VkLayerProperties> availableLayers(layerCount);
    vkEnumerateInstanceLayerProperties(&layerCount, availableLayers.data());

    for (const char *layerName : validationLayers) {
        bool layerFound = false;

        for (const auto &layerProperties : availableLayers) {
            if (strcmp(layerName, layerProperties.layerName) == 0) {
                layerFound = true;
                break;
            }
        }
        if (!layerFound) {
            return false;
        }
    }

    return true;
}

std::vector<const char *> ven::Device::getRequiredExtensions() const
{
    uint32_t glfwExtensionCount = 0;
    const char **glfwExtensions = nullptr;
    glfwExtensions = glfwGetRequiredInstanceExtensions(&glfwExtensionCount);

    std::vector<const char *> extensions(glfwExtensions, glfwExtensions + glfwExtensionCount);

    if (enableValidationLayers) {
        extensions.push_back(VK_EXT_DEBUG_UTILS_EXTENSION_NAME);
    }

    return extensions;
}

void ven::Device::hasGlfwRequiredInstanceExtensions() const
{
    uint32_t extensionCount = 0;
    vkEnumerateInstanceExtensionProperties(nullptr, &extensionCount, nullptr);
    std::vector<VkExtensionProperties> extensions(extensionCount);
    vkEnumerateInstanceExtensionProperties(nullptr, &extensionCount, extensions.data());

    std::cout « "available extensions:\n";
    std::unordered_set<std::string> available;
    for (const auto &extension : extensions) {
        std::cout « '\t' « extension.extensionName « '\n';
        available.insert(extension.extensionName);
    }

    std::cout « "required extensions:\n";
    const std::vector<const char *> requiredExtensions = getRequiredExtensions();
    for (const auto &required : requiredExtensions) {
        std::cout « "\t" « required « '\n';
        if (available.find(required) == available.end()) {
            throw std::runtime_error("Missing required glfw extension");
        }
    }
}

bool ven::Device::checkDeviceExtensionSupport(const VkPhysicalDevice device) const
{
    uint32_t extensionCount = 0;
    vkEnumerateDeviceExtensionProperties(device, nullptr, &extensionCount, nullptr);

    std::vector<VkExtensionProperties> availableExtensions(extensionCount);
    vkEnumerateDeviceExtensionProperties(device, nullptr, &extensionCount, availableExtensions.data());
```

```cpp

    std::set<std::string> requiredExtensions(deviceExtensions.begin(), deviceExtensions.end());
    for (const auto &extension : availableExtensions) {
        requiredExtensions.erase(extension.extensionName);
    }

    return requiredExtensions.empty();
}

ven::QueueFamilyIndices ven::Device::findQueueFamilies(const VkPhysicalDevice device) const
{
    QueueFamilyIndices indices;

    uint32_t queueFamilyCount = 0;
    vkGetPhysicalDeviceQueueFamilyProperties(device, &queueFamilyCount, nullptr);
    std::vector<VkQueueFamilyProperties> queueFamilies(queueFamilyCount);
    vkGetPhysicalDeviceQueueFamilyProperties(device, &queueFamilyCount, queueFamilies.data());
    uint32_t index = 0;

    for (const auto &queueFamily : queueFamilies) {
        if (queueFamily.queueCount > 0 && ((queueFamily.queueFlags & VK_QUEUE_GRAPHICS_BIT) != 0U)) {
            indices.graphicsFamily = index;
            indices.graphicsFamilyHasValue = true;
        }
        VkBool32 presentSupport = 0U;
        vkGetPhysicalDeviceSurfaceSupportKHR(device, index, m_surface, &presentSupport);
        if (queueFamily.queueCount > 0 && (presentSupport != 0U)) {
            indices.presentFamily = index;
            indices.presentFamilyHasValue = true;
        }
        if (indices.isComplete()) {
            break;
        }
        index++;
    }
    return indices;
}

ven::SwapChainSupportDetails ven::Device::querySwapChainSupport(const VkPhysicalDevice device) const
{
    SwapChainSupportDetails details;
    vkGetPhysicalDeviceSurfaceCapabilitiesKHR(device, m_surface, &details.capabilities);
    uint32_t formatCount = 0;

    vkGetPhysicalDeviceSurfaceFormatsKHR(device, m_surface, &formatCount, nullptr);
    if (formatCount != 0) {
        details.formats.resize(formatCount);
        vkGetPhysicalDeviceSurfaceFormatsKHR(device, m_surface, &formatCount, details.formats.data());
    }
    uint32_t presentModeCount = 0;
    vkGetPhysicalDeviceSurfacePresentModesKHR(device, m_surface, &presentModeCount, nullptr);
    if (presentModeCount != 0) {
        details.presentModes.resize(presentModeCount);
        vkGetPhysicalDeviceSurfacePresentModesKHR(device, m_surface, &presentModeCount, details.presentModes.data());
    }

    return details;
}

VkFormat ven::Device::findSupportedFormat(const std::vector<VkFormat> &candidates, const VkImageTiling tiling, const VkFormatFeatureFlags features) const
{
    for (const VkFormat format : candidates) {
        VkFormatProperties props;
        vkGetPhysicalDeviceFormatProperties(m_physicalDevice, format, &props);
        if (tiling == VK_IMAGE_TILING_LINEAR && (props.linearTilingFeatures & features) == features) {
            return format;
        } if (tiling == VK_IMAGE_TILING_OPTIMAL && (props.optimalTilingFeatures & features) == features) {
            return format;
        }
    }
    throw std::runtime_error("failed to find supported format!");
}

uint32_t ven::Device::findMemoryType(const uint32_t typeFilter, const VkMemoryPropertyFlags propertiesp) const
{
    VkPhysicalDeviceMemoryProperties memProperties;
    vkGetPhysicalDeviceMemoryProperties(m_physicalDevice, &memProperties);

    for (uint32_t i = 0; i < memProperties.memoryTypeCount; i++) {
        if (((typeFilter & (1 << i)) != 0U) &&
        (memProperties.memoryTypes[i].propertyFlags & propertiesp) == propertiesp) {
            return i;
        }
```

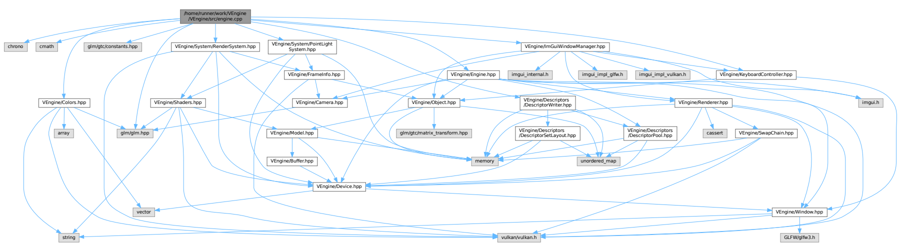
```
    }

    throw std::runtime_error("failed to find suitable m_memory type!");
}

void ven::Device::createBuffer(const VkDeviceSize size, const VkBufferUsageFlags usage, const
VkMemoryPropertyFlags propertiesp, VkBuffer &buffer, VkDeviceMemory &bufferMemory) const
{
    VkBufferCreateInfo bufferInfo{};
    bufferInfo.sType = VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO;
    bufferInfo.size = size;
    bufferInfo.usage = usage;
    bufferInfo.sharingMode = VK_SHARING_MODE_EXCLUSIVE;

    if (vkCreateBuffer(m_device, &bufferInfo, nullptr, &buffer) != VK_SUCCESS) {
        throw std::runtime_error("failed to create vertex m_buffer!");
    }

    VkMemoryRequirements memRequirements;
    vkGetBufferMemoryRequirements(m_device, buffer, &memRequirements);

    VkMemoryAllocateInfo allocInfo{};
    allocInfo.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
    allocInfo.allocationSize = memRequirements.size;
    allocInfo.memoryTypeIndex = findMemoryType(memRequirements.memoryTypeBits, propertiesp);

    if (vkAllocateMemory(m_device, &allocInfo, nullptr, &bufferMemory) != VK_SUCCESS) {
        throw std::runtime_error("failed to allocate vertex m_buffer m_memory!");
    }

    vkBindBufferMemory(m_device, buffer, bufferMemory, 0);
}

VkCommandBuffer ven::Device::beginSingleTimeCommands() const
{
    VkCommandBufferAllocateInfo allocInfo{};
    allocInfo.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO;
    allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
    allocInfo.commandPool = m_commandPool;
    allocInfo.commandBufferCount = 1;

    VkCommandBuffer commandBuffer = nullptr;
    vkAllocateCommandBuffers(m_device, &allocInfo, &commandBuffer);

    VkCommandBufferBeginInfo beginInfo{};
    beginInfo.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO;
    beginInfo.flags = VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT;

    vkBeginCommandBuffer(commandBuffer, &beginInfo);
    return commandBuffer;
}

void ven::Device::endSingleTimeCommands(const VkCommandBuffer commandBuffer) const
{
    vkEndCommandBuffer(commandBuffer);

    VkSubmitInfo submitInfo{};
    submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
    submitInfo.commandBufferCount = 1;
    submitInfo.pCommandBuffers = &commandBuffer;

    vkQueueSubmit(m_graphicsQueue, 1, &submitInfo, VK_NULL_HANDLE);
    vkQueueWaitIdle(m_graphicsQueue);

    vkFreeCommandBuffers(m_device, m_commandPool, 1, &commandBuffer);
}

void ven::Device::copyBuffer(const VkBuffer srcBuffer, const VkBuffer dstBuffer, const VkDeviceSize
size) const
{
    const VkCommandBuffer commandBuffer = beginSingleTimeCommands();

    VkBufferCopy copyRegion{};
    copyRegion.srcOffset = 0;  // Optional
    copyRegion.dstOffset = 0;  // Optional
    copyRegion.size = size;
    vkCmdCopyBuffer(commandBuffer, srcBuffer, dstBuffer, 1, &copyRegion);

    endSingleTimeCommands(commandBuffer);
}

void ven::Device::copyBufferToImage(const VkBuffer buffer, const VkImage image, const uint32_t width,
const uint32_t height, const uint32_t layerCount) const
{
    const VkCommandBuffer commandBuffer = beginSingleTimeCommands();

    VkBufferImageCopy region{};
```

```
    region.bufferOffset = 0;
    region.bufferRowLength = 0;
    region.bufferImageHeight = 0;

    region.imageSubresource.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
    region.imageSubresource.mipLevel = 0;
    region.imageSubresource.baseArrayLayer = 0;
    region.imageSubresource.layerCount = layerCount;

    region.imageOffset = {0, 0, 0};
    region.imageExtent = {width, height, 1};

    vkCmdCopyBufferToImage(commandBuffer, buffer, image, VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL, 1,
&region);
    endSingleTimeCommands(commandBuffer);
}

void ven::Device::createImageWithInfo(const VkImageCreateInfo &imageInfo, const VkMemoryPropertyFlags
properties, VkImage &image, VkDeviceMemory &imageMemory) const
{
    if (vkCreateImage(m_device, &imageInfo, nullptr, &image) != VK_SUCCESS) {
        throw std::runtime_error("failed to create image!");
    }

    VkMemoryRequirements memRequirements;
    vkGetImageMemoryRequirements(m_device, image, &memRequirements);

    VkMemoryAllocateInfo allocInfo{};
    allocInfo.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
    allocInfo.allocationSize = memRequirements.size;
    allocInfo.memoryTypeIndex = findMemoryType(memRequirements.memoryTypeBits, properties);

    if (vkAllocateMemory(m_device, &allocInfo, nullptr, &imageMemory) != VK_SUCCESS) {
        throw std::runtime_error("failed to allocate image m_memory!");
    }

    if (vkBindImageMemory(m_device, image, imageMemory, 0) != VK_SUCCESS) {
        throw std::runtime_error("failed to bind image m_memory!");
    }
}
```

7.64 /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/Descriptors/DescriptorWriter.hpp"
#include "VEngine/ImGuiWindowManager.hpp"
#include "VEngine/Colors.hpp"
```

Include dependency graph for engine.cpp:



