# vengine

0.1.0

Generated by Doxygen 1.11.0

# Chapter 1

# vengine

## 1.1 VEngine - Vulkan Graphics Engine

**WORK IN PROGRESS!**

Welcome to **VEngine**, a Vulkan-based graphics engine.

This project is designed to provide a high-performance and flexible foundation for building 3D applications and games, taking full advantage of the Vulkan API.

### 1.1.1 Features

- **Vulkan Rendering Pipeline**: Leveraging Vulkan for high-performance graphics rendering
- **Basic Camera System**: Control camera movement in the 3D space
- **Input System**: Keyboard-based controls for movement and looking around
- **Model Loading**: Import 3D models using `assimp`
- **Real-time debugging**: Toggle debug windows using key bindings
- **Doxygen Documentation**: Automatically generated documentation hosted on `GitHub Pages`

#### 1.1.1.1 Planned Features:

- **Cross-platform support** (Linux, macOS, Windows)
- Improve shadow
- Physics Integration
- Support for more input devices (e.g., mouse, game controller)
- Audio Integration

### 1.1.2 Prerequisites

Make sure you have the following dependencies installed on your system:

- `CMake 3.27`
- `C++20`
- `Vulkan`
- `GLM`

### 1.1.3 Usage

#### 1.1.3.1 Build

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

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

#### 1.1.3.2 Run

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

### 1.1.4 Key Bindings

The following keyboard controls are currently available for interacting with the engine:

| 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 |

### 1.1.5 Documentation

The documentation is generated using `Doxygen`. You can access the latest version on the `GitHub Pages`.

### 1.1.6 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 |

### 1.1.7 License

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

### 1.1.8 Acknowledgements

Special thanks to `Brendan Galea` for inspiration and resources related to Vulkan development.

# Chapter 2

# Namespace Index

## 2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

# Chapter 3

# Hierarchical Index

## 3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

# Chapter 4

# Class Index

## 4.1 Class List

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

# Chapter 5

# File Index

## 5.1 File List

Here is a list of all files with brief descriptions:

# Chapter 6

# Namespace Documentation

## 6.1 ven Namespace Reference

### Classes

- class ARenderSystemBase

  *Abstract class for render system base.*
- 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 for device.*
- class Engine

  *Class for engine.*
- class EventManager

  *Class for event manager.*
- struct FrameInfo
- struct GlobalUbo
- class Gui

  *Class for Gui.*
- struct KeyAction
- struct KeyMappings
- class Light

  *Class for light.*
- struct LightPushConstantData
- class Model

  *Class for model.*

- class Object

    *Class for object.*
- struct ObjectPushConstantData
- class ObjectRenderSystem

    *Class for object render system.*
- struct PipelineConfigInfo
- struct PointLightData
- class PointLightRenderSystem

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

    *Class for renderer.*
- class Shaders

    *Class for shaders.*
- class SwapChain

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

    *Class for 3D transformation.*
- class Window

    *Class for window.*

## Enumerations

- enum ENGINE_STATE : uint8_t { EDITOR = 0 , GAME = 1 , PAUSED = 2 , EXIT = 3 }
- enum GUI_STATE : uint8_t { VISIBLE = 0 , HIDDEN = 1 }

## 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 float DEFAULT_MOVE_SPEED = 3.F
- static constexpr float DEFAULT_LOOK_SPEED = 1.5F
- static constexpr float COLOR_MAX = 255.0F
- static constexpr uint32_t DEFAULT_MAX_SETS = 1000
- static constexpr uint16_t MAX_LIGHTS = 10
- static constexpr float DEFAULT_AMBIENT_LIGHT_INTENSITY = .2F
- static constexpr glm::vec4 DEFAULT_AMBIENT_LIGHT_COLOR = {glm::vec3(1.F), DEFAULT_AMBIENT_LIGHT_INTENSITY}
- static constexpr float DEFAULT_LIGHT_INTENSITY = .2F
- static constexpr float DEFAULT_LIGHT_RADIUS = 0.1F
- static constexpr glm::vec4 DEFAULT_LIGHT_COLOR = {glm::vec3(1.F), DEFAULT_LIGHT_INTENSITY}
- 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 = "build/shaders/"
- static constexpr int MAX_FRAMES_IN_FLIGHT = 2
- static constexpr uint32_t DEFAULT_WIDTH = 1920
- static constexpr uint32_t DEFAULT_HEIGHT = 1080
- static constexpr std::string_view DEFAULT_TITLE = "VEngine"

## 6.1.1 Enumeration Type Documentation

### 6.1.1.1 ENGINE_STATE

```
enum ven::ENGINE_STATE : uint8_t
```

**Enumerator**

| | |
|---|---|
| EDITOR | |
| GAME | |
| PAUSED | |
| EXIT | |

Definition at line 21 of file Engine.hpp.

### 6.1.1.2 GUI_STATE

```
enum ven::GUI_STATE : uint8_t
```

**Enumerator**

| | |
|---|---|
| VISIBLE | |
| HIDDEN | |

Definition at line 18 of file Gui.hpp.

## 6.1.2 Function Documentation

### 6.1.2.1 hashCombine()

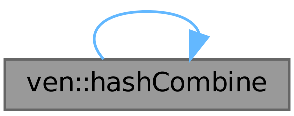
```
template<typename T , typename... Rest>
void ven::hashCombine (
            std::size_t & seed,
            const T & v,
            const Rest &... rest)
```

Definition at line 14 of file Utils.hpp.

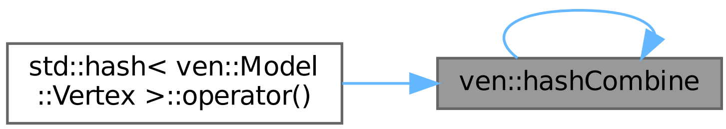
References hashCombine().

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

Here is the call graph for this function:



Here is the caller graph for this function:




### 6.1.3 Variable Documentation

#### 6.1.3.1 COLOR_MAX

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

Definition at line 15 of file Colors.hpp.

#### 6.1.3.2 DEFAULT_AMBIENT_LIGHT_COLOR

```
glm::vec4 ven::DEFAULT_AMBIENT_LIGHT_COLOR = {glm::vec3(1.F), DEFAULT_AMBIENT_LIGHT_INTENSITY}
[static], [constexpr]
```

Definition at line 20 of file FrameInfo.hpp.

#### 6.1.3.3 DEFAULT_AMBIENT_LIGHT_INTENSITY

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

Definition at line 19 of file FrameInfo.hpp.

Referenced by ven::Gui::rendererSection().

#### 6.1.3.4 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.

#### 6.1.3.5 DEFAULT_CLEAR_DEPTH

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

Definition at line 21 of file Renderer.hpp.

#### 6.1.3.6 DEFAULT_FAR

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

Definition at line 20 of file Camera.hpp.

Referenced by ven::Gui::cameraSection().

#### 6.1.3.7 DEFAULT_FOV

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

Definition at line 18 of file Camera.hpp.

Referenced by ven::Gui::cameraSection().

#### 6.1.3.8 DEFAULT_HEIGHT

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

Definition at line 18 of file Window.hpp.

#### 6.1.3.9 DEFAULT_LIGHT_COLOR

```
glm::vec4 ven::DEFAULT_LIGHT_COLOR = {glm::vec3(1.F), DEFAULT_LIGHT_INTENSITY}  [static],
[constexpr]
```

Definition at line 20 of file Light.hpp.

#### 6.1.3.10 DEFAULT_LIGHT_INTENSITY

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

Definition at line 18 of file Light.hpp.

Referenced by ven::Gui::lightsSection().

#### 6.1.3.11 DEFAULT_LIGHT_RADIUS

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

Definition at line 19 of file Light.hpp.

#### 6.1.3.12 DEFAULT_LOOK_SPEED

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

Definition at line 23 of file Camera.hpp.

Referenced by ven::Gui::cameraSection().

#### 6.1.3.13 DEFAULT_MAX_SETS

```
uint32_t ven::DEFAULT_MAX_SETS = 1000  [static], [constexpr]
```

Definition at line 15 of file DescriptorPool.hpp.

#### 6.1.3.14 DEFAULT_MOVE_SPEED

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

Definition at line 22 of file Camera.hpp.

Referenced by ven::Gui::cameraSection().