7.65 engine.cpp

Go to the documentation of this file.

```
#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/Descriptors/DescriptorWriter.hpp"
#include "VEngine/ImGuiWindowManager.hpp"
#include "VEngine/Colors.hpp"

ven::Engine::Engine(const uint32_t width, const uint32_t height, const std::string &title) :
m_window(width, height, title)
{
    createInstance();
    createSurface();
    ImGuiWindowManager::init(m_window.getGLFWindow(), m_instance, &m_device,
m_renderer.getSwapChainRenderPass());
    m_globalPool =
DescriptorPool::Builder(m_device).setMaxSets(SwapChain::MAX_FRAMES_IN_FLIGHT).addPoolSize(VK_DESCRIPTOR_TYPE_UNIFORM_BU
SwapChain::MAX_FRAMES_IN_FLIGHT).build();
    loadObjects();
}

void ven::Engine::createInstance()
{
    uint32_t glfwExtensionCount = 0;
    const char** glfwExtensions = nullptr;
    VkInstanceCreateInfo createInfo{};
    VkApplicationInfo appInfo{ .sType = VK_STRUCTURE_TYPE_APPLICATION_INFO, .pNext = nullptr,
.pApplicationName = "VEngine App", .applicationVersion = VK_MAKE_API_VERSION(0, 1, 0, 0), .pEngineName
= "VEngine", .engineVersion = VK_MAKE_API_VERSION(0, 1, 0, 0), .apiVersion = VK_API_VERSION_1_0 };

    createInfo.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
    createInfo.pApplicationInfo = &appInfo;
    glfwExtensions = glfwGetRequiredInstanceExtensions(&glfwExtensionCount);
    createInfo.enabledExtensionCount = glfwExtensionCount;
    createInfo.ppEnabledExtensionNames = glfwExtensions;

    if (vkCreateInstance(&createInfo, nullptr, &m_instance) != VK_SUCCESS)
    {
        throw std::runtime_error("Failed to create Vulkan instance");
    }
}

void ven::Engine::loadObjects()
{
    std::shared_ptr model = Model::createModelFromFile(m_device, "models/flat_vase.obj");

    Object flatVase = Object::createObject();
    flatVase.name = "flat vase";
    flatVase.model = model;
    flatVase.transform3D.translation = {-.5F, .5F, 0.F};
    flatVase.transform3D.scale = {3.F, 1.5F, 3.F};
    m_objects.emplace(flatVase.getId(), std::move(flatVase));

    model = Model::createModelFromFile(m_device, "models/smooth_vase.obj");
    Object smoothVase = Object::createObject();
    smoothVase.name = "smooth vase";
    smoothVase.model = model;
    smoothVase.transform3D.translation = {.5F, .5F, 0.F};
    smoothVase.transform3D.scale = {3.F, 1.5F, 3.F};
    m_objects.emplace(smoothVase.getId(), std::move(smoothVase));

    model = Model::createModelFromFile(m_device, "models/quad.obj");
    Object floor = Object::createObject();
    floor.name = "floor";
    floor.model = model;
    floor.transform3D.translation = {0.F, .5F, 0.F};
    floor.transform3D.scale = {3.F, 1.F, 3.F};
    m_objects.emplace(floor.getId(), std::move(floor));

    const std::vector<glm::vec3> lightColors{
            {Colors::RED},
            {Colors::GREEN},
            {Colors::BLUE},
            {Colors::YELLOW},
            {Colors::CYAN},
            {Colors::MAGENTA}
```

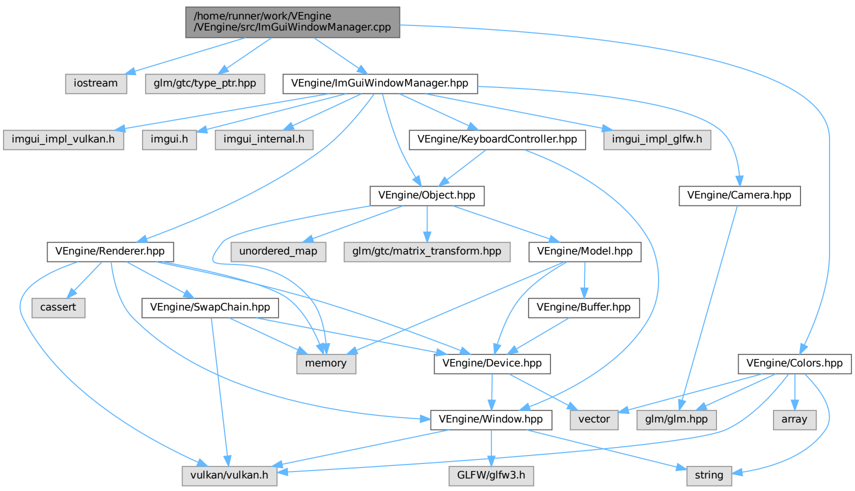
```
    };

    for (std::size_t i = 0; i < lightColors.size(); i++)
    {
        Object pointLight = Object::makePointLight();
        pointLight.name = "point light " + std::to_string(i);
        pointLight.color = lightColors[i];
        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});
        pointLight.transform3D.translation = glm::vec3(rotateLight * glm::vec4(-1.F, -1.F, -1.F, 1.F));
        m_objects.emplace(pointLight.getId(), std::move(pointLight));
    }
}

void ven::Engine::mainLoop()
{
    GlobalUbo ubo{};
    Camera camera{};
    KeyboardController cameraController{};
    std::chrono::duration<float> deltaTime{};
    VkCommandBuffer_T *commandBuffer = nullptr;
    bool showDebugWindow = true;
    float frameTime = NAN;
    int frameIndex = 0;
    Object viewerObject = Object::createObject();
    std::chrono::time_point<std::chrono::system_clock> newTime;
    std::chrono::time_point<std::chrono::system_clock> currentTime = std::chrono::high_resolution_clock::now();
    std::unique_ptr<DescriptorSetLayout> globalSetLayout = DescriptorSetLayout::Builder(m_device).addBinding(0, VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER, VK_SHADER_STAGE_ALL_GRAPHICS).build();
    std::vector<std::unique_ptr<Buffer>> uboBuffers(SwapChain::MAX_FRAMES_IN_FLIGHT);
    std::vector<VkDescriptorSet> globalDescriptorSets(SwapChain::MAX_FRAMES_IN_FLIGHT);
    RenderSystem renderSystem(m_device, m_renderer.getSwapChainRenderPass(), globalSetLayout->getDescriptorSetLayout());
    PointLightSystem pointLightSystem(m_device, m_renderer.getSwapChainRenderPass(), globalSetLayout->getDescriptorSetLayout());
    ImGuiIO &io = ImGui::GetIO();
    VkDescriptorBufferInfo bufferInfo{};

    io.ConfigFlags |= ImGuiConfigFlags_NavEnableKeyboard | ImGuiConfigFlags_NavEnableGamepad;

    for (auto& uboBuffer : uboBuffers)
    {
        uboBuffer = std::make_unique<Buffer>(m_device, sizeof(GlobalUbo), 1, VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT, VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT);
        uboBuffer->map();
    }
    for (std::size_t i = 0; i < globalDescriptorSets.size(); i++) {
        bufferInfo = uboBuffers[i]->descriptorInfo();
        DescriptorWriter(*globalSetLayout, *m_globalPool).writeBuffer(0, &bufferInfo).build(globalDescriptorSets[i]);
    }
    camera.setViewTarget(glm::vec3(-1.F, -2.F, -2.F), glm::vec3(0.F, 0.F, 2.5F));
    viewerObject.transform3D.translation.z = DEFAULT_POSITION[2];

    m_renderer.setClearValue();

    while (glfwWindowShouldClose(m_window.getGLFWindow()) == 0)
    {
        glfwPollEvents();

        newTime = std::chrono::high_resolution_clock::now();
        deltaTime = newTime - currentTime;
        currentTime = newTime;
        frameTime = deltaTime.count();
        commandBuffer = m_renderer.beginFrame();

        cameraController.moveInPlaneXZ(m_window.getGLFWindow(), frameTime, viewerObject, &showDebugWindow);
        camera.setViewYXZ(viewerObject.transform3D.translation, viewerObject.transform3D.rotation);
        camera.setPerspectiveProjection(m_renderer.getAspectRatio());

        if (commandBuffer != nullptr) {
            frameIndex = m_renderer.getFrameIndex();
            FrameInfo frameInfo{frameIndex, frameTime, commandBuffer, camera, globalDescriptorSets[static_cast<unsigned long>(frameIndex)], m_objects};
            ubo.projection = camera.getProjection();
            ubo.view = camera.getView();
            ubo.inverseView = camera.getInverseView();
            PointLightSystem::update(frameInfo, ubo);
            uboBuffers[static_cast<unsigned long>(frameIndex)]->writeToBuffer(&ubo);
            uboBuffers[static_cast<unsigned long>(frameIndex)]->flush();

            m_renderer.beginSwapChainRenderPass(frameInfo.commandBuffer);
            renderSystem.renderObjects(frameInfo);
```

```
            pointLightSystem.render(frameInfo);

            if (showDebugWindow) { ImGuiWindowManager::render(&m_renderer, m_objects, io,
    viewerObject, camera, cameraController, m_device.getPhysicalDevice()); }

            m_renderer.endSwapChainRenderPass(commandBuffer);
            m_renderer.endFrame();
            commandBuffer = nullptr;
        }
    }
    ImGuiWindowManager::cleanup();
    vkDeviceWaitIdle(m_device.device());
}
```

7.66 /home/runner/work/VEngine/VEngine/src/ImGuiWindowManager.cpp File Reference

```
#include <iostream>
#include <glm/gtc/type_ptr.hpp>
#include "VEngine/ImGuiWindowManager.hpp"
#include "VEngine/Colors.hpp"
```

Include dependency graph for ImGuiWindowManager.cpp:



7.67 ImGuiWindowManager.cpp

Go to the documentation of this file.

```
#include <iostream>

#include <glm/gtc/type_ptr.hpp>

#include "VEngine/ImGuiWindowManager.hpp"
#include "VEngine/Colors.hpp"

void ven::ImGuiWindowManager::init(GLFWwindow* window, VkInstance instance, Device* device,
    VkRenderPass renderPass)
{
    VkDescriptorPool pool = nullptr;

    ImGui::CreateContext();
    // ImGui::StyleColorsDark();
    VkDescriptorPoolSize pool_sizes[] = {
            { VK_DESCRIPTOR_TYPE_SAMPLER, 1000 },
```

```
            { VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, 1000 },
            { VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, 1000 },
            { VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, 1000 },
            { VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER, 1000 },
            { VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER, 1000 },
            { VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER, 1000 },
            { VK_DESCRIPTOR_TYPE_STORAGE_BUFFER, 1000 },
            { VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC, 1000 },
            { VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC, 1000 },
            { VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT, 1000 }
    };
    VkDescriptorPoolCreateInfo pool_info = {
            VK_STRUCTURE_TYPE_DESCRIPTOR_POOL_CREATE_INFO,
            nullptr,
            VK_DESCRIPTOR_POOL_CREATE_FREE_DESCRIPTOR_SET_BIT,
            1000,
            std::size(pool_sizes),
            pool_sizes
    };

    if (vkCreateDescriptorPool(device->device(), &pool_info, nullptr, &pool) != VK_SUCCESS) {
        throw std::runtime_error("Failed to create ImGui descriptor pool");
    }
    ImGui_ImplVulkan_InitInfo init_info = {
            .Instance = instance,
            .PhysicalDevice = device->getPhysicalDevice(),
            .Device = device->device(),
            .Queue = device->graphicsQueue(),
            .DescriptorPool = pool,
            .MinImageCount = 3,
            .ImageCount = 3,
            .MSAASamples = VK_SAMPLE_COUNT_1_BIT
    };

    ImGui_ImplGlfw_InitForVulkan(window, true);
    ImGui_ImplVulkan_Init(&init_info, renderPass);
}

void ven::ImGuiWindowManager::render(Renderer* renderer, std::unordered_map<unsigned int, Object>& objects, ImGuiIO& io, Object& cameraObj, Camera& camera, KeyboardController& cameraController, VkPhysicalDevice physicalDevice)
{
    const float framerate = io.Framerate;
    VkPhysicalDeviceProperties deviceProperties;
    vkGetPhysicalDeviceProperties(physicalDevice, &deviceProperties);

    ImGui_ImplVulkan_NewFrame();
    ImGui_ImplGlfw_NewFrame();
    ImGui::NewFrame();

    ImGui::Begin("Application Info");
    ImGui::Text("FPS: %.1f", framerate);
    ImGui::Text("Frame time: %.3fms", 1000.0f / framerate);
    ImGui::End();

    ImGui::Begin("Debug Window");

    if (ImGui::CollapsingHeader("Camera")) {
        float fov = camera.getFov();
        float near = camera.getNear();
        float far = camera.getFar();
        if (ImGui::BeginTable("CameraTable", 2)) {
            ImGui::TableNextColumn();
            ImGui::DragFloat3("Position", glm::value_ptr(cameraObj.transform3D.translation), 0.1F);
            ImGui::TableNextColumn();
            if (ImGui::Button("Reset##position")) { cameraObj.transform3D.translation = DEFAULT_POSITION; }

            ImGui::TableNextColumn();
            ImGui::DragFloat3("Rotation", glm::value_ptr(cameraObj.transform3D.rotation), 0.1F);
            ImGui::TableNextColumn();
            if (ImGui::Button("Reset##rotation")) { cameraObj.transform3D.rotation = DEFAULT_ROTATION; }

            ImGui::TableNextColumn();
            if (ImGui::SliderFloat("FOV", &fov, glm::radians(0.1F), glm::radians(180.0F))) { camera.setFov(fov); }
            ImGui::TableNextColumn();
            if (ImGui::Button("Reset##fov")) { camera.setFov(DEFAULT_FOV); }

            ImGui::TableNextColumn();
            if (ImGui::SliderFloat("Near", &near, 0.001F, 10.0F)) { camera.setNear(near); }
            ImGui::TableNextColumn();
            if (ImGui::Button("Reset##near")) { camera.setNear(DEFAULT_NEAR); }

            ImGui::TableNextColumn();
            if (ImGui::SliderFloat("Far", &far, 1.F, 1000.0F)) { camera.setFar(far); }
```

```
            ImGui::TableNextColumn();
            if (ImGui::Button("Reset##far")) { camera.setFar(DEFAULT_FAR); }

            ImGui::TableNextColumn();
            ImGui::SliderFloat("Move Speed", &cameraController.m_moveSpeed, 0.1F, 10.0F);
            ImGui::TableNextColumn();
            if (ImGui::Button("Reset##moveSpeed")) { cameraController.m_moveSpeed = DEFAULT_MOVE_SPEED; }

            ImGui::TableNextColumn();
            ImGui::SliderFloat("Look Speed", &cameraController.m_lookSpeed, 0.1F, 10.0F);
            ImGui::TableNextColumn();
            if (ImGui::Button("Reset##lookSpeed")) { cameraController.m_lookSpeed = DEFAULT_LOOK_SPEED; }

            ImGui::EndTable();
        }
    }

    if (ImGui::CollapsingHeader("Input")) {
        ImGui::IsMousePosValid() ? ImGui::Text("Mouse pos: (%g, %g)", io.MousePos.x, io.MousePos.y) : ImGui::Text("Mouse pos: <INVALID>");
        ImGui::Text("Mouse delta: (%g, %g)", io.MouseDelta.x, io.MouseDelta.y);
        ImGui::Text("Mouse down:");
        for (int i = 0; i < static_cast<int>(std::size(io.MouseDown)); i++) {
            if (ImGui::IsMouseDown(i)) {
                ImGui::SameLine();
                ImGui::Text("b%d (%.02f secs)", i, io.MouseDownDuration[i]);
            }
        }
        ImGui::Text("Mouse wheel: %.1f", io.MouseWheel);
        ImGui::Text("Keys down:");
        for (ImGuiKey key = static_cast<ImGuiKey>(0); key < ImGuiKey_NamedKey_END; key = static_cast<ImGuiKey>(key + 1)) {
            if (funcs::IsLegacyNativeDupe(key) || !ImGui::IsKeyDown(key)) { continue; }
            ImGui::SameLine();
            ImGui::Text((key < ImGuiKey_NamedKey_BEGIN) ? "\"%s\"" : "\"%s\" %d", ImGui::GetKeyName(key), key);
        }
    }

    if (ImGui::CollapsingHeader("Render Settings")) {
        ImGui::Text("Aspect Ratio: %.2f", renderer->getAspectRatio());

        if (ImGui::BeginTable("ClearColorTable", 2)) {
            ImGui::TableNextColumn();
            std::array<float, 4> clearColor = renderer->getClearColor();

            if (ImGui::ColorEdit4("Clear Color", clearColor.data())) {
                VkClearColorValue clearColorValue = {{clearColor[0], clearColor[1], clearColor[2], clearColor[3]}};
                renderer->setClearValue(clearColorValue);
            }

            ImGui::TableNextColumn();
            static int item_current = 0;

            if (ImGui::Combo("Color Presets",
                             &item_current,
                             [](void*, int idx, const char** out_text) -> bool {
                                 if (idx < 0 || idx >= static_cast<int>(std::size(Colors::CLEAR_COLORS))) { return false; }
                                 *out_text = Colors::CLEAR_COLORS.at(static_cast<unsigned long>(idx)).first;
                                 return true;
                             },
                             nullptr,
                             std::size(Colors::CLEAR_COLORS))) {
                renderer->setClearValue(Colors::CLEAR_COLORS.at(static_cast<unsigned long>(item_current)).second);
            }
            ImGui::EndTable();
        }
    }

    if (ImGui::CollapsingHeader("Device Properties")) {
        if (ImGui::BeginTable("DevicePropertiesTable", 2)) {

            ImGui::TableNextColumn(); ImGui::Text("Device Name: %s", deviceProperties.deviceName);
            ImGui::TableNextColumn(); ImGui::Text("API Version: %d.%d.%d", VK_VERSION_MAJOR(deviceProperties.apiVersion), VK_VERSION_MINOR(deviceProperties.apiVersion), VK_VERSION_PATCH(deviceProperties.apiVersion));
            ImGui::TableNextColumn(); ImGui::Text("Driver Version: %d.%d.%d", VK_VERSION_MAJOR(deviceProperties.driverVersion), VK_VERSION_MINOR(deviceProperties.driverVersion), VK_VERSION_PATCH(deviceProperties.driverVersion));
            ImGui::TableNextColumn();  ImGui::Text("Vendor ID: %d", deviceProperties.vendorID);
            ImGui::TableNextColumn(); ImGui::Text("Device ID: %d", deviceProperties.deviceID);
```

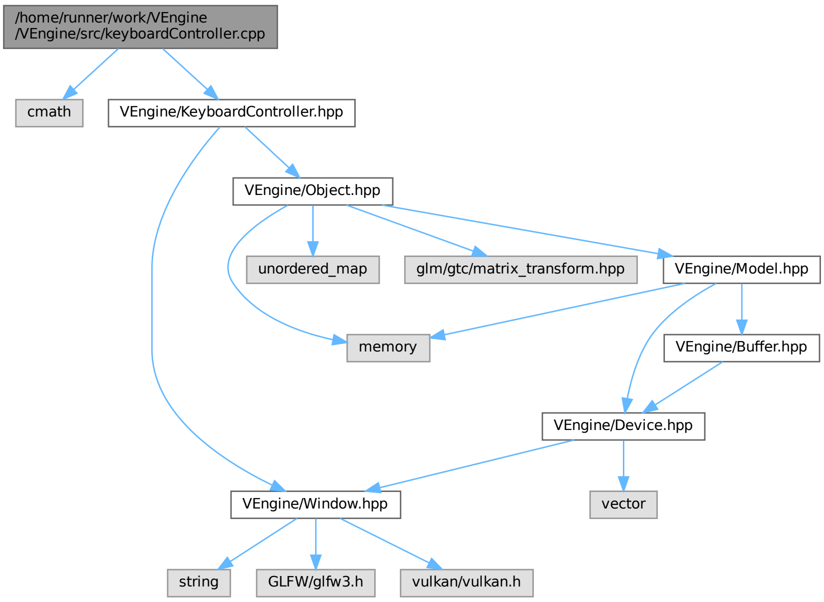
```
            ImGui::TableNextColumn(); ImGui::Text("Device Type: %d", deviceProperties.deviceType);
            ImGui::TableNextColumn(); ImGui::Text("Discrete Queue Priorities: %d", deviceProperties.limits.discreteQueuePriorities);
            ImGui::TableNextColumn(); ImGui::Text("Max Push Constants Size: %d", deviceProperties.limits.maxPushConstantsSize);
            ImGui::TableNextColumn(); ImGui::Text("Max Memory Allocation Count: %d", deviceProperties.limits.maxMemoryAllocationCount);
            ImGui::TableNextColumn(); ImGui::Text("Max Image Dimension 1D: %d", deviceProperties.limits.maxImageDimension1D);
            ImGui::TableNextColumn(); ImGui::Text("Max Image Dimension 2D: %d", deviceProperties.limits.maxImageDimension2D);
            ImGui::TableNextColumn(); ImGui::Text("Max Image Dimension 3D: %d", deviceProperties.limits.maxImageDimension3D);
            ImGui::TableNextColumn(); ImGui::Text("Max Image Dimension Cube: %d", deviceProperties.limits.maxImageDimensionCube);
            ImGui::TableNextColumn(); ImGui::Text("Max Image Array Layers: %d", deviceProperties.limits.maxImageArrayLayers);
            ImGui::TableNextColumn(); ImGui::Text("Max Texel Buffer Elements: %d", deviceProperties.limits.maxTexelBufferElements);
            ImGui::TableNextColumn(); ImGui::Text("Max Uniform Buffer Range: %d", deviceProperties.limits.maxUniformBufferRange);
            ImGui::TableNextColumn(); ImGui::Text("Max Storage Buffer Range: %d", deviceProperties.limits.maxStorageBufferRange);
            ImGui::EndTable();
        }
    }

    if (ImGui::CollapsingHeader("Objects")) {
        ImVec4 color;
        bool open = false;

        for (auto& [id, object] : objects) {
            if (object.color.r == 0.0F && object.color.g == 0.0F && object.color.b == 0.0F) {
                color = { Colors::GRAY.r, Colors::GRAY.g, Colors::GRAY.b, 1.0F };
            } else {
                color = { object.color.r, object.color.g, object.color.b, 1.0F };
            }
            ImGui::PushStyleColor(ImGuiCol_Text, color);
            open = ImGui::TreeNode(std::string(object.name + " [" + std::to_string(object.getId()) + "]").c_str());
            ImGui::PopStyleColor(1);
            if (open) {
                ImGui::Text("Address: %p", &object);
                ImGui::DragFloat3(("Position##" + object.name).c_str(), glm::value_ptr(object.transform3D.translation), 0.1F);
                ImGui::DragFloat3(("Rotation##" + object.name).c_str(), glm::value_ptr(object.transform3D.rotation), 0.1F);
                ImGui::DragFloat3(("Scale##" + object.name).c_str(), glm::value_ptr(object.transform3D.scale), 0.1F);
                if (ImGui::BeginTable("ColorTable", 2)) {
                    ImGui::TableNextColumn(); ImGui::ColorEdit3(("Color##" + object.name).c_str(), glm::value_ptr(object.color));

                    ImGui::TableNextColumn();
                    static int item_current = 0;
                    if (ImGui::Combo("Color Presets",
                                     &item_current,
                                     [](void*, int idx, const char** out_text) -> bool {
                        if (idx < 0 || idx >= static_cast<int>(std::size(Colors::COLORS))) { return false; }
                        *out_text = Colors::COLORS.at(static_cast<unsigned long>(idx)).first;
                        return true;
                        },
                        nullptr,
                        std::size(Colors::COLORS))) {
                        object.color = Colors::COLORS.at(static_cast<unsigned long>(item_current)).second;
                    }

                    ImGui::EndTable();
                }
                if (object.pointLight != nullptr) {
                    ImGui::SliderFloat(("Intensity##" + object.name).c_str(), &object.pointLight->lightIntensity, 0.0F, 10.0F);
                }
                ImGui::TreePop();
            }
        }
    }

    ImGui::End();
    ImGui::Render();
    ImGui_ImplVulkan_RenderDrawData(ImGui::GetDrawData(), renderer->getCurrentCommandBuffer());
}

void ven::ImGuiWindowManager::cleanup()
{
```

```
    ImGui_ImplVulkan_Shutdown();
    ImGui_ImplGlfw_Shutdown();
    ImGui::DestroyContext();
}
```

7.68 /home/runner/work/VEngine/VEngine/src/keyboardController.cpp File Reference

```
#include <cmath>
#include "VEngine/KeyboardController.hpp"
```

Include dependency graph for keyboardController.cpp:



7.69 keyboardController.cpp

Go to the documentation of this file.

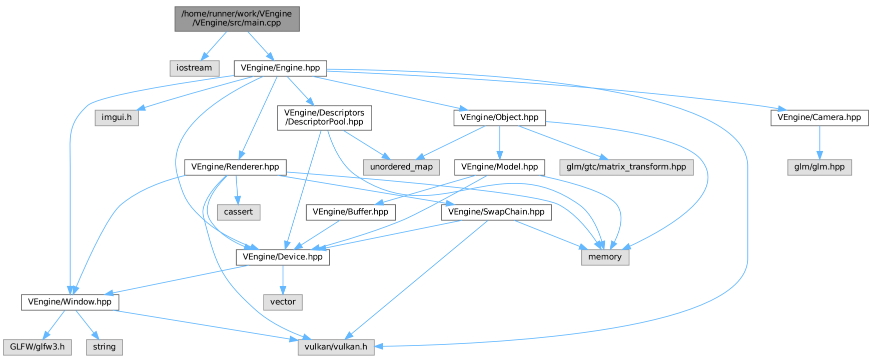
```
#include <cmath>

#include "VEngine/KeyboardController.hpp"

void ven::KeyboardController::moveInPlaneXZ(GLFWwindow* window, float dt, Object& object, bool*
showDebugWindow) const
{
    glm::vec3 rotate{0};
    if (glfwGetKey(window, m_keys.lookLeft) == GLFW_PRESS) { rotate.y -= 1.F; }
    if (glfwGetKey(window, m_keys.lookRight) == GLFW_PRESS) { rotate.y += 1.F; }
    if (glfwGetKey(window, m_keys.lookUp) == GLFW_PRESS) { rotate.x += 1.F; }
    if (glfwGetKey(window, m_keys.lookDown) == GLFW_PRESS) { rotate.x -= 1.F; }

    if (dot(rotate, rotate) > std::numeric_limits<float>::epsilon()) {
        object.transform3D.rotation += m_lookSpeed * dt * normalize(rotate);
    }

    object.transform3D.rotation.x = glm::clamp(object.transform3D.rotation.x, -1.5F, 1.5F);
    object.transform3D.rotation.y = glm::mod(object.transform3D.rotation.y, glm::two_pi<float>());

    float yaw = object.transform3D.rotation.y;
    const glm::vec3 forwardDir{std::sin(yaw), 0.F, std::cos(yaw)};
```

```
    const glm::vec3 rightDir{forwardDir.z, 0.F, -forwardDir.x};
    constexpr  glm::vec3 upDir{0.F, -1.F, 0.F};

    glm::vec3 moveDir{0.F};
    if (glfwGetKey(window, m_keys.moveForward) == GLFW_PRESS) {moveDir += forwardDir;}
    if (glfwGetKey(window, m_keys.moveBackward) == GLFW_PRESS) {moveDir -= forwardDir;}
    if (glfwGetKey(window, m_keys.moveRight) == GLFW_PRESS) {moveDir += rightDir;}
    if (glfwGetKey(window, m_keys.moveLeft) == GLFW_PRESS) {moveDir -= rightDir;}
    if (glfwGetKey(window, m_keys.moveUp) == GLFW_PRESS) {moveDir += upDir;}
    if (glfwGetKey(window, m_keys.moveDown) == GLFW_PRESS) {moveDir -= upDir;}

    if (dot(moveDir, moveDir) > std::numeric_limits<float>::epsilon()) {
        object.transform3D.translation += m_moveSpeed * dt * normalize(moveDir);
    }

    // imgui debug window
    if (glfwGetKey(window, GLFW_KEY_F1) == GLFW_PRESS) {
        *showDebugWindow = !*showDebugWindow;
    }
}
```