#### 6.1.3.15 DEFAULT_NEAR

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

Definition at line 19 of file Camera.hpp.

Referenced by ven::Gui::cameraSection().

#### 6.1.3.16 DEFAULT_POSITION

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

Definition at line 15 of file Camera.hpp.

Referenced by ven::Gui::cameraSection().

#### 6.1.3.17 DEFAULT_ROTATION

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

Definition at line 16 of file Camera.hpp.

Referenced by ven::Gui::cameraSection().

#### 6.1.3.18 DEFAULT_TITLE

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

Definition at line 19 of file Window.hpp.

#### 6.1.3.19 DEFAULT_WIDTH

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

Definition at line 17 of file Window.hpp.

#### 6.1.3.20 MAX_FRAMES_IN_FLIGHT

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

Definition at line 16 of file SwapChain.hpp.

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

#### 6.1.3.21 MAX_LIGHTS

```
uint16_t ven::MAX_LIGHTS = 10  [static], [constexpr]
```

Definition at line 17 of file FrameInfo.hpp.

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

#### 6.1.3.22 SHADERS_BIN_PATH

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

Definition at line 17 of file Shaders.hpp.

Referenced by ven::ObjectRenderSystem::ObjectRenderSystem(), and ven::PointLightRenderSystem::PointLightRenderSystem().

# Chapter 7

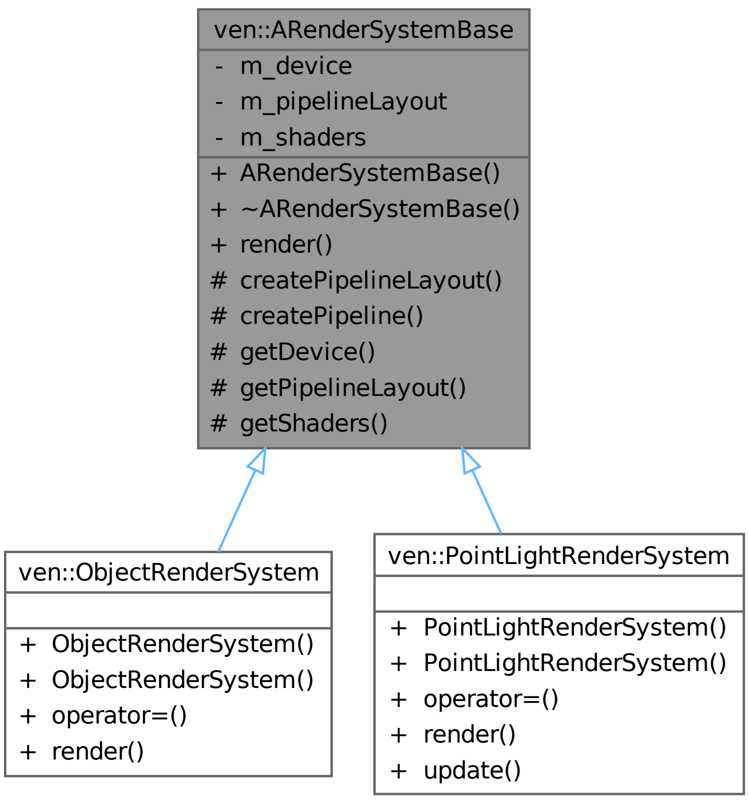
# Class Documentation

## 7.1 ven::ARenderSystemBase Class Reference

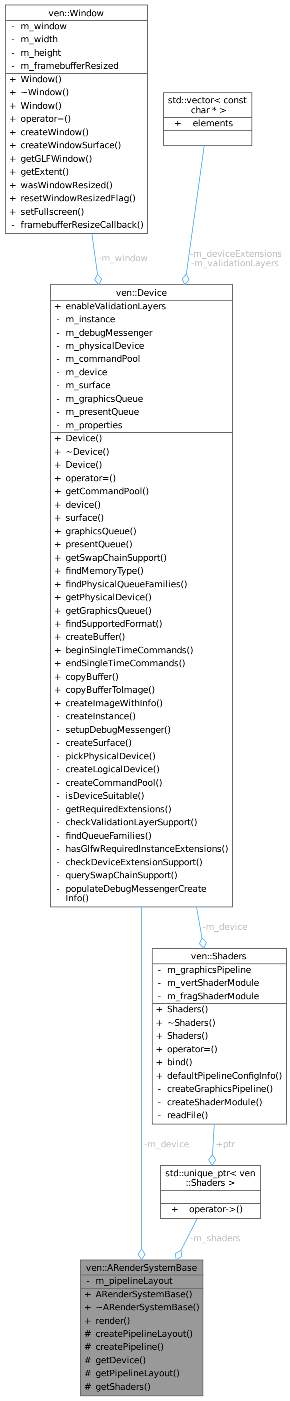
Abstract class for render system base.

```
#include <ARenderSystemBase.hpp>
```

Inheritance diagram for ven::ARenderSystemBase:



Collaboration diagram for ven::ARenderSystemBase:




## Public Member Functions

- ARenderSystemBase (Device &device)
- virtual ∼ARenderSystemBase ()
- virtual void render (const FrameInfo &frameInfo) const =0

#### Protected Member Functions

- void createPipelineLayout (VkDescriptorSetLayout globalSetLayout, uint32_t pushConstantSize)
- void createPipeline (VkRenderPass renderPass, const std::string &shadersVertPath, const std::string &shadersFragPath, bool isLight)
- Device & getDevice () const
- VkPipelineLayout getPipelineLayout () const
- const std::unique_ptr< Shaders > & getShaders () const

#### Private Attributes

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

## 7.1.1 Detailed Description

Abstract class for render system base.

Definition at line 22 of file ARenderSystemBase.hpp.

## 7.1.2 Constructor & Destructor Documentation

### 7.1.2.1 ARenderSystemBase()

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

Definition at line 26 of file ARenderSystemBase.hpp.

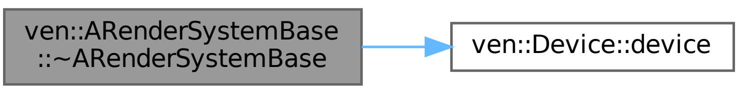
### 7.1.2.2 ∼ARenderSystemBase()

```
virtual ven::ARenderSystemBase::~ARenderSystemBase ()  [inline], [virtual]
```

Definition at line 27 of file ARenderSystemBase.hpp.

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

Here is the call graph for this function:



### 7.1.3 Member Function Documentation

#### 7.1.3.1 createPipeline()

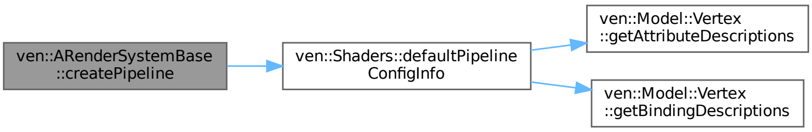
```
void ven::ARenderSystemBase::createPipeline (
            VkRenderPass renderPass,
            const std::string & shadersVertPath,
            const std::string & shadersFragPath,
            bool isLight)  [protected]
```

Definition at line 26 of file renderSystemBase.cpp.

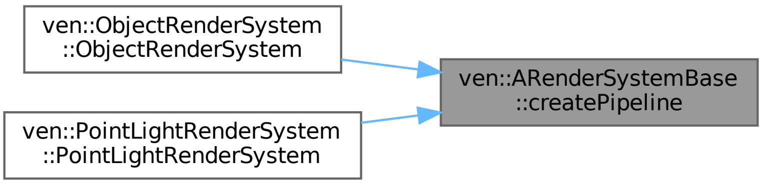
References ven::Shaders::defaultPipelineConfigInfo().

Referenced by ven::ObjectRenderSystem::ObjectRenderSystem(), and ven::PointLightRenderSystem::PointLightRenderSystem().

Here is the call graph for this function:




Here is the caller graph for this function:




#### 7.1.3.2 createPipelineLayout()

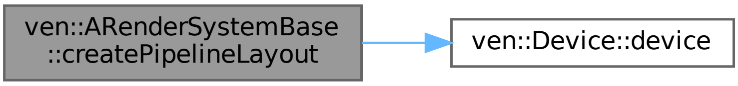
```
void ven::ARenderSystemBase::createPipelineLayout (
            VkDescriptorSetLayout globalSetLayout,
            uint32_t pushConstantSize)  [protected]
```

Definition at line 5 of file renderSystemBase.cpp.