7.70 /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 ()

7.70.1 Function Documentation

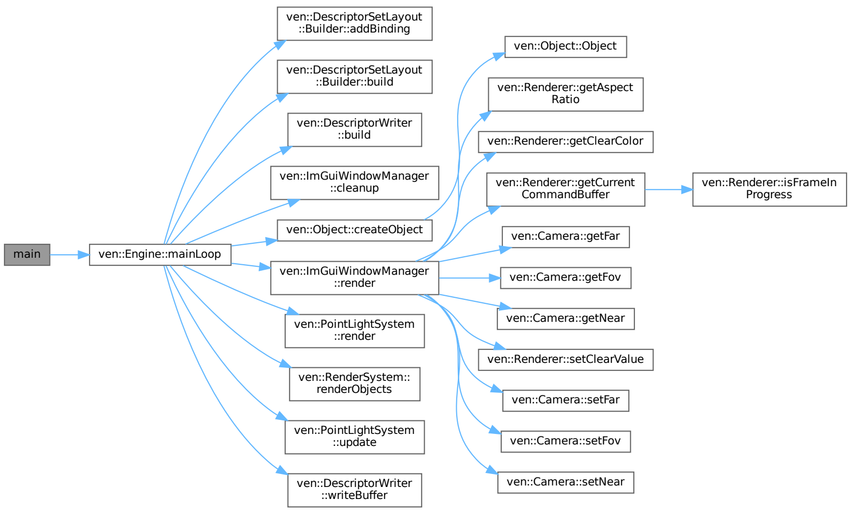
7.70.1.1 main()

```
int main ()
```

Definition at line 7 of file main.cpp.

References ven::Engine::mainLoop().

Here is the call graph for this function:



7.71 main.cpp

Go to the documentation of this file.

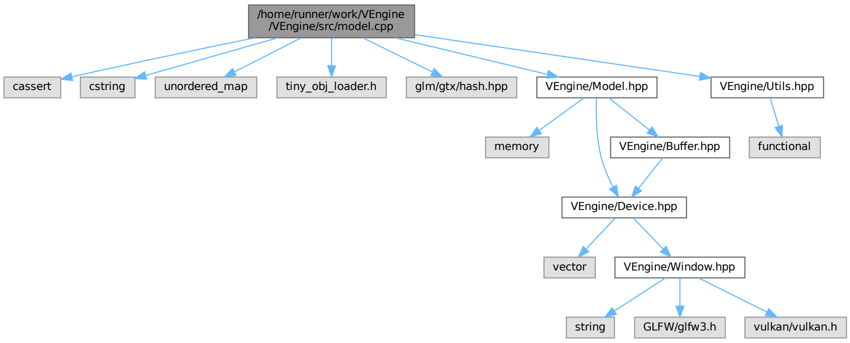
```
#include <iostream>

#include "VEngine/Engine.hpp"

using namespace ven;

int main()
{
    try {
        Engine engine{};
        engine.mainLoop();
    } catch (const std::exception &e) {
        std::cerr « "std exception: " « e.what() « '\n';
        return EXIT_FAILURE;
    } catch (...) {
        std::cerr « "Unknown error\n";
        return EXIT_SUCCESS;
    }
    return EXIT_SUCCESS;
}
```

7.72 /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

7.72.1 Macro Definition Documentation

7.72.1.1 TINYOBJLOADER_IMPLEMENTATION

```
#define TINYOBJLOADER_IMPLEMENTATION
```

Definition at line 5 of file model.cpp.

7.73 model.cpp

Go to the documentation of this file.

```cpp
#include <cassert>
#include <cstring>
#include <unordered_map>

#define TINYOBJLOADER_IMPLEMENTATION
#include <tiny_obj_loader.h>

#include <glm/gtx/hash.hpp>

#include "VEngine/Model.hpp"
#include "VEngine/Utils.hpp"

namespace std {
    template<>
    struct hash<ven::Model::Vertex> {
        size_t operator()(ven::Model::Vertex const &vertex) const {
            size_t seed = 0;
            ven::hashCombine(seed, vertex.position, vertex.color, vertex.normal, vertex.uv);
            return seed;
        }
    };
}

ven::Model::Model(Device &device, const Builder &builder) : m_device{device}, m_vertexCount(0), m_indexCount(0)
{
    createVertexBuffer(builder.vertices);
    createIndexBuffer(builder.indices);
}

ven::Model::~Model() = default;

void ven::Model::createVertexBuffer(const std::vector<Vertex> &vertices)
{
    m_vertexCount = static_cast<uint32_t>(vertices.size());
    assert(m_vertexCount >= 3 && "Vertex count must be at least 3");
    const VkDeviceSize bufferSize = sizeof(vertices[0]) * m_vertexCount;
    uint32_t vertexSize = sizeof(vertices[0]);

    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};

    stagingBuffer.map();
    stagingBuffer.writeToBuffer(vertices.data());

    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);

    m_device.copyBuffer(stagingBuffer.getBuffer(), m_vertexBuffer->getBuffer(), bufferSize);
}

void ven::Model::createIndexBuffer(const std::vector<uint32_t> &indices)
{
    m_indexCount = static_cast<uint32_t>(indices.size());
    m_hasIndexBuffer = m_indexCount > 0;

    if (!m_hasIndexBuffer) {
        return;
    }

    const VkDeviceSize bufferSize = sizeof(indices[0]) * m_indexCount;
    uint32_t indexSize = sizeof(indices[0]);

    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};

    stagingBuffer.map();
    stagingBuffer.writeToBuffer(indices.data());

    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);

    m_device.copyBuffer(stagingBuffer.getBuffer(), m_indexBuffer->getBuffer(), bufferSize);
}

void ven::Model::draw(const VkCommandBuffer commandBuffer) const
{
    if (m_hasIndexBuffer) {
        vkCmdDrawIndexed(commandBuffer, m_indexCount, 1, 0, 0, 0);
    } else {
```

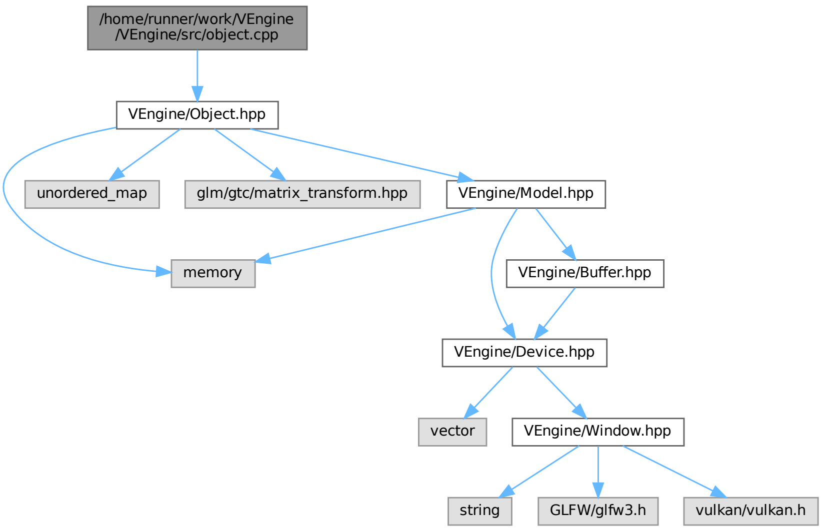
```cpp
        vkCmdDraw(commandBuffer, m_vertexCount, 1, 0, 0);
    }
}

void ven::Model::bind(const VkCommandBuffer commandBuffer) const
{
    const VkBuffer buffers[] = {m_vertexBuffer->getBuffer()};
    constexpr VkDeviceSize offsets[] = {0};
    vkCmdBindVertexBuffers(commandBuffer, 0, 1, buffers, offsets);

    if (m_hasIndexBuffer) {
        vkCmdBindIndexBuffer(commandBuffer, m_indexBuffer->getBuffer(), 0, VK_INDEX_TYPE_UINT32);
    }
}

std::unique_ptr<ven::Model> ven::Model::createModelFromFile(Device &device, const std::string
      &filename)
{
    Builder builder{};
    builder.loadModel(filename);
    return std::make_unique<Model>(device, builder);
}

std::vector<VkVertexInputBindingDescription> ven::Model::Vertex::getBindingDescriptions()
{
    std::vector<VkVertexInputBindingDescription> bindingDescriptions(1);
    bindingDescriptions[0].binding = 0;
    bindingDescriptions[0].stride = sizeof(Vertex);
    bindingDescriptions[0].inputRate = VK_VERTEX_INPUT_RATE_VERTEX;
    return bindingDescriptions;
}

std::vector<VkVertexInputAttributeDescription> ven::Model::Vertex::getAttributeDescriptions()
{
    std::vector<VkVertexInputAttributeDescription> attributeDescriptions{};

    attributeDescriptions.push_back({0, 0, VK_FORMAT_R32G32B32_SFLOAT, offsetof(Vertex, position)});
    attributeDescriptions.push_back({1, 0, VK_FORMAT_R32G32B32_SFLOAT, offsetof(Vertex, color)});
    attributeDescriptions.push_back({2, 0, VK_FORMAT_R32G32B32_SFLOAT, offsetof(Vertex, normal)});
    attributeDescriptions.push_back({3, 0, VK_FORMAT_R32G32_SFLOAT, offsetof(Vertex, uv)});

    return attributeDescriptions;
}

void ven::Model::Builder::loadModel(const std::string &filename)
{
    tinyobj::attrib_t attrib;
    std::vector<tinyobj::shape_t> shapes;
    std::vector<tinyobj::material_t> materials;
    std::string warn;
    std::string err;

    if (!LoadObj(&attrib, &shapes, &materials, &warn, &err, filename.c_str()))
    {
        throw std::runtime_error(warn + err);
    }

    vertices.clear();
    indices.clear();

    std::unordered_map<Vertex, uint32_t> uniqueVertices{};
    for (const auto &shape : shapes) {
        for (const auto &index : shape.mesh.indices) {
            Vertex vertex{};
            if (index.vertex_index >= 0) {
                vertex.position = {
                        attrib.vertices[3 * static_cast<size_t>(index.vertex_index) + 0],
                        attrib.vertices[3 * static_cast<size_t>(index.vertex_index) + 1],
                        attrib.vertices[3 * static_cast<size_t>(index.vertex_index) + 2]
                };

                vertex.color = {
                        attrib.colors[3 * static_cast<size_t>(index.vertex_index) + 0],
                        attrib.colors[3 * static_cast<size_t>(index.vertex_index) + 1],
                        attrib.colors[3 * static_cast<size_t>(index.vertex_index) + 2]
                };
            }

            if (index.normal_index >= 0) {
                vertex.normal = {
                        attrib.normals[3 * static_cast<size_t>(index.normal_index) + 0],
                        attrib.normals[3 * static_cast<size_t>(index.normal_index) + 1],
                        attrib.normals[3 * static_cast<size_t>(index.normal_index) + 2]
                };
            }

            if (index.texcoord_index >= 0) {
```

```
                    vertex.uv = {
                            attrib.texcoords[2 * static_cast<size_t>(index.texcoord_index) + 0],
                            attrib.texcoords[2 * static_cast<size_t>(index.texcoord_index) + 1]
                    };
                }

                if (!uniqueVertices.contains(vertex)) {
                    uniqueVertices[vertex] = static_cast<uint32_t>(vertices.size());
                    vertices.push_back(vertex);
                }
                indices.push_back(uniqueVertices[vertex]);
            }
        }
}
```

7.74 /home/runner/work/VEngine/VEngine/src/object.cpp File Reference

`#include "VEngine/Object.hpp"`

Include dependency graph for object.cpp:



7.75 object.cpp

Go to the documentation of this file.