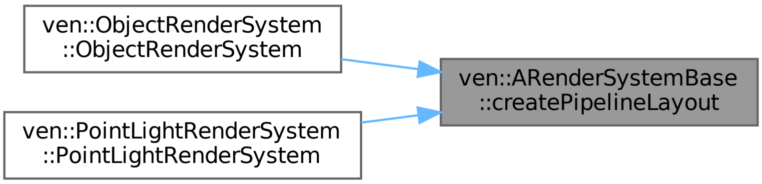
References ven::Device::device(), m_device, and m_pipelineLayout.

Referenced by ven::ObjectRenderSystem::ObjectRenderSystem(), and ven::PointLightRenderSystem::PointLightRenderSystem().

Here is the call graph for this function:




Here is the caller graph for this function:




#### 7.1.3.3 getDevice()

`Device & ven::ARenderSystemBase::getDevice () const [inline], [nodiscard], [protected]`

Definition at line 36 of file ARenderSystemBase.hpp.

References m_device.

#### 7.1.3.4 getPipelineLayout()

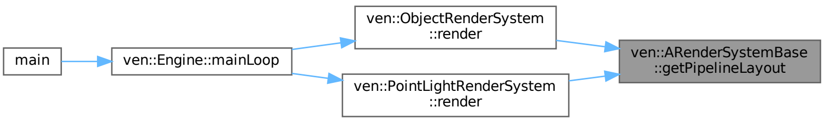
`VkPipelineLayout ven::ARenderSystemBase::getPipelineLayout () const [inline], [nodiscard], [protected]`

Definition at line 37 of file ARenderSystemBase.hpp.

References m_pipelineLayout.

Referenced by ven::ObjectRenderSystem::render(), and ven::PointLightRenderSystem::render().

Here is the caller graph for this function:



#### 7.1.3.5 getShaders()

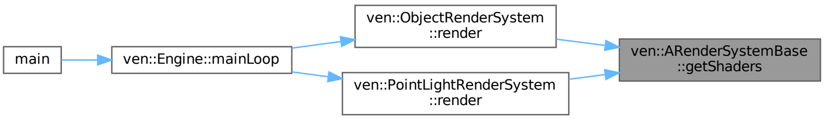
```
const std::unique_ptr< Shaders > & ven::ARenderSystemBase::getShaders () const [inline],
[nodiscard], [protected]
```

Definition at line 38 of file ARenderSystemBase.hpp.

References m_shaders.

Referenced by ven::ObjectRenderSystem::render(), and ven::PointLightRenderSystem::render().

Here is the caller graph for this function:




#### 7.1.3.6 render()

```
virtual void ven::ARenderSystemBase::render (
            const FrameInfo & frameInfo) const [pure virtual]
```

Implemented in ven::ObjectRenderSystem, and ven::PointLightRenderSystem.

### 7.1.4 Member Data Documentation

#### 7.1.4.1 m_device

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

Definition at line 42 of file ARenderSystemBase.hpp.

Referenced by createPipelineLayout(), getDevice(), and ∼ARenderSystemBase().

#### 7.1.4.2 m_pipelineLayout

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

Definition at line 43 of file ARenderSystemBase.hpp.

Referenced by createPipelineLayout(), getPipelineLayout(), and ∼ARenderSystemBase().

#### 7.1.4.3 m_shaders

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

Definition at line 44 of file ARenderSystemBase.hpp.

Referenced by getShaders().

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

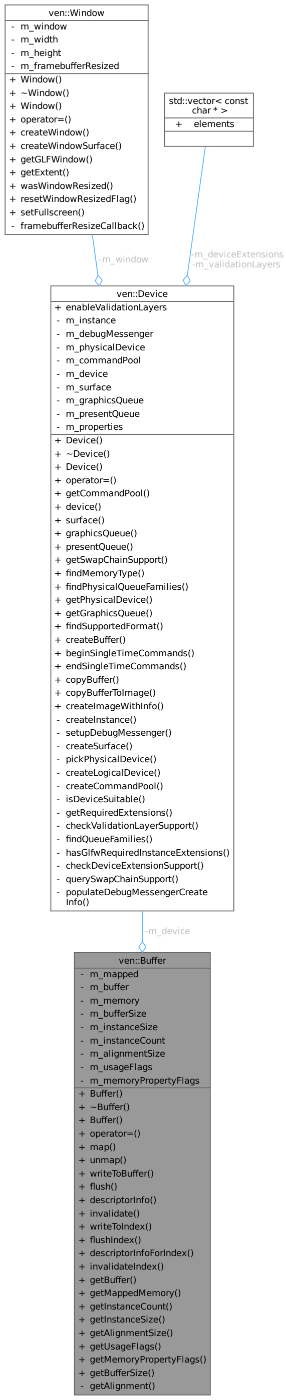
- /home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/ARenderSystemBase.hpp
- /home/runner/work/VEngine/VEngine/src/system/renderSystemBase.cpp

## 7.2 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

### 7.2.1 Detailed Description

Class for buffer.

Definition at line 18 of file Buffer.hpp.

### 7.2.2 Constructor & Destructor Documentation

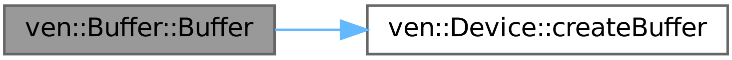
#### 7.2.2.1 Buffer() [1/2]

```
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:




#### 7.2.2.2 ∼Buffer()

```
ven::Buffer::∼Buffer ()
```

Definition at line 19 of file buffer.cpp.

#### 7.2.2.3 Buffer() [2/2]

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

### 7.2.3 Member Function Documentation

#### 7.2.3.1 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**

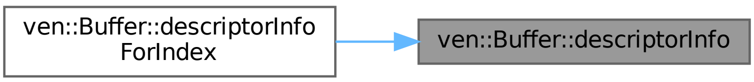
VkDescriptorBufferInfo of specified offset and range

Definition at line 74 of file Buffer.hpp.

References m_buffer.

Referenced by descriptorInfoForIndex().

Here is the caller graph for this function:




#### 7.2.3.2 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 |

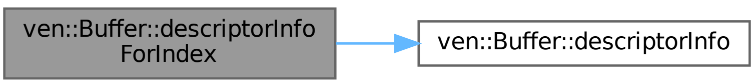
**Returns**

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:



#### 7.2.3.3 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 |

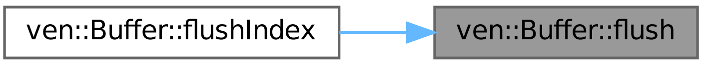
**Returns**

VkResult of the flush call

Definition at line 53 of file buffer.cpp.

Referenced by flushIndex().

Here is the caller graph for this function:




#### 7.2.3.4 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**

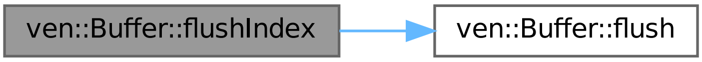
| | |
|---|---|
| *index* | Used in offset calculation |

---

Definition at line 103 of file Buffer.hpp.

References flush(), and m_alignmentSize.

Here is the call graph for this function:




#### 7.2.3.5 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.

#### 7.2.3.6 getAlignmentSize()

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

Definition at line 130 of file Buffer.hpp.

References m_alignmentSize.

#### 7.2.3.7 getBuffer()

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

Definition at line 126 of file Buffer.hpp.

References m_buffer.

#### 7.2.3.8 getBufferSize()

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

Definition at line 133 of file Buffer.hpp.

References m_bufferSize.

#### 7.2.3.9 getInstanceCount()

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

Definition at line 128 of file Buffer.hpp.

References m_instanceCount.

#### 7.2.3.10 getInstanceSize()

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

Definition at line 129 of file Buffer.hpp.

References m_instanceSize.

#### 7.2.3.11 getMappedMemory()

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

Definition at line 127 of file Buffer.hpp.

References m_mapped.

#### 7.2.3.12 getMemoryPropertyFlags()

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

Definition at line 132 of file Buffer.hpp.

References m_memoryPropertyFlags.

#### 7.2.3.13 getUsageFlags()

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

Definition at line 131 of file Buffer.hpp.

References m_usageFlags.

#### 7.2.3.14 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 |

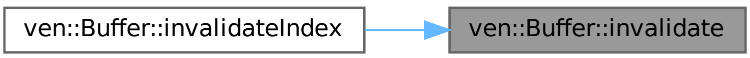
**Returns**

VkResult of the invalidate call

Definition at line 63 of file buffer.cpp.

Referenced by invalidateIndex().

Here is the caller graph for this function:




#### 7.2.3.15 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 |

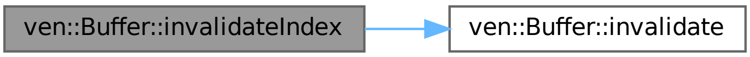
**Returns**

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:




### 7.2.3.16 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 |

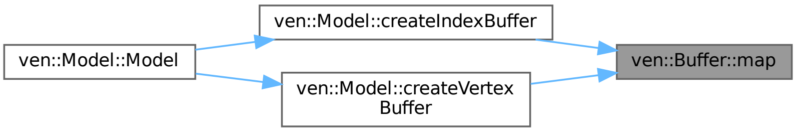
**Returns**

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:



#### 7.2.3.17 operator=()

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

#### 7.2.3.18 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.

#### 7.2.3.19 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

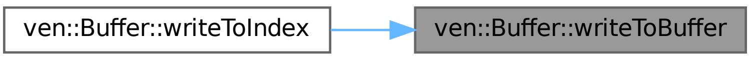
**Parameters**

| | |
|---|---|
| *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:




#### 7.2.3.20 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.

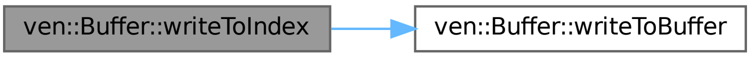
**Parameters**

| | |
|---|---|
| *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:




### 7.2.4 Member Data Documentation

#### 7.2.4.1 m_alignmentSize

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

Definition at line 155 of file Buffer.hpp.

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

#### 7.2.4.2 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().

#### 7.2.4.3 m_bufferSize

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

Definition at line 152 of file Buffer.hpp.

Referenced by Buffer(), and getBufferSize().

#### 7.2.4.4 m_device

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

Definition at line 147 of file Buffer.hpp.

#### 7.2.4.5 m_instanceCount

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

Definition at line 154 of file Buffer.hpp.

Referenced by Buffer(), and getInstanceCount().

#### 7.2.4.6 m_instanceSize

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

Definition at line 153 of file Buffer.hpp.

Referenced by getInstanceSize(), and writeToIndex().

#### 7.2.4.7 m_mapped

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

Definition at line 148 of file Buffer.hpp.

Referenced by getMappedMemory().

#### 7.2.4.8 m_memory

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

Definition at line 150 of file Buffer.hpp.

Referenced by Buffer().

#### 7.2.4.9 m_memoryPropertyFlags

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

Definition at line 157 of file Buffer.hpp.

Referenced by Buffer(), and getMemoryPropertyFlags().

#### 7.2.4.10 m_usageFlags

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

Definition at line 156 of file Buffer.hpp.

Referenced by Buffer(), and getUsageFlags().

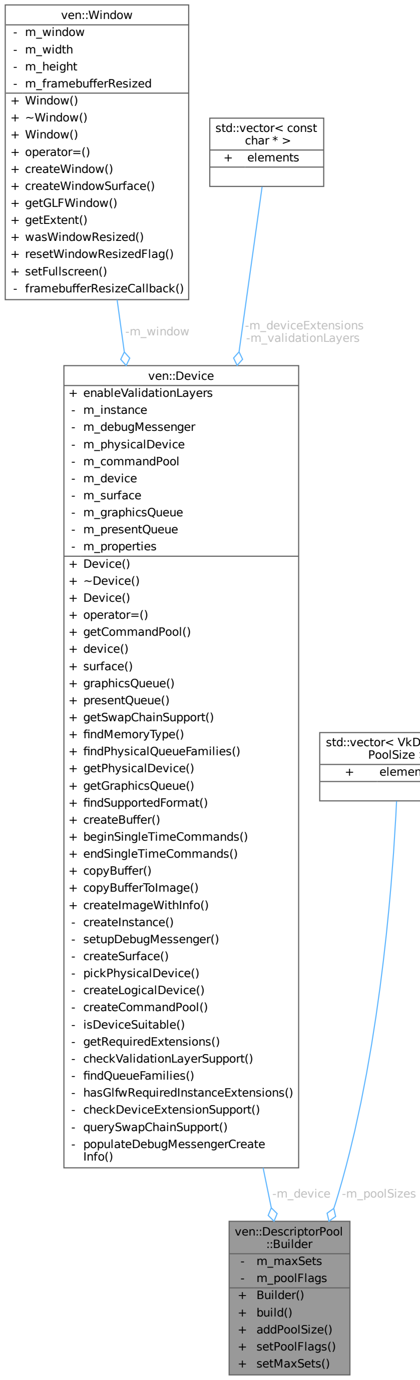
The documentation for this class was generated from the following files:

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

## 7.3 ven::DescriptorPool::Builder Class Reference

```
#include <DescriptorPool.hpp>
```

Collaboration diagram for ven::DescriptorPool::Builder:




## Public Member Functions

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

**Private Attributes**

- Device & m_device
- std::vector< VkDescriptorPoolSize > m_poolSizes
- uint32_t m_maxSets {DEFAULT_MAX_SETS}
- VkDescriptorPoolCreateFlags m_poolFlags {0}

## 7.3.1 Detailed Description

Definition at line 26 of file DescriptorPool.hpp.

## 7.3.2 Constructor & Destructor Documentation

### 7.3.2.1 Builder()

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

Definition at line 30 of file DescriptorPool.hpp.

## 7.3.3 Member Function Documentation

### 7.3.3.1 addPoolSize()

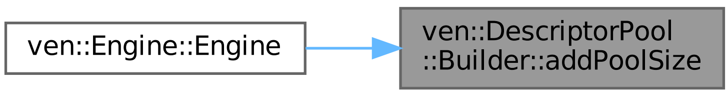
```
Builder & ven::DescriptorPool::Builder::addPoolSize (
            const VkDescriptorType descriptorType,
            const uint32_t count)  [inline]
```

Definition at line 34 of file DescriptorPool.hpp.

References m_poolSizes.

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

Here is the caller graph for this function:



#### 7.3.3.2 build()

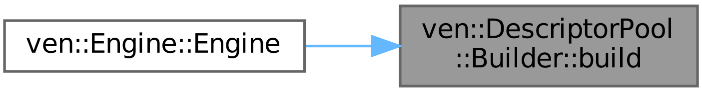
```
std::unique_ptr< DescriptorPool > ven::DescriptorPool::Builder::build () const  [inline],
[nodiscard]
```

Definition at line 32 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:




#### 7.3.3.3 setMaxSets()

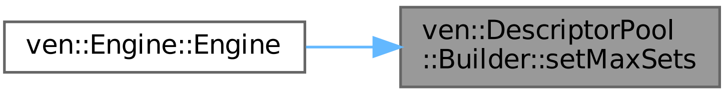
```
Builder & ven::DescriptorPool::Builder::setMaxSets (
            const uint32_t count)  [inline]
```

Definition at line 36 of file DescriptorPool.hpp.

References m_maxSets.

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

Here is the caller graph for this function:




#### 7.3.3.4 setPoolFlags()