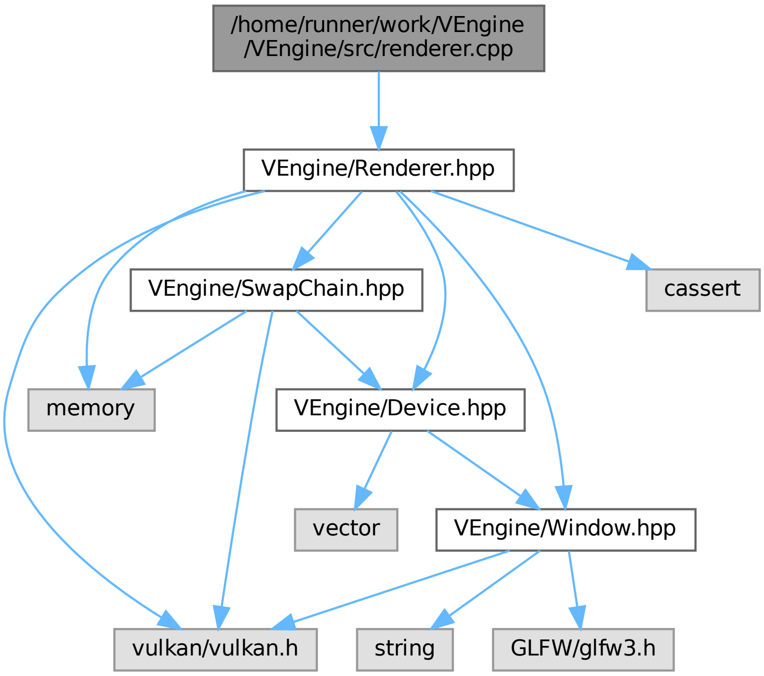
```
#include "VEngine/Object.hpp"

glm::mat4 ven::Transform3DComponent::mat4() const {
    const float c3 = glm::cos(rotation.z);
    const float s3 = glm::sin(rotation.z);
    const float c2 = glm::cos(rotation.x);
    const float s2 = glm::sin(rotation.x);
    const float c1 = glm::cos(rotation.y);
    const float s1 = glm::sin(rotation.y);
    return glm::mat4{
            {
                    scale.x * (c1 * c3 + s1 * s2 * s3),
                    scale.x * (c2 * s3),
                    scale.x * (c1 * s2 * s3 - c3 * s1),
                    0.0F,
            },
            {
                    scale.y * (c3 * s1 * s2 - c1 * s3),
                    scale.y * (c2 * c3),
```

```
                scale.y * (c1 * c3 * s2 + s1 * s3),
                0.0F,
            },
            {
                scale.z * (c2 * s1),
                scale.z * (-s2),
                scale.z * (c1 * c2),
                0.0F,
            },
            {
                translation.x,
                translation.y,
                translation.z,
                1.0F
            }
    };
}

glm::mat3 ven::Transform3DComponent::normalMatrix() const
{
    const float c3 = glm::cos(rotation.z);
    const float s3 = glm::sin(rotation.z);
    const float c2 = glm::cos(rotation.x);
    const float s2 = glm::sin(rotation.x);
    const float c1 = glm::cos(rotation.y);
    const float s1 = glm::sin(rotation.y);
    const glm::vec3 invScale = 1.0F / scale;

    return glm::mat3{
            {
                invScale.x * (c1 * c3 + s1 * s2 * s3),
                invScale.x * (c2 * s3),
                invScale.x * (c1 * s2 * s3 - c3 * s1)
            },
            {
                invScale.y * (c3 * s1 * s2 - c1 * s3),
                invScale.y * (c2 * c3),
                invScale.y * (c1 * c3 * s2 + s1 * s3)
            },
            {
                invScale.z * (c2 * s1),
                invScale.z * (-s2),
                invScale.z * (c1 * c2)
            }
    };
}

ven::Object ven::Object::makePointLight(const float intensity, const float radius, const glm::vec3 color)
{
    Object obj = Object::createObject();
    obj.color = color;
    obj.transform3D.scale.x = radius;
    obj.pointLight = std::make_unique<PointLightComponent>();
    obj.pointLight->lightIntensity = intensity;
    return obj;
}
```

7.76 /home/runner/work/VEngine/VEngine/src/renderer.cpp File Reference

```
#include "VEngine/Renderer.hpp"
```

Include dependency graph for renderer.cpp:



7.77 renderer.cpp

Go to the documentation of this file.

```
#include "VEngine/Renderer.hpp"

void ven::Renderer::createCommandBuffers()
{
    m_commandBuffers.resize(SwapChain::MAX_FRAMES_IN_FLIGHT);
    VkCommandBufferAllocateInfo allocInfo{};
    allocInfo.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO;
    allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
    allocInfo.commandPool = m_device.getCommandPool();
    allocInfo.commandBufferCount = static_cast<uint32_t>(m_commandBuffers.size());

    if (vkAllocateCommandBuffers(m_device.device(), &allocInfo, m_commandBuffers.data()) != VK_SUCCESS) {
        throw std::runtime_error("Failed to allocate command buffers");
    }
}

void ven::Renderer::freeCommandBuffers()
{
    vkFreeCommandBuffers(m_device.device(), m_device.getCommandPool(), static_cast<uint32_t>(m_commandBuffers.size()), m_commandBuffers.data());
    m_commandBuffers.clear();
}

void ven::Renderer::recreateSwapChain()
{
```

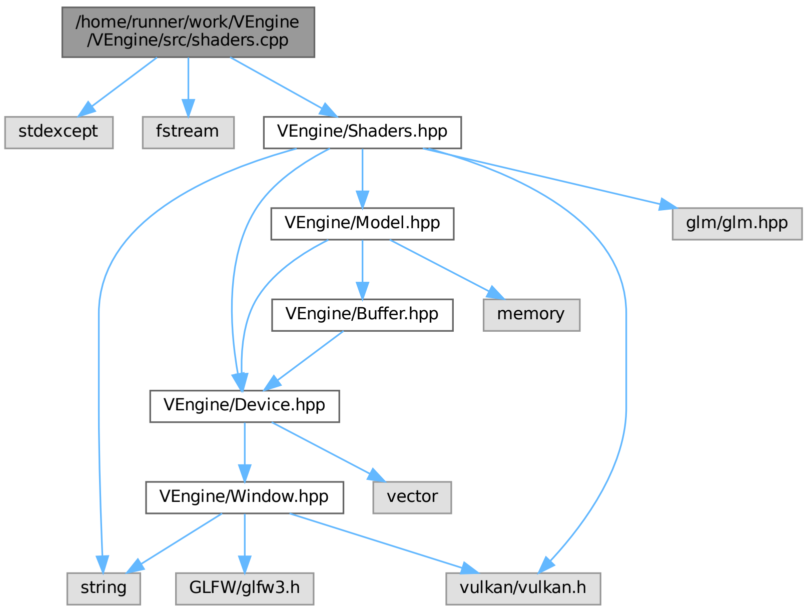
```
    VkExtent2D extent = m_window.getExtent();
    while (extent.width == 0 || extent.height == 0) {
        extent = m_window.getExtent();
        glfwWaitEvents();
    }
    vkDeviceWaitIdle(m_device.device());
    if (m_swapChain == nullptr) {
        m_swapChain = std::make_unique<SwapChain>(m_device, extent);
    } else {
        std::shared_ptr<SwapChain> oldSwapChain = std::move(m_swapChain);
        m_swapChain = std::make_unique<SwapChain>(m_device, extent, oldSwapChain);
        if (!oldSwapChain->compareSwapFormats(*m_swapChain)) {
            throw std::runtime_error("Swap chain image/depth format changed");
        }
    }
    // well be back
}

VkCommandBuffer ven::Renderer::beginFrame()
{
    assert(!m_isFrameStarted && "Can't start new frame while previous one is still in progress");

    const VkResult result = m_swapChain->acquireNextImage(&m_currentImageIndex);
    if (result == VK_ERROR_OUT_OF_DATE_KHR) {
        recreateSwapChain();
        return nullptr;
    }

    if (result != VK_SUCCESS && result != VK_SUBOPTIMAL_KHR) {
        throw std::runtime_error("Failed to acquire swap chain image");
    }

    m_isFrameStarted = true;

    VkCommandBuffer_T *commandBuffer = getCurrentCommandBuffer();
    VkCommandBufferBeginInfo beginInfo{};
    beginInfo.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO;

    if (vkBeginCommandBuffer(commandBuffer, &beginInfo) != VK_SUCCESS) {
        throw std::runtime_error("Failed to begin recording command m_buffer");
    }
    return commandBuffer;
}

void ven::Renderer::endFrame()
{
    assert(m_isFrameStarted && "Can't end frame that hasn't been started");

    VkCommandBuffer_T *commandBuffer = getCurrentCommandBuffer();
    if (vkEndCommandBuffer(commandBuffer) != VK_SUCCESS) {
        throw std::runtime_error("Failed to record command buffer");
    }
    VkResult result = m_swapChain->submitCommandBuffers(&commandBuffer, &m_currentImageIndex);
    if (result == VK_ERROR_OUT_OF_DATE_KHR || result == VK_SUBOPTIMAL_KHR || m_window.wasWindowResized()) {
        m_window.resetWindowResizedFlag();
        recreateSwapChain();
    }
    else if (result != VK_SUCCESS) {
        throw std::runtime_error("Failed to submit command buffer");
    }

    m_isFrameStarted = false;
    m_currentFrameIndex = (m_currentFrameIndex + 1) % SwapChain::MAX_FRAMES_IN_FLIGHT;
}

void ven::Renderer::beginSwapChainRenderPass(const VkCommandBuffer commandBuffer)
{
    assert(m_isFrameStarted && "Can't begin render pass when frame not in progress");
    assert(commandBuffer == getCurrentCommandBuffer() && "Can't begin render pass on command m_buffer from a different frame");

    VkRenderPassBeginInfo renderPassInfo{};
    renderPassInfo.sType = VK_STRUCTURE_TYPE_RENDER_PASS_BEGIN_INFO;
    renderPassInfo.renderPass = m_swapChain->getRenderPass();
    renderPassInfo.framebuffer = m_swapChain->getFrameBuffer(m_currentImageIndex);

    renderPassInfo.renderArea.offset = {0, 0};
    renderPassInfo.renderArea.extent = m_swapChain->getSwapChainExtent();

    renderPassInfo.clearValueCount = static_cast<uint32_t>(m_clearValues.size());
    renderPassInfo.pClearValues = m_clearValues.data();

    vkCmdBeginRenderPass(commandBuffer, &renderPassInfo, VK_SUBPASS_CONTENTS_INLINE);

    VkViewport viewport{};
    viewport.x = 0.0F;
```

```
    viewport.y = 0.0F;
    viewport.width = static_cast<float>(m_swapChain->getSwapChainExtent().width);
    viewport.height = static_cast<float>(m_swapChain->getSwapChainExtent().height);
    viewport.minDepth = 0.0F;
    viewport.maxDepth = 1.0F;
    const VkRect2D scissor{{0, 0}, m_swapChain->getSwapChainExtent()};
    vkCmdSetViewport(commandBuffer, 0, 1, &viewport);
    vkCmdSetScissor(commandBuffer, 0, 1, &scissor);
}

void ven::Renderer::endSwapChainRenderPass(const VkCommandBuffer commandBuffer)
{
    assert(m_isFrameStarted && "Can't end render pass when frame not in progress");
    assert(commandBuffer == getCurrentCommandBuffer() && "Can't end render pass on command m_buffer from a different frame");

    vkCmdEndRenderPass(commandBuffer);
}
```

7.78 /home/runner/work/VEngine/VEngine/src/shaders.cpp File Reference

```
#include <stdexcept>
#include <fstream>
#include "VEngine/Shaders.hpp"
```

Include dependency graph for shaders.cpp:



7.79 shaders.cpp

Go to the documentation of this file.

```
#include <stdexcept>
#include <fstream>

#include "VEngine/Shaders.hpp"

```

```
ven::Shaders::~Shaders()
{
    vkDestroyShaderModule(m_device.device(), m_vertShaderModule, nullptr);
    vkDestroyShaderModule(m_device.device(), m_fragShaderModule, nullptr);
    vkDestroyPipeline(m_device.device(), m_graphicsPipeline, nullptr);
}

std::vector<char> ven::Shaders::readFile(const std::string &filename)
{
    std::ifstream file(filename, std::ios::ate | std::ios::binary);

    if (!file.is_open()) {
        throw std::runtime_error("failed to open file!");
    }

    const std::streamsize fileSize = file.tellg();
    std::vector<char> buffer(static_cast<unsigned long>(fileSize));

    file.seekg(0);
    file.read(buffer.data(), fileSize);

    file.close();
    return buffer;
}

void ven::Shaders::createGraphicsPipeline(const std::string& vertFilepath, const std::string& fragFilepath, const PipelineConfigInfo& configInfo)
{
    const std::vector<char> vertCode = readFile(vertFilepath);
    const std::vector<char> fragCode = readFile(fragFilepath);

    createShaderModule(vertCode, &m_vertShaderModule);
    createShaderModule(fragCode, &m_fragShaderModule);

    VkPipelineShaderStageCreateInfo shaderStages[2];
    shaderStages[0].sType = VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_CREATE_INFO;
    shaderStages[0].stage = VK_SHADER_STAGE_VERTEX_BIT;
    shaderStages[0].module = m_vertShaderModule;
    shaderStages[0].pName = "main";
    shaderStages[0].flags = 0;
    shaderStages[0].pNext = nullptr;
    shaderStages[0].pSpecializationInfo = nullptr;

    shaderStages[1].sType = VK_STRUCTURE_TYPE_PIPELINE_SHADER_STAGE_CREATE_INFO;
    shaderStages[1].stage = VK_SHADER_STAGE_FRAGMENT_BIT;
    shaderStages[1].module = m_fragShaderModule;
    shaderStages[1].pName = "main";
    shaderStages[1].flags = 0;
    shaderStages[1].pNext = nullptr;
    shaderStages[1].pSpecializationInfo = nullptr;

    const auto& bindingDescriptions = configInfo.bindingDescriptions;
    const auto& attributeDescriptions = configInfo.attributeDescriptions;
    VkPipelineVertexInputStateCreateInfo vertexInputInfo{};
    vertexInputInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_STATE_CREATE_INFO;
    vertexInputInfo.vertexAttributeDescriptionCount = static_cast<uint32_t>(attributeDescriptions.size());
    vertexInputInfo.vertexBindingDescriptionCount = static_cast<uint32_t>(bindingDescriptions.size());
    vertexInputInfo.pVertexAttributeDescriptions = attributeDescriptions.data();
    vertexInputInfo.pVertexBindingDescriptions = bindingDescriptions.data();


    VkPipelineViewportStateCreateInfo viewportInfo{};
    viewportInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_VIEWPORT_STATE_CREATE_INFO;
    viewportInfo.viewportCount = 1;
    viewportInfo.pViewports = nullptr;
    viewportInfo.scissorCount = 1;
    viewportInfo.pScissors = nullptr;


    VkGraphicsPipelineCreateInfo pipelineInfo{};
    pipelineInfo.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
    pipelineInfo.stageCount = 2;
    pipelineInfo.pStages = shaderStages;
    pipelineInfo.pVertexInputState = &vertexInputInfo;
    pipelineInfo.pInputAssemblyState = &configInfo.inputAssemblyInfo;
    pipelineInfo.pViewportState = &viewportInfo;
    pipelineInfo.pRasterizationState = &configInfo.rasterizationInfo;
    pipelineInfo.pMultisampleState = &configInfo.multisampleInfo;

    pipelineInfo.pColorBlendState = &configInfo.colorBlendInfo;
    pipelineInfo.pDepthStencilState = &configInfo.depthStencilInfo;
    pipelineInfo.pDynamicState = &configInfo.dynamicStateInfo;

    pipelineInfo.layout = configInfo.pipelineLayout;
    pipelineInfo.renderPass = configInfo.renderPass;
    pipelineInfo.subpass = configInfo.subpass;
```