```
Builder & ven::DescriptorPool::Builder::setPoolFlags (
            const VkDescriptorPoolCreateFlags flags)  [inline]
```

Definition at line 35 of file DescriptorPool.hpp.

References m_poolFlags.

## 7.3.4 Member Data Documentation

### 7.3.4.1 m_device

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

Definition at line 40 of file DescriptorPool.hpp.

Referenced by build().

### 7.3.4.2 m_maxSets

`uint32_t ven::DescriptorPool::Builder::m_maxSets {DEFAULT_MAX_SETS} [private]`

Definition at line 42 of file DescriptorPool.hpp.

Referenced by build(), and setMaxSets().

### 7.3.4.3 m_poolFlags

`VkDescriptorPoolCreateFlags ven::DescriptorPool::Builder::m_poolFlags {0} [private]`

Definition at line 43 of file DescriptorPool.hpp.

Referenced by build(), and setPoolFlags().

### 7.3.4.4 m_poolSizes

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

Definition at line 41 of file DescriptorPool.hpp.

Referenced by addPoolSize(), and build().

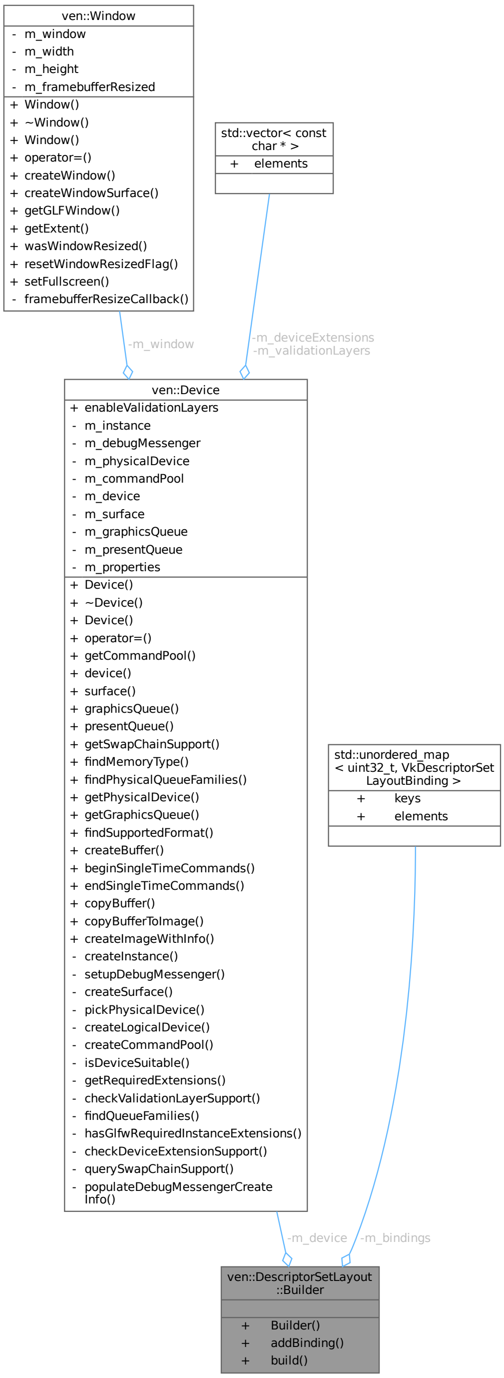
The documentation for this class was generated from the following file:

- /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorPool.hpp

## 7.4 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

## 7.4.1 Detailed Description

Definition at line 25 of file DescriptorSetLayout.hpp.

## 7.4.2 Constructor & Destructor Documentation

### 7.4.2.1 Builder()

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

Definition at line 29 of file DescriptorSetLayout.hpp.

## 7.4.3 Member Function Documentation

### 7.4.3.1 addBinding()

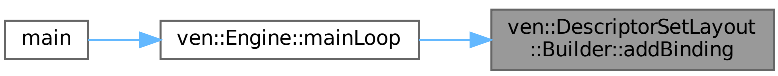
```
ven::DescriptorSetLayout::Builder & ven::DescriptorSetLayout::Builder::addBinding (
            uint32_t binding,
            VkDescriptorType descriptorType,
            VkShaderStageFlags stageFlags,
            uint32_t count = 1)
```

Definition at line 5 of file descriptorSetLayout.cpp.

References m_bindings.

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

Here is the caller graph for this function:



#### 7.4.3.2 build()

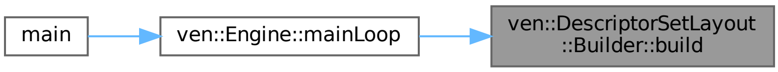
```
std::unique_ptr< DescriptorSetLayout > ven::DescriptorSetLayout::Builder::build () const
[inline]
```

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:




### 7.4.4 Member Data Documentation

#### 7.4.4.1 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().

#### 7.4.4.2 m_device

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

Definition at line 36 of file DescriptorSetLayout.hpp.

Referenced by build().

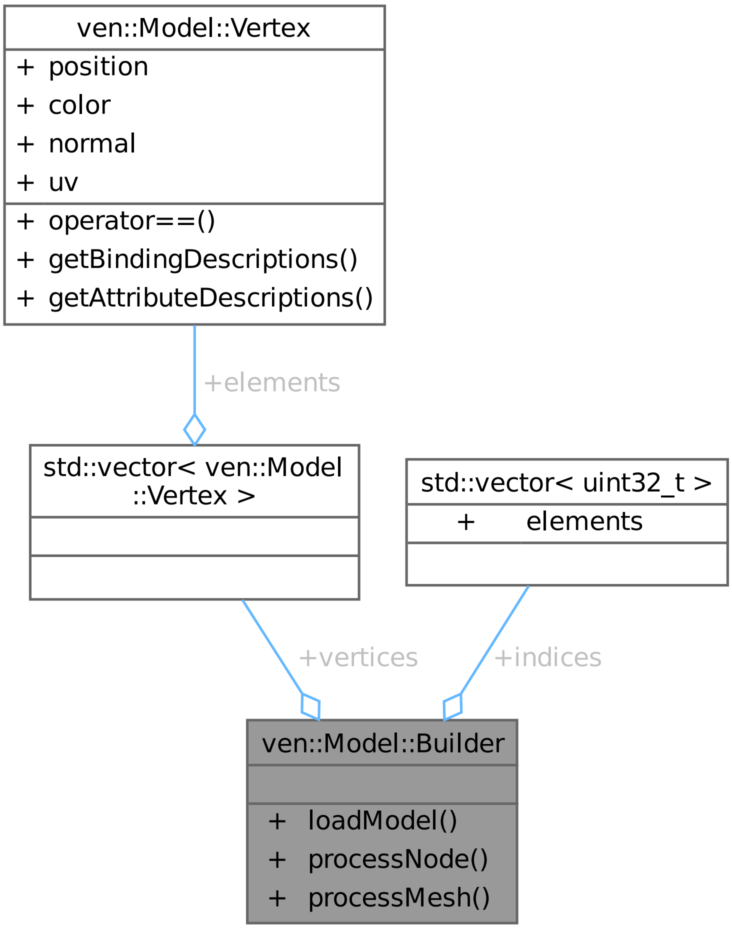
The documentation for this class was generated from the following files:

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

## 7.5 ven::Model::Builder Struct Reference

`#include <Model.hpp>`

Collaboration diagram for ven::Model::Builder:




### Public Member Functions

- void loadModel (const std::string &filename)
- void processNode (const aiNode ∗node, const aiScene ∗scene)
- void processMesh (const aiMesh ∗mesh, const aiScene ∗scene)

### Public Attributes

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

## 7.5.1 Detailed Description

Definition at line 43 of file Model.hpp.

## 7.5.2 Member Function Documentation

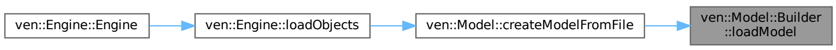
### 7.5.2.1 loadModel()

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

Definition at line 117 of file model.cpp.

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

Here is the caller graph for this function:




### 7.5.2.2 processMesh()

```
void ven::Model::Builder::processMesh (
            const aiMesh * mesh,
            const aiScene * scene)
```

Definition at line 143 of file model.cpp.

References ven::Colors::BLACK_3, ven::Model::Vertex::position, and ven::Colors::WHITE_3.

### 7.5.2.3 processNode()