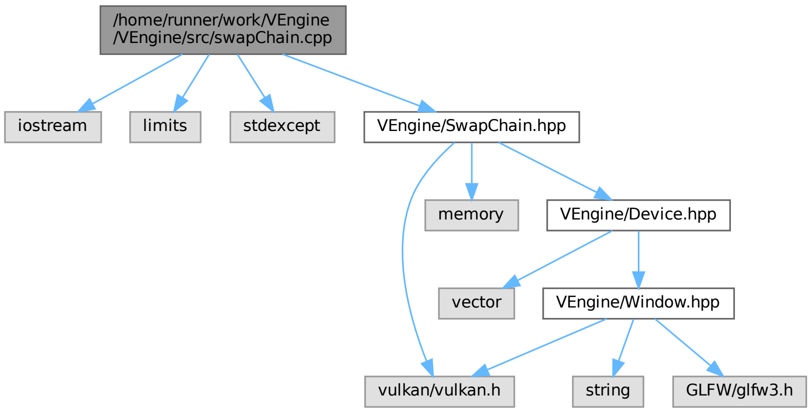
```
    pipelineInfo.basePipelineIndex = -1;
    pipelineInfo.basePipelineHandle = VK_NULL_HANDLE;

    if (vkCreateGraphicsPipelines(m_device.device(), VK_NULL_HANDLE, 1, &pipelineInfo, nullptr, &m_graphicsPipeline) != VK_SUCCESS) {
        throw std::runtime_error("failed to create graphics pipeline");
    }
}

void ven::Shaders::createShaderModule(const std::vector<char> &code, VkShaderModule *shaderModule) const
{
    VkShaderModuleCreateInfo createInfo{};
    createInfo.sType = VK_STRUCTURE_TYPE_SHADER_MODULE_CREATE_INFO;
    createInfo.codeSize = code.size();
    createInfo.pCode = reinterpret_cast<const uint32_t*>(code.data());

    if (vkCreateShaderModule(m_device.device(), &createInfo, nullptr, shaderModule) != VK_SUCCESS) {
        throw std::runtime_error("failed to create shader module");
    }
}

void ven::Shaders::defaultPipelineConfigInfo(PipelineConfigInfo& configInfo)
{
    configInfo.inputAssemblyInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_INPUT_ASSEMBLY_STATE_CREATE_INFO;
    configInfo.inputAssemblyInfo.topology = VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST;
    configInfo.inputAssemblyInfo.primitiveRestartEnable = VK_FALSE;

    configInfo.rasterizationInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_RASTERIZATION_STATE_CREATE_INFO;
    configInfo.rasterizationInfo.depthClampEnable = VK_FALSE;
    configInfo.rasterizationInfo.rasterizerDiscardEnable = VK_FALSE;
    configInfo.rasterizationInfo.polygonMode = VK_POLYGON_MODE_FILL;
    configInfo.rasterizationInfo.lineWidth = 1.0F;
    configInfo.rasterizationInfo.cullMode = VK_CULL_MODE_NONE;
    configInfo.rasterizationInfo.frontFace = VK_FRONT_FACE_COUNTER_CLOCKWISE;
    configInfo.rasterizationInfo.depthBiasEnable = VK_FALSE;
    configInfo.rasterizationInfo.depthBiasConstantFactor = 0.0F;
    configInfo.rasterizationInfo.depthBiasClamp = 0.0F;
    configInfo.rasterizationInfo.depthBiasSlopeFactor = 0.0F;

    configInfo.multisampleInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_MULTISAMPLE_STATE_CREATE_INFO;
    configInfo.multisampleInfo.sampleShadingEnable = VK_FALSE;
    configInfo.multisampleInfo.rasterizationSamples = VK_SAMPLE_COUNT_1_BIT;
    configInfo.multisampleInfo.minSampleShading = 1.0F;
    configInfo.multisampleInfo.pSampleMask = nullptr;
    configInfo.multisampleInfo.alphaToCoverageEnable = VK_FALSE;
    configInfo.multisampleInfo.alphaToOneEnable = VK_FALSE;

    configInfo.colorBlendAttachment.colorWriteMask = VK_COLOR_COMPONENT_R_BIT | VK_COLOR_COMPONENT_G_BIT | VK_COLOR_COMPONENT_B_BIT | VK_COLOR_COMPONENT_A_BIT;
    configInfo.colorBlendAttachment.blendEnable = VK_FALSE;
    configInfo.colorBlendAttachment.srcColorBlendFactor = VK_BLEND_FACTOR_ONE;
    configInfo.colorBlendAttachment.dstColorBlendFactor = VK_BLEND_FACTOR_ZERO;
    configInfo.colorBlendAttachment.colorBlendOp = VK_BLEND_OP_ADD;
    configInfo.colorBlendAttachment.srcAlphaBlendFactor = VK_BLEND_FACTOR_ONE;
    configInfo.colorBlendAttachment.dstAlphaBlendFactor = VK_BLEND_FACTOR_ZERO;
    configInfo.colorBlendAttachment.alphaBlendOp = VK_BLEND_OP_ADD;

    configInfo.colorBlendInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_COLOR_BLEND_STATE_CREATE_INFO;
    configInfo.colorBlendInfo.logicOpEnable = VK_FALSE;
    configInfo.colorBlendInfo.logicOp = VK_LOGIC_OP_COPY;
    configInfo.colorBlendInfo.attachmentCount = 1;
    configInfo.colorBlendInfo.pAttachments = &configInfo.colorBlendAttachment;
    configInfo.colorBlendInfo.blendConstants[0] = 0.0F;
    configInfo.colorBlendInfo.blendConstants[1] = 0.0F;
    configInfo.colorBlendInfo.blendConstants[2] = 0.0F;
    configInfo.colorBlendInfo.blendConstants[3] = 0.0F;

    configInfo.depthStencilInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_DEPTH_STENCIL_STATE_CREATE_INFO;
    configInfo.depthStencilInfo.depthTestEnable = VK_TRUE;
    configInfo.depthStencilInfo.depthWriteEnable = VK_TRUE;
    configInfo.depthStencilInfo.depthCompareOp = VK_COMPARE_OP_LESS;
    configInfo.depthStencilInfo.depthBoundsTestEnable = VK_FALSE;
    configInfo.depthStencilInfo.minDepthBounds = 0.0F;
    configInfo.depthStencilInfo.maxDepthBounds = 1.0F;
    configInfo.depthStencilInfo.stencilTestEnable = VK_FALSE;
    configInfo.depthStencilInfo.front = {};
    configInfo.depthStencilInfo.back = {};

    configInfo.dynamicStateEnables = {VK_DYNAMIC_STATE_VIEWPORT, VK_DYNAMIC_STATE_SCISSOR};
    configInfo.dynamicStateInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_DYNAMIC_STATE_CREATE_INFO;
    configInfo.dynamicStateInfo.pDynamicStates = configInfo.dynamicStateEnables.data();
    configInfo.dynamicStateInfo.dynamicStateCount = static_cast<uint32_t>(configInfo.dynamicStateEnables.size());
    configInfo.dynamicStateInfo.flags = 0;
    configInfo.bindingDescriptions = Model::Vertex::getBindingDescriptions();
```

```
    configInfo.attributeDescriptions = Model::Vertex::getAttributeDescriptions();
}
```

7.80 /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:



7.81 swapChain.cpp

Go to the documentation of this file.

```
#include <iostream>
#include <limits>
#include <stdexcept>

#include "VEngine/SwapChain.hpp"

ven::SwapChain::~SwapChain()
{
    for (VkImageView_T *imageView : swapChainImageViews) {
        vkDestroyImageView(device.device(), imageView, nullptr);
    }
    swapChainImageViews.clear();

    if (swapChain != nullptr) {
        vkDestroySwapchainKHR(device.device(), swapChain, nullptr);
        swapChain = nullptr;
    }

    for (size_t i = 0; i < depthImages.size(); i++) {
        vkDestroyImageView(device.device(), depthImageViews[i], nullptr);
        vkDestroyImage(device.device(), depthImages[i], nullptr);
        vkFreeMemory(device.device(), depthImageMemory[i], nullptr);
    }

    for (VkFramebuffer_T *framebuffer : swapChainFrameBuffers) {
        vkDestroyFramebuffer(device.device(), framebuffer, nullptr);
    }

    vkDestroyRenderPass(device.device(), renderPass, nullptr);
```

```
    // cleanup synchronization objects
    for (size_t i = 0; i < MAX_FRAMES_IN_FLIGHT; i++) {
        vkDestroySemaphore(device.device(), renderFinishedSemaphores[i], nullptr);
        vkDestroySemaphore(device.device(), imageAvailableSemaphores[i], nullptr);
        vkDestroyFence(device.device(), inFlightFences[i], nullptr);
    }
}

void ven::SwapChain::init()
{
    createSwapChain();
    createImageViews();
    createRenderPass();
    createDepthResources();
    createFrameBuffers();
    createSyncObjects();
}

VkResult ven::SwapChain::acquireNextImage(uint32_t *imageIndex) const
{
    vkWaitForFences(device.device(), 1, &inFlightFences[currentFrame], VK_TRUE,
std::numeric_limits<uint64_t>::max());

    return vkAcquireNextImageKHR(device.device(), swapChain, std::numeric_limits<uint64_t>::max(),
imageAvailableSemaphores[currentFrame], VK_NULL_HANDLE, imageIndex);;
}

VkResult ven::SwapChain::submitCommandBuffers(const VkCommandBuffer *buffers, const uint32_t
*imageIndex)
{
    if (imagesInFlight[*imageIndex] != VK_NULL_HANDLE) {
        vkWaitForFences(device.device(), 1, &imagesInFlight[*imageIndex], VK_TRUE, UINT64_MAX);
    }
    imagesInFlight[*imageIndex] = inFlightFences[currentFrame];

    VkSubmitInfo submitInfo = {};
    submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;

    const VkSemaphore waitSemaphores[] = {imageAvailableSemaphores[currentFrame]};
    constexpr VkPipelineStageFlags waitStages[] = {VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT};
    submitInfo.waitSemaphoreCount = 1;
    submitInfo.pWaitSemaphores = waitSemaphores;
    submitInfo.pWaitDstStageMask = waitStages;

    submitInfo.commandBufferCount = 1;
    submitInfo.pCommandBuffers = buffers;

    const VkSemaphore signalSemaphores[] = {renderFinishedSemaphores[currentFrame]};
    submitInfo.signalSemaphoreCount = 1;
    submitInfo.pSignalSemaphores = signalSemaphores;

    vkResetFences(device.device(), 1, &inFlightFences[currentFrame]);
    if (vkQueueSubmit(device.graphicsQueue(), 1, &submitInfo, inFlightFences[currentFrame]) !=
VK_SUCCESS) {
        throw std::runtime_error("failed to submit draw command m_buffer!");
    }

    VkPresentInfoKHR presentInfo = {};
    presentInfo.sType = VK_STRUCTURE_TYPE_PRESENT_INFO_KHR;

    presentInfo.waitSemaphoreCount = 1;
    presentInfo.pWaitSemaphores = signalSemaphores;

    const VkSwapchainKHR swapChains[] = {swapChain};
    presentInfo.swapchainCount = 1;
    presentInfo.pSwapchains = swapChains;

    presentInfo.pImageIndices = imageIndex;

    const VkResult result = vkQueuePresentKHR(device.presentQueue(), &presentInfo);

    currentFrame = (currentFrame + 1) % MAX_FRAMES_IN_FLIGHT;

    return result;
}

void ven::SwapChain::createSwapChain()
{
    const SwapChainSupportDetails swapChainSupport = device.getSwapChainSupport();

    const VkSurfaceFormatKHR surfaceFormat = chooseSwapSurfaceFormat(swapChainSupport.formats);
    const VkPresentModeKHR presentMode = chooseSwapPresentMode(swapChainSupport.presentModes);
    const VkExtent2D extent = chooseSwapExtent(swapChainSupport.capabilities);

    uint32_t imageCount = swapChainSupport.capabilities.minImageCount + 1;
    if (swapChainSupport.capabilities.maxImageCount > 0 && imageCount >
```

```
swapChainSupport.capabilities.maxImageCount) {
        imageCount = swapChainSupport.capabilities.maxImageCount;
    }

    VkSwapchainCreateInfoKHR createInfo = {};
    createInfo.sType = VK_STRUCTURE_TYPE_SWAPCHAIN_CREATE_INFO_KHR;
    createInfo.surface = device.surface();

    createInfo.minImageCount = imageCount;
    createInfo.imageFormat = surfaceFormat.format;
    createInfo.imageColorSpace = surfaceFormat.colorSpace;
    createInfo.imageExtent = extent;
    createInfo.imageArrayLayers = 1;
    createInfo.imageUsage = VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT;

    const QueueFamilyIndices indices = device.findPhysicalQueueFamilies();
    const uint32_t queueFamilyIndices[] = {indices.graphicsFamily, indices.presentFamily};

    if (indices.graphicsFamily != indices.presentFamily) {
        createInfo.imageSharingMode = VK_SHARING_MODE_CONCURRENT;
        createInfo.queueFamilyIndexCount = 2;
        createInfo.pQueueFamilyIndices = queueFamilyIndices;
    } else {
        createInfo.imageSharingMode = VK_SHARING_MODE_EXCLUSIVE;
        createInfo.queueFamilyIndexCount = 0;      // Optional
        createInfo.pQueueFamilyIndices = nullptr;  // Optional
    }

    createInfo.preTransform = swapChainSupport.capabilities.currentTransform;
    createInfo.compositeAlpha = VK_COMPOSITE_ALPHA_OPAQUE_BIT_KHR;

    createInfo.presentMode = presentMode;
    createInfo.clipped = VK_TRUE;

    createInfo.oldSwapchain = oldSwapChain == nullptr ? VK_NULL_HANDLE : oldSwapChain->swapChain;

    if (vkCreateSwapchainKHR(device.device(), &createInfo, nullptr, &swapChain) != VK_SUCCESS) {
        throw std::runtime_error("failed to create swap chain!");
    }

    vkGetSwapchainImagesKHR(device.device(), swapChain, &imageCount, nullptr);
    swapChainImages.resize(imageCount);
    vkGetSwapchainImagesKHR(device.device(), swapChain, &imageCount, swapChainImages.data());

    swapChainImageFormat = surfaceFormat.format;
    m_swapChainExtent = extent;
}

void ven::SwapChain::createImageViews()
{
    swapChainImageViews.resize(swapChainImages.size());
    for (size_t i = 0; i < swapChainImages.size(); i++) {
        VkImageViewCreateInfo viewInfo{};
        viewInfo.sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO;
        viewInfo.image = swapChainImages[i];
        viewInfo.viewType = VK_IMAGE_VIEW_TYPE_2D;
        viewInfo.format = swapChainImageFormat;
        viewInfo.subresourceRange.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
        viewInfo.subresourceRange.baseMipLevel = 0;
        viewInfo.subresourceRange.levelCount = 1;
        viewInfo.subresourceRange.baseArrayLayer = 0;
        viewInfo.subresourceRange.layerCount = 1;

        if (vkCreateImageView(device.device(), &viewInfo, nullptr, &swapChainImageViews[i]) !=
VK_SUCCESS) {
            throw std::runtime_error("failed to create texture image view!");
        }
    }
}

void ven::SwapChain::createRenderPass()
{
    VkAttachmentDescription depthAttachment{};
    depthAttachment.format = findDepthFormat();
    depthAttachment.samples = VK_SAMPLE_COUNT_1_BIT;
    depthAttachment.loadOp = VK_ATTACHMENT_LOAD_OP_CLEAR;
    depthAttachment.storeOp = VK_ATTACHMENT_STORE_OP_DONT_CARE;
    depthAttachment.stencilLoadOp = VK_ATTACHMENT_LOAD_OP_DONT_CARE;
    depthAttachment.stencilStoreOp = VK_ATTACHMENT_STORE_OP_DONT_CARE;
    depthAttachment.initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
    depthAttachment.finalLayout = VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL;

    VkAttachmentReference depthAttachmentRef{};
    depthAttachmentRef.attachment = 1;
    depthAttachmentRef.layout = VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL;

    VkAttachmentDescription colorAttachment = {};
```

```
    colorAttachment.format = getSwapChainImageFormat();
    colorAttachment.samples = VK_SAMPLE_COUNT_1_BIT;
    colorAttachment.loadOp = VK_ATTACHMENT_LOAD_OP_CLEAR;
    colorAttachment.storeOp = VK_ATTACHMENT_STORE_OP_STORE;
    colorAttachment.stencilStoreOp = VK_ATTACHMENT_STORE_OP_DONT_CARE;
    colorAttachment.stencilLoadOp = VK_ATTACHMENT_LOAD_OP_DONT_CARE;
    colorAttachment.initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
    colorAttachment.finalLayout = VK_IMAGE_LAYOUT_PRESENT_SRC_KHR;

    VkAttachmentReference colorAttachmentRef = {};
    colorAttachmentRef.attachment = 0;
    colorAttachmentRef.layout = VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL;

    VkSubpassDescription subpass = {};
    subpass.pipelineBindPoint = VK_PIPELINE_BIND_POINT_GRAPHICS;
    subpass.colorAttachmentCount = 1;
    subpass.pColorAttachments = &colorAttachmentRef;
    subpass.pDepthStencilAttachment = &depthAttachmentRef;

    VkSubpassDependency dependency = {};
    dependency.srcSubpass = VK_SUBPASS_EXTERNAL;
    dependency.srcAccessMask = 0;
    dependency.srcStageMask = VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT | VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT;
    dependency.dstSubpass = 0;
    dependency.dstStageMask = VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT | VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT;
    dependency.dstAccessMask = VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT | VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT;

    const std::array<VkAttachmentDescription, 2> attachments = {colorAttachment, depthAttachment};
    VkRenderPassCreateInfo renderPassInfo = {};
    renderPassInfo.sType = VK_STRUCTURE_TYPE_RENDER_PASS_CREATE_INFO;
    renderPassInfo.attachmentCount = static_cast<uint32_t>(attachments.size());
    renderPassInfo.pAttachments = attachments.data();
    renderPassInfo.subpassCount = 1;
    renderPassInfo.pSubpasses = &subpass;
    renderPassInfo.dependencyCount = 1;
    renderPassInfo.pDependencies = &dependency;

    if (vkCreateRenderPass(device.device(), &renderPassInfo, nullptr, &renderPass) != VK_SUCCESS) {
        throw std::runtime_error("failed to create render pass!");
    }
}

void ven::SwapChain::createFrameBuffers()
{
    swapChainFrameBuffers.resize(imageCount());
    for (size_t i = 0; i < imageCount(); i++) {
        std::array<VkImageView, 2> attachments = {swapChainImageViews[i], depthImageViews[i]};

        const VkExtent2D swapChainExtent = getSwapChainExtent();
        VkFramebufferCreateInfo framebufferInfo = {};
        framebufferInfo.sType = VK_STRUCTURE_TYPE_FRAMEBUFFER_CREATE_INFO;
        framebufferInfo.renderPass = renderPass;
        framebufferInfo.attachmentCount = static_cast<uint32_t>(attachments.size());
        framebufferInfo.pAttachments = attachments.data();
        framebufferInfo.width = swapChainExtent.width;
        framebufferInfo.height = swapChainExtent.height;
        framebufferInfo.layers = 1;

        if (vkCreateFramebuffer(device.device(), &framebufferInfo, nullptr, &swapChainFrameBuffers[i]) != VK_SUCCESS) {
            throw std::runtime_error("failed to create framebuffer!");
        }
    }
}

void ven::SwapChain::createDepthResources()
{
    const VkFormat depthFormat = findDepthFormat();
    const VkExtent2D swapChainExtent = getSwapChainExtent();

    swapChainDepthFormat = depthFormat;
    depthImages.resize(imageCount());
    depthImageMemory.resize(imageCount());
    depthImageViews.resize(imageCount());

    for (size_t i = 0; i < depthImages.size(); i++) {
        VkImageCreateInfo imageInfo{};
        imageInfo.sType = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
        imageInfo.imageType = VK_IMAGE_TYPE_2D;
        imageInfo.extent.width = swapChainExtent.width;
        imageInfo.extent.height = swapChainExtent.height;
        imageInfo.extent.depth = 1;
        imageInfo.mipLevels = 1;
        imageInfo.arrayLayers = 1;
```

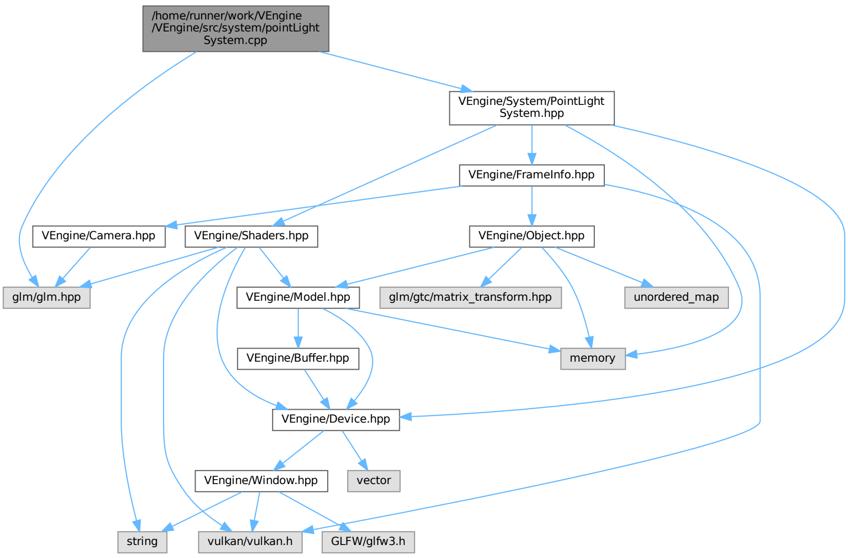
```
        imageInfo.format = depthFormat;
        imageInfo.tiling = VK_IMAGE_TILING_OPTIMAL;
        imageInfo.initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
        imageInfo.usage = VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT;
        imageInfo.samples = VK_SAMPLE_COUNT_1_BIT;
        imageInfo.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
        imageInfo.flags = 0;

        device.createImageWithInfo(imageInfo, VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT, depthImages[i],
depthImageMemory[i]);

        VkImageViewCreateInfo viewInfo{};
        viewInfo.sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO;
        viewInfo.image = depthImages[i];
        viewInfo.viewType = VK_IMAGE_VIEW_TYPE_2D;
        viewInfo.format = depthFormat;
        viewInfo.subresourceRange.aspectMask = VK_IMAGE_ASPECT_DEPTH_BIT;
        viewInfo.subresourceRange.baseMipLevel = 0;
        viewInfo.subresourceRange.levelCount = 1;
        viewInfo.subresourceRange.baseArrayLayer = 0;
        viewInfo.subresourceRange.layerCount = 1;

        if (vkCreateImageView(device.device(), &viewInfo, nullptr, &depthImageViews[i]) != VK_SUCCESS)
{
            throw std::runtime_error("failed to create texture image view!");
        }
    }
}

void ven::SwapChain::createSyncObjects()
{
    imageAvailableSemaphores.resize(MAX_FRAMES_IN_FLIGHT);
    renderFinishedSemaphores.resize(MAX_FRAMES_IN_FLIGHT);
    inFlightFences.resize(MAX_FRAMES_IN_FLIGHT);
    imagesInFlight.resize(imageCount(), VK_NULL_HANDLE);

    VkSemaphoreCreateInfo semaphoreInfo = {};
    semaphoreInfo.sType = VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO;

    VkFenceCreateInfo fenceInfo = {};
    fenceInfo.sType = VK_STRUCTURE_TYPE_FENCE_CREATE_INFO;
    fenceInfo.flags = VK_FENCE_CREATE_SIGNALED_BIT;

    for (size_t i = 0; i < MAX_FRAMES_IN_FLIGHT; i++) {
        if (vkCreateSemaphore(device.device(), &semaphoreInfo, nullptr, &imageAvailableSemaphores[i])
!= VK_SUCCESS ||
            vkCreateSemaphore(device.device(), &semaphoreInfo, nullptr, &renderFinishedSemaphores[i])
!= VK_SUCCESS ||
            vkCreateFence(device.device(), &fenceInfo, nullptr, &inFlightFences[i]) != VK_SUCCESS) {
                throw std::runtime_error("failed to create synchronization objects for a frame!");
        }
    }
}

VkSurfaceFormatKHR ven::SwapChain::chooseSwapSurfaceFormat(const std::vector<VkSurfaceFormatKHR>
&availableFormats)
{
    for (const auto &availableFormat : availableFormats) {
        if (availableFormat.format == VK_FORMAT_B8G8R8A8_UNORM && availableFormat.colorSpace ==
VK_COLOR_SPACE_SRGB_NONLINEAR_KHR) {
            return availableFormat;
        }
    }

    return availableFormats[0];
}

VkPresentModeKHR ven::SwapChain::chooseSwapPresentMode(const std::vector<VkPresentModeKHR>
&availablePresentModes)
{
    for (const auto &availablePresentMode : availablePresentModes) {
        if (availablePresentMode == VK_PRESENT_MODE_MAILBOX_KHR) {
            std::cout « "Present mode: Mailbox\n";
            return availablePresentMode;
        }
    }

   for (const auto &availablePresentMode : availablePresentModes) {
     if (availablePresentMode == VK_PRESENT_MODE_IMMEDIATE_KHR) {
       std::cout « "Present mode: Immediate" « '\n';
       return availablePresentMode;
     }
   }

  std::cout « "Present mode: V-Sync\n";
  return VK_PRESENT_MODE_FIFO_KHR;
}
```

```
VkExtent2D ven::SwapChain::chooseSwapExtent(const VkSurfaceCapabilitiesKHR &capabilities) const
{
    if (capabilities.currentExtent.width != std::numeric_limits<uint32_t>::max()) {
        return capabilities.currentExtent;
    }
    VkExtent2D actualExtent = windowExtent;
    actualExtent.width = std::max(capabilities.minImageExtent.width,
std::min(capabilities.maxImageExtent.width, actualExtent.width));
    actualExtent.height = std::max(capabilities.minImageExtent.height,
std::min(capabilities.maxImageExtent.height, actualExtent.height));

    return actualExtent;
}

VkFormat ven::SwapChain::findDepthFormat() const
{
    return device.findSupportedFormat(
        {VK_FORMAT_D32_SFLOAT, VK_FORMAT_D32_SFLOAT_S8_UINT, VK_FORMAT_D24_UNORM_S8_UINT},
        VK_IMAGE_TILING_OPTIMAL,
        VK_FORMAT_FEATURE_DEPTH_STENCIL_ATTACHMENT_BIT);
}
```