```
void ven::Model::Builder::processNode (
            const aiNode * node,
            const aiScene * scene)
```

Definition at line 132 of file model.cpp.

## 7.5.3 Member Data Documentation

### 7.5.3.1 indices

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

Definition at line 45 of file Model.hpp.

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

#### 7.5.3.2 vertices

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

Definition at line 44 of file Model.hpp.

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

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

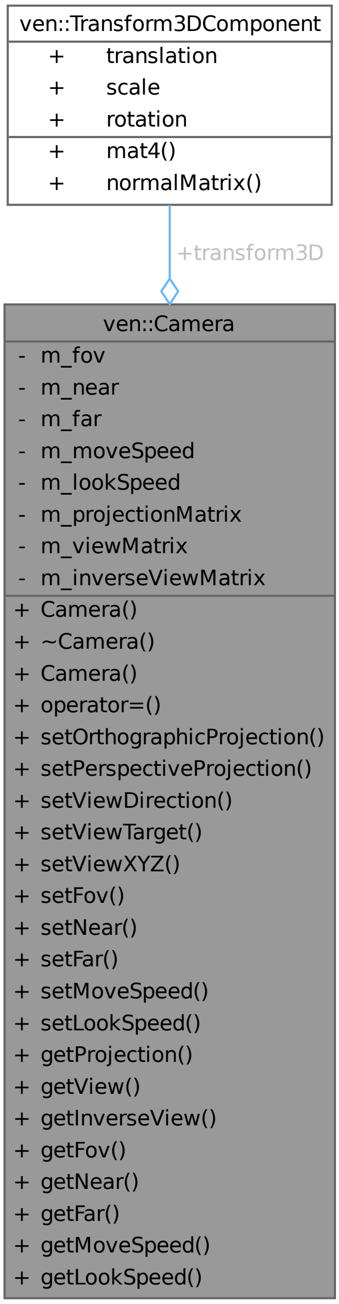
- /home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp
- /home/runner/work/VEngine/VEngine/src/model.cpp

## 7.6 ven::Camera Class Reference

Class for camera.

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

Collaboration diagram for ven::Camera:




### Public Member Functions

- Camera ()=default
- ∼Camera ()=default
- Camera (const Camera &)=delete
- Camera & operator= (const Camera &)=delete
- 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={0.F, -1.F, 0.F})
- void setViewTarget (const glm::vec3 position, const glm::vec3 target, const glm::vec3 up={0.F, -1.F, 0.F})
- void setViewXYZ (glm::vec3 position, glm::vec3 rotation)
- void setFov (const float fov)
- void setNear (const float near)
- void setFar (const float far)
- void setMoveSpeed (const float moveSpeed)
- void setLookSpeed (const float lookSpeed)
- const glm::mat4 & getProjection () const
- const glm::mat4 & getView () const
- const glm::mat4 & getInverseView () const
- float getFov () const
- float getNear () const
- float getFar () const
- float getMoveSpeed () const
- float getLookSpeed () const

**Public Attributes**

- Transform3DComponent transform3D {DEFAULT_POSITION, {1.F, 1.F, 1.F}, DEFAULT_ROTATION}

**Private Attributes**

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

## 7.6.1 Detailed Description

Class for camera.

Definition at line 30 of file Camera.hpp.

## 7.6.2 Constructor & Destructor Documentation

### 7.6.2.1 Camera() [1/2]

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

### 7.6.2.2 ~Camera()

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

#### 7.6.2.3 Camera() [2/2]

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

### 7.6.3 Member Function Documentation

#### 7.6.3.1 getFar()

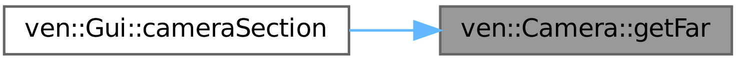
```
float ven::Camera::getFar () const  [inline], [nodiscard]
```

Definition at line 56 of file Camera.hpp.

References m_far.

Referenced by ven::Gui::cameraSection().

Here is the caller graph for this function:




#### 7.6.3.2 getFov()

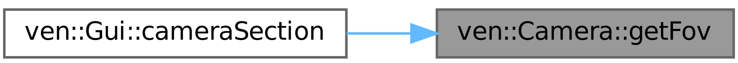
```
float ven::Camera::getFov () const  [inline], [nodiscard]
```

Definition at line 54 of file Camera.hpp.

References m_fov.

Referenced by ven::Gui::cameraSection().

Here is the caller graph for this function:



#### 7.6.3.3 getInverseView()

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

Definition at line 53 of file Camera.hpp.

References m_inverseViewMatrix.

#### 7.6.3.4 getLookSpeed()

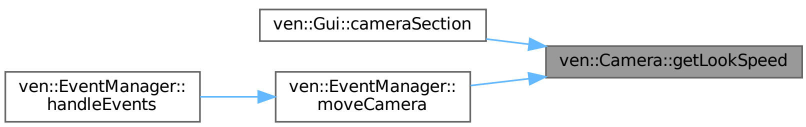
`float ven::Camera::getLookSpeed () const [inline], [nodiscard]`

Definition at line 58 of file Camera.hpp.

References m_lookSpeed.

Referenced by ven::Gui::cameraSection(), and ven::EventManager::moveCamera().

Here is the caller graph for this function:




#### 7.6.3.5 getMoveSpeed()

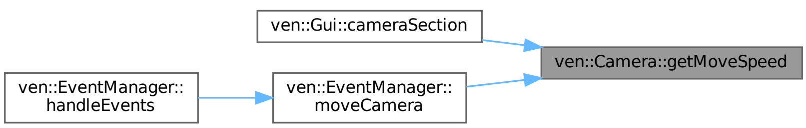
`float ven::Camera::getMoveSpeed () const [inline], [nodiscard]`

Definition at line 57 of file Camera.hpp.

References m_moveSpeed.

Referenced by ven::Gui::cameraSection(), and ven::EventManager::moveCamera().

Here is the caller graph for this function:



#### 7.6.3.6 getNear()

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

Definition at line 55 of file Camera.hpp.

References m_near.

Referenced by ven::Gui::cameraSection().

Here is the caller graph for this function:


#### 7.6.3.7 getProjection()

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

Definition at line 51 of file Camera.hpp.

References m_projectionMatrix.

#### 7.6.3.8 getView()

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

Definition at line 52 of file Camera.hpp.

References m_viewMatrix.

#### 7.6.3.9 operator=()

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

### 7.6.3.10 setFar()

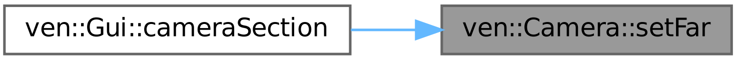
```
void ven::Camera::setFar (
            const float far)  [inline]
```

Definition at line 47 of file Camera.hpp.

References m_far.

Referenced by ven::Gui::cameraSection().

Here is the caller graph for this function:




### 7.6.3.11 setFov()

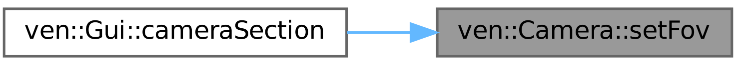
```
void ven::Camera::setFov (
            const float fov)  [inline]
```

Definition at line 45 of file Camera.hpp.

References m_fov.

Referenced by ven::Gui::cameraSection().

Here is the caller graph for this function:



#### 7.6.3.12 setLookSpeed()

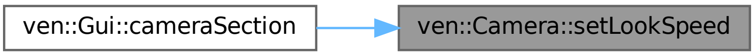
```
void ven::Camera::setLookSpeed (
            const float lookSpeed)  [inline]
```

Definition at line 49 of file Camera.hpp.

References m_lookSpeed.

Referenced by ven::Gui::cameraSection().

Here is the caller graph for this function:




#### 7.6.3.13 setMoveSpeed()

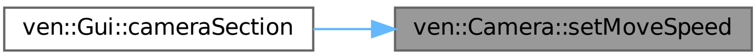
```
void ven::Camera::setMoveSpeed (
            const float moveSpeed)  [inline]
```

Definition at line 48 of file Camera.hpp.

References m_moveSpeed.

Referenced by ven::Gui::cameraSection().

Here is the caller graph for this function:



#### 7.6.3.14 setNear()

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

Definition at line 46 of file Camera.hpp.

References m_near.

Referenced by ven::Gui::cameraSection().

Here is the caller graph for this function:


#### 7.6.3.15 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.

#### 7.6.3.16 setPerspectiveProjection()

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

Definition at line 17 of file camera.cpp.

#### 7.6.3.17 setViewDirection()

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

Definition at line 29 of file camera.cpp.

#### 7.6.3.18 setViewTarget()

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

Definition at line 43 of file Camera.hpp.

#### 7.6.3.19 setViewXYZ()

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

Definition at line 64 of file camera.cpp.

### 7.6.4 Member Data Documentation

#### 7.6.4.1 m_far

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

Definition at line 66 of file Camera.hpp.

Referenced by getFar(), and setFar().

#### 7.6.4.2 m_fov

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

Definition at line 64 of file Camera.hpp.

Referenced by getFov(), and setFov().

#### 7.6.4.3 m_inverseViewMatrix

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

Definition at line 71 of file Camera.hpp.

Referenced by getInverseView().

#### 7.6.4.4 m_lookSpeed