7.82 /home/runner/work/VEngine/VEngine/src/system/pointLightSystem.cpp File Reference

```
#include <glm/glm.hpp>
#include "VEngine/System/PointLightSystem.hpp"
```

Include dependency graph for pointLightSystem.cpp:



Classes

- struct PointLightPushConstants

7.83 pointLightSystem.cpp

Go to the documentation of this file.

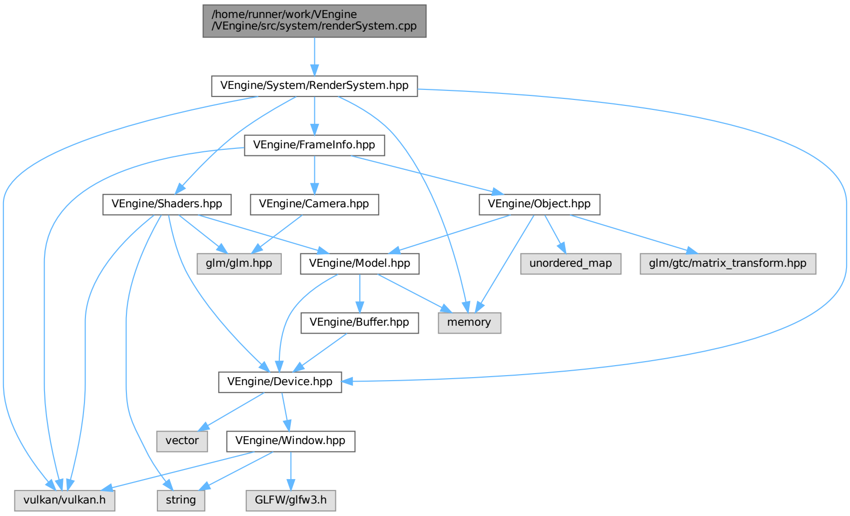
```
#include <glm/glm.hpp>

#include "VEngine/System/PointLightSystem.hpp"

struct PointLightPushConstants {
    glm::vec4 position{};
    glm::vec4 color{};
    float radius;
};

ven::PointLightSystem::PointLightSystem(Device& device, const VkRenderPass renderPass,const VkDescriptorSetLayout globalSetLayout) : m_device{device}
{
    createPipelineLayout(globalSetLayout);
    createPipeline(renderPass);
}

void ven::PointLightSystem::createPipelineLayout(const VkDescriptorSetLayout globalSetLayout)
{
    VkPushConstantRange pushConstantRange{};
    pushConstantRange.stageFlags = VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT;
    pushConstantRange.offset = 0;
    pushConstantRange.size = sizeof(PointLightPushConstants);

    const std::vector<VkDescriptorSetLayout> descriptorSetLayouts{globalSetLayout};

    VkPipelineLayoutCreateInfo pipelineLayoutInfo{};
    pipelineLayoutInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_LAYOUT_CREATE_INFO;
    pipelineLayoutInfo.setLayoutCount = static_cast<uint32_t>(descriptorSetLayouts.size());
    pipelineLayoutInfo.pSetLayouts = descriptorSetLayouts.data();
    pipelineLayoutInfo.pushConstantRangeCount = 1;
    pipelineLayoutInfo.pPushConstantRanges = &pushConstantRange;
    if (vkCreatePipelineLayout(m_device.device(), &pipelineLayoutInfo, nullptr, &m_pipelineLayout) != VK_SUCCESS)
    {
        throw std::runtime_error("Failed to create pipeline layout");
    }
}

void ven::PointLightSystem::createPipeline(const VkRenderPass renderPass)
{
    PipelineConfigInfo pipelineConfig{};
    Shaders::defaultPipelineConfigInfo(pipelineConfig);
    pipelineConfig.attributeDescriptions.clear();
    pipelineConfig.bindingDescriptions.clear();
    pipelineConfig.renderPass = renderPass;
    pipelineConfig.pipelineLayout = m_pipelineLayout;
    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);
}

void ven::PointLightSystem::render(const FrameInfo &frameInfo) const
{
    m_shaders->bind(frameInfo.commandBuffer);

    vkCmdBindDescriptorSets(frameInfo.commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, m_pipelineLayout, 0, 1, &frameInfo.globalDescriptorSet, 0, nullptr);

    for (auto &kv : frameInfo.objects)
    {
        Object &object = kv.second;
        if (object.pointLight == nullptr) continue;
        PointLightPushConstants push{};
        push.position = glm::vec4(object.transform3D.translation, 1.F);
        push.color = glm::vec4(object.color, object.pointLight->lightIntensity);
        push.radius = object.transform3D.scale.x;
        vkCmdPushConstants(frameInfo.commandBuffer, m_pipelineLayout, VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT, 0, sizeof(PointLightPushConstants), &push);
        vkCmdDraw(frameInfo.commandBuffer, 6, 1, 0, 0);
    }

}

void ven::PointLightSystem::update(const FrameInfo &frameInfo, GlobalUbo &ubo)
{
    const auto rotateLight = rotate(glm::mat4(1.F), frameInfo.frameTime, {0.F, -1.F, 0.F});
    unsigned long lightIndex = 0;
    for (auto &kv : frameInfo.objects)
    {
        Object &object = kv.second;
        if (object.pointLight == nullptr) continue;
        assert(lightIndex < MAX_LIGHTS && "Too many lights");
        object.transform3D.translation = glm::vec3(rotateLight * glm::vec4(object.transform3D.translation, 1.F));
        ubo.pointLights[lightIndex].position = glm::vec4(object.transform3D.translation, 1.F);
        ubo.pointLights[lightIndex].color = glm::vec4(object.color, object.pointLight->lightIntensity);
```

```
        lightIndex++;
    }
    ubo.numLights = static_cast<int>(lightIndex);
}
```

7.84 /home/runner/work/VEngine/VEngine/src/system/renderSystem.cpp File Reference

`#include "VEngine/System/RenderSystem.hpp"`

Include dependency graph for renderSystem.cpp:



7.85 renderSystem.cpp

Go to the documentation of this file.

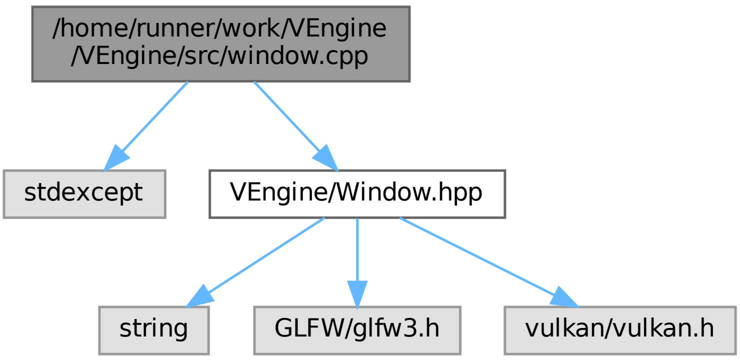
```
#include "VEngine/System/RenderSystem.hpp"

ven::RenderSystem::RenderSystem(Device& device, const VkRenderPass renderPass,const
    VkDescriptorSetLayout globalSetLayout) : m_device{device}
{
    createPipelineLayout(globalSetLayout);
    createPipeline(renderPass);
}

void ven::RenderSystem::createPipelineLayout(const VkDescriptorSetLayout globalSetLayout)
{
    VkPushConstantRange pushConstantRange{};
    pushConstantRange.stageFlags = VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT;
    pushConstantRange.offset = 0;
    pushConstantRange.size = sizeof(SimplePushConstantData);

    const std::vector<VkDescriptorSetLayout> descriptorSetLayouts{globalSetLayout};

    VkPipelineLayoutCreateInfo pipelineLayoutInfo{};
    pipelineLayoutInfo.sType = VK_STRUCTURE_TYPE_PIPELINE_LAYOUT_CREATE_INFO;
    pipelineLayoutInfo.setLayoutCount = static_cast<uint32_t>(descriptorSetLayouts.size());
    pipelineLayoutInfo.pSetLayouts = descriptorSetLayouts.data();
    pipelineLayoutInfo.pushConstantRangeCount = 1;
    pipelineLayoutInfo.pPushConstantRanges = &pushConstantRange;
    if (vkCreatePipelineLayout(m_device.device(), &pipelineLayoutInfo, nullptr, &m_pipelineLayout) !=
    VK_SUCCESS)
    {
        throw std::runtime_error("Failed to create pipeline layout");
```

```
    }
}

void ven::RenderSystem::createPipeline(const VkRenderPass renderPass)
{
    PipelineConfigInfo pipelineConfig{};
    Shaders::defaultPipelineConfigInfo(pipelineConfig);
    pipelineConfig.renderPass = renderPass;
    pipelineConfig.pipelineLayout = m_pipelineLayout;
    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);
}

void ven::RenderSystem::renderObjects(const FrameInfo &frameInfo) const
{
    m_shaders->bind(frameInfo.commandBuffer);

    vkCmdBindDescriptorSets(frameInfo.commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS, m_pipelineLayout, 0, 1, &frameInfo.globalDescriptorSet, 0, nullptr);

    for (auto &kv : frameInfo.objects)
    {
        Object &object = kv.second;
        if (object.model == nullptr) continue;
        SimplePushConstantData push{};
        push.modelMatrix = object.transform3D.mat4();
        push.normalMatrix = object.transform3D.normalMatrix();
        vkCmdPushConstants(frameInfo.commandBuffer, m_pipelineLayout, VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT, 0, sizeof(SimplePushConstantData), &push);
        object.model->bind(frameInfo.commandBuffer);
        object.model->draw(frameInfo.commandBuffer);
    }
}
```

7.86 /home/runner/work/VEngine/VEngine/src/window.cpp File Reference

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

Include dependency graph for window.cpp:



7.87 window.cpp

Go to the documentation of this file.

```cpp
#include <stdexcept>

#include "VEngine/Window.hpp"

GLFWwindow* ven::Window::createWindow(const uint32_t width, const uint32_t height, const std::string &title)
{
    if (glfwInit() == GLFW_FALSE) {
        throw std::runtime_error("Failed to initialize GLFW");
    }

    glfwWindowHint(GLFW_CLIENT_API, GLFW_NO_API);
    glfwWindowHint(GLFW_RESIZABLE, GLFW_TRUE);

    GLFWwindow *window = glfwCreateWindow(static_cast<int>(width), static_cast<int>(height), title.c_str(), nullptr, nullptr);
    if (window == nullptr) {
        glfwTerminate();
        throw std::runtime_error("Failed to create window");
    }
    glfwSetWindowUserPointer(window, this);
    glfwSetFramebufferSizeCallback(window, framebufferResizeCallback);
    return window;
}

void ven::Window::createWindowSurface(const VkInstance instance, VkSurfaceKHR *surface) const
{
    if (glfwCreateWindowSurface(instance, m_window, nullptr, surface) != VK_SUCCESS) {
        throw std::runtime_error("Failed to create window surface");
    }
}

void ven::Window::framebufferResizeCallback(GLFWwindow *window, const int width, const int height)
{
    auto *app = static_cast<Window *>(glfwGetWindowUserPointer(window));
    app->m_framebufferResized = true;
    app->m_width = static_cast<uint32_t>(width);
    app->m_height = static_cast<uint32_t>(height);
}
```

Index