```
float ven::Camera::m_lookSpeed {DEFAULT_LOOK_SPEED}  [private]
```

Definition at line 68 of file Camera.hpp.

Referenced by getLookSpeed(), and setLookSpeed().

#### 7.6.4.5 m_moveSpeed

```
float ven::Camera::m_moveSpeed {DEFAULT_MOVE_SPEED} [private]
```

Definition at line 67 of file Camera.hpp.

Referenced by getMoveSpeed(), and setMoveSpeed().

#### 7.6.4.6 m_near

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

Definition at line 65 of file Camera.hpp.

Referenced by getNear(), and setNear().

#### 7.6.4.7 m_projectionMatrix

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

Definition at line 69 of file Camera.hpp.

Referenced by getProjection(), and setOrthographicProjection().

#### 7.6.4.8 m_viewMatrix

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

Definition at line 70 of file Camera.hpp.

Referenced by getView().

#### 7.6.4.9 transform3D

```
Transform3DComponent ven::Camera::transform3D {DEFAULT_POSITION, {1.F, 1.F, 1.F}, DEFAULT_ROTATION}
```

Definition at line 60 of file Camera.hpp.

Referenced by ven::Gui::cameraSection(), and ven::EventManager::moveCamera().

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

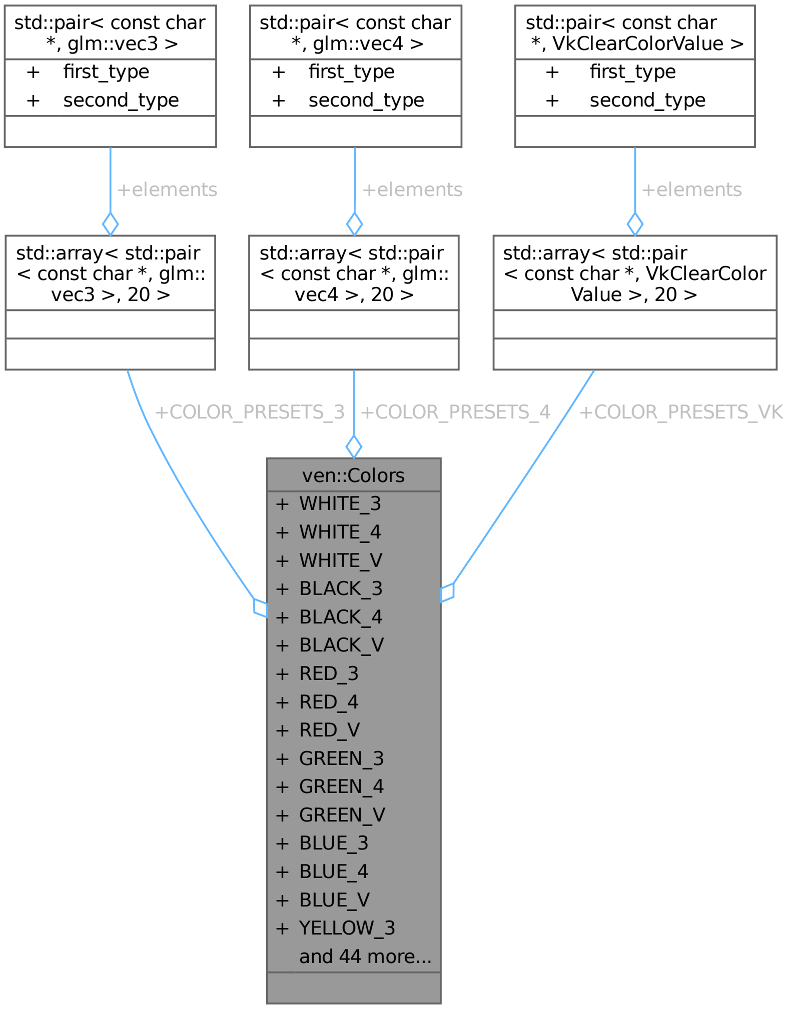
- /home/runner/work/VEngine/VEngine/include/VEngine/Camera.hpp
- /home/runner/work/VEngine/VEngine/src/camera.cpp

## 7.7 ven::Colors Class Reference

Class for colors.

```
#include <Colors.hpp>
```

Collaboration diagram for ven::Colors:




### Static Public Attributes

- static constexpr glm::vec3 WHITE_3 = glm::vec3(COLOR_MAX) / COLOR_MAX
- static constexpr glm::vec4 WHITE_4 = { 1.0F, 1.0F, 1.0F, 1.0F }
- static constexpr VkClearColorValue WHITE_V = { { 1.0F, 1.0F, 1.0F, 1.0F } }
- static constexpr glm::vec3 BLACK_3 = glm::vec3(0.0F)
- static constexpr glm::vec4 BLACK_4 = { 0.0F, 0.0F, 0.0F, 1.0F }
- static constexpr VkClearColorValue BLACK_V = { { 0.0F, 0.0F, 0.0F, 1.0F } }
- static constexpr glm::vec3 RED_3 = glm::vec3(COLOR_MAX, 0.0F, 0.0F) / COLOR_MAX
- static constexpr glm::vec4 RED_4 = { 1.0F, 0.0F, 0.0F, 1.0F }

- static constexpr VkClearColorValue RED_V = { { 1.0F, 0.0F, 0.0F, 1.0F } }
- static constexpr glm::vec3 GREEN_3 = glm::vec3(0.0F, COLOR_MAX, 0.0F) / COLOR_MAX
- static constexpr glm::vec4 GREEN_4 = { 0.0F, 1.0F, 0.0F, 1.0F }
- static constexpr VkClearColorValue GREEN_V = { { 0.0F, 1.0F, 0.0F, 1.0F } }
- static constexpr glm::vec3 BLUE_3 = glm::vec3(0.0F, 0.0F, COLOR_MAX) / COLOR_MAX
- static constexpr glm::vec4 BLUE_4 = { 0.0F, 0.0F, 1.0F, 1.0F }
- static constexpr VkClearColorValue BLUE_V = { { 0.0F, 0.0F, 1.0F, 1.0F } }
- static constexpr glm::vec3 YELLOW_3 = glm::vec3(COLOR_MAX, COLOR_MAX, 0.0F) / COLOR_MAX
- static constexpr glm::vec4 YELLOW_4 = { 1.0F, 1.0F, 0.0F, 1.0F }
- static constexpr VkClearColorValue YELLOW_V = { { 1.0F, 1.0F, 0.0F, 1.0F } }
- static constexpr glm::vec3 CYAN_3 = glm::vec3(0.0F, COLOR_MAX, COLOR_MAX) / COLOR_MAX
- static constexpr glm::vec4 CYAN_4 = { 0.0F, 1.0F, 1.0F, 1.0F }
- static constexpr VkClearColorValue CYAN_V = { { 0.0F, 1.0F, 1.0F, 1.0F } }
- static constexpr glm::vec3 MAGENTA_3 = glm::vec3(COLOR_MAX, 0.0F, COLOR_MAX) / COLOR_MAX
- static constexpr glm::vec4 MAGENTA_4 = { 1.0F, 0.0F, 1.0F, 1.0F }
- static constexpr VkClearColorValue MAGENTA_V = { { 1.0F, 0.0F, 1.0F, 1.0F } }
- static constexpr glm::vec3 SILVER_3 = glm::vec3(192.0F, 192.0F, 192.0F) / COLOR_MAX
- static constexpr glm::vec4 SILVER_4 = { 0.75F, 0.75F, 0.75F, 1.0F }
- static constexpr VkClearColorValue SILVER_V = { { 0.75F, 0.75F, 0.75F, 1.0F } }
- static constexpr glm::vec3 GRAY_3 = glm::vec3(128.0F, 128.0F, 128.0F) / COLOR_MAX
- static constexpr glm::vec4 GRAY_4 = { 0.5F, 0.5F, 0.5F, 1.0F }
- static constexpr VkClearColorValue GRAY_V = { { 0.5F, 0.5F, 0.5F, 1.0F } }
- static constexpr glm::vec3 MAROON_3 = glm::vec3(128.0F, 0.0F, 0.0F) / COLOR_MAX
- static constexpr glm::vec4 MAROON_4 = { 0.5F, 0.0F, 0.0F, 1.0F }
- static constexpr VkClearColorValue MAROON_V = { { 0.5F, 0.0F, 0.0F, 1.0F } }
- static constexpr glm::vec3 OLIVE_3 = glm::vec3(128.0F, 128.0F, 0.0F) / COLOR_MAX
- static constexpr glm::vec4 OLIVE_4 = { 0.5F, 0.5F, 0.0F, 1.0F }
- static constexpr VkClearColorValue OLIVE_V = { { 0.5F, 0.5F, 0.0F, 1.0F } }
- static constexpr glm::vec3 LIME_3 = glm::vec3(0.0F, COLOR_MAX, 0.0F) / COLOR_MAX
- static constexpr glm::vec4 LIME_4 = { 0.0F, 1.0F, 0.0F, 1.0F }
- static constexpr VkClearColorValue LIME_V = { { 0.0F, 1.0F, 0.0F, 1.0F } }
- static constexpr glm::vec3 AQUA_3 = glm::vec3(0.0F, COLOR_MAX, COLOR_MAX) / COLOR_MAX
- static constexpr glm::vec4 AQUA_4 = { 0.0F, 1.0F, 1.0F, 1.0F }
- static constexpr VkClearColorValue AQUA_V = { { 0.0F, 1.0F, 1.0F, 1.0F } }
- static constexpr glm::vec3 TEAL_3 = glm::vec3(0.0F, 128.0F, 128.0F) / COLOR_MAX
- static constexpr glm::vec4 TEAL_4 = { 0.0F, 0.5F, 0.5F, 1.0F }
- static constexpr VkClearColorValue TEAL_V = { { 0.0F, 0.5F, 0.5F, 1.0F } }
- static constexpr glm::vec3 NAVY_3 = glm::vec3(0.0F, 0.0F, 128.0F) / COLOR_MAX
- static constexpr glm::vec4 NAVY_4 = { 0.0F, 0.0F, 0.5F, 1.0F }
- static constexpr VkClearColorValue NAVY_V = { { 0.0F, 0.0F, 0.5F, 1.0F } }
- static constexpr glm::vec3 FUCHSIA_3 = glm::vec3(COLOR_MAX, 0.0F, COLOR_MAX) / COLOR_MAX
- static constexpr glm::vec4 FUCHSIA_4 = { 1.0F, 0.0F, 1.0F, 1.0F }
- static constexpr VkClearColorValue FUCHSIA_V = { { 1.0F, 0.0F, 1.0F, 1.0F } }
- static constexpr glm::vec3 NIGHT_BLUE_3 = glm::vec3(25.0F, 25.0F, 112.0F) / COLOR_MAX
- static constexpr glm::vec4 NIGHT_BLUE_4 = { 0.098F, 0.098F, 0.439F, 1.0F }
- static constexpr VkClearColorValue NIGHT_BLUE_V = { { 0.098F, 0.098F, 0.439F, 1.0F } }
- static constexpr glm::vec3 SKY_BLUE_3 = glm::vec3(102.0F, 178.0F, 255.0F) / COLOR_MAX
- static constexpr glm::vec4 SKY_BLUE_4 = { 0.4F, 0.698F, 1.0F, 1.0F }
- static constexpr VkClearColorValue SKY_BLUE_V = { { 0.4F, 0.698F, 1.0F, 1.0F } }
- static constexpr glm::vec3 SUNSET_3 = glm::vec3(255.0F, 128.0F, 0.0F) / COLOR_MAX
- static constexpr glm::vec4 SUNSET_4 = { 1.0F, 0.5F, 0.0F, 1.0F }
- static constexpr VkClearColorValue SUNSET_V = { { 1.0F, 0.5F, 0.0F, 1.0F } }
- static constexpr std::array< std::pair< const char *, glm::vec3 >, 20 > COLOR_PRESETS_3
- static constexpr std::array< std::pair< const char *, glm::vec4 >, 20 > COLOR_PRESETS_4
- static constexpr std::array< std::pair< const char *, VkClearColorValue >, 20 > COLOR_PRESETS_VK

### 7.7.1 Detailed Description

Class for colors.

Definition at line 22 of file Colors.hpp.

### 7.7.2 Member Data Documentation

#### 7.7.2.1 AQUA_3

```
glm::vec3 ven::Colors::AQUA_3 = glm::vec3(0.0F, COLOR_MAX, COLOR_MAX) / COLOR_MAX  [static],
[constexpr]
```

Definition at line 78 of file Colors.hpp.

#### 7.7.2.2 AQUA_4

```
glm::vec4 ven::Colors::AQUA_4 = { 0.0F, 1.0F, 1.0F, 1.0F }  [static], [constexpr]
```

Definition at line 79 of file Colors.hpp.

#### 7.7.2.3 AQUA_V

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

Definition at line 80 of file Colors.hpp.

#### 7.7.2.4 BLACK_3

```
glm::vec3 ven::Colors::BLACK_3 = glm::vec3(0.0F)  [static], [constexpr]
```

Definition at line 30 of file Colors.hpp.

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

#### 7.7.2.5 BLACK_4

```
glm::vec4 ven::Colors::BLACK_4 = { 0.0F, 0.0F, 0.0F, 1.0F }  [static], [constexpr]
```

Definition at line 31 of file Colors.hpp.

#### 7.7.2.6 BLACK_V

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

Definition at line 32 of file Colors.hpp.

### 7.7.2.7 BLUE_3

```
glm::vec3 ven::Colors::BLUE_3 = glm::vec3(0.0F, 0.0F, COLOR_MAX) / COLOR_MAX  [static], [constexpr]
```

Definition at line 42 of file Colors.hpp.

### 7.7.2.8 BLUE_4

```
glm::vec4 ven::Colors::BLUE_4 = { 0.0F, 0.0F, 1.0F, 1.0F }  [static], [constexpr]
```

Definition at line 43 of file Colors.hpp.

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

### 7.7.2.9 BLUE_V

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

Definition at line 44 of file Colors.hpp.

### 7.7.2.10 COLOR_PRESETS_3

```
std::array<std::pair<const char *, glm::vec3>, 20> ven::Colors::COLOR_PRESETS_3  [static],
[constexpr]
```

**Initial value:**

```
= {{
                {"White", WHITE_3},
                {"Black", BLACK_3},
                {"Red", RED_3},
                {"Green", GREEN_3},
                {"Blue", BLUE_3},
                {"Yellow", YELLOW_3},
                {"Cyan", CYAN_3},
                {"Magenta", MAGENTA_3},
                {"Silver", SILVER_3},
                {"Gray", GRAY_3},
                {"Maroon", MAROON_3},
                {"Olive", OLIVE_3},
                {"Lime", LIME_3},
                {"Aqua", AQUA_3},
                {"Teal", TEAL_3},
                {"Navy", NAVY_3},
                {"Fuchsia", FUCHSIA_3},
                {"Night Blue", NIGHT_BLUE_3},
                {"Sky Blue", SKY_BLUE_3},
                {"Sunset", SUNSET_3}
            }}
```

Definition at line 107 of file Colors.hpp.

Referenced by ven::Gui::lightsSection().

#### 7.7.2.11 COLOR_PRESETS_4

```
std::array<std::pair<const char *, glm::vec4>, 20> ven::Colors::COLOR_PRESETS_4 [static],
[constexpr]
```

**Initial value:**

```
= {{
                {"White", WHITE_4},
                {"Black", BLACK_4},
                {"Red", RED_4},
                {"Green", GREEN_4},
                {"Blue", BLUE_4},
                {"Yellow", YELLOW_4},
                {"Cyan", CYAN_4},
                {"Magenta", MAGENTA_4},
                {"Silver", SILVER_4},
                {"Gray", GRAY_4},
                {"Maroon", MAROON_4},
                {"Olive", OLIVE_4},
                {"Lime", LIME_4},
                {"Aqua", AQUA_4},
                {"Teal", TEAL_4},
                {"Navy", NAVY_4},
                {"Fuchsia", FUCHSIA_4},
                {"Night Blue", NIGHT_BLUE_4},
                {"Sky Blue", SKY_BLUE_4},
                {"Sunset", SUNSET_4}
            }}
```

Definition at line 130 of file Colors.hpp.

Referenced by ven::Gui::rendererSection().

#### 7.7.2.12 COLOR_PRESETS_VK

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

**Initial value:**

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

Definition at line 153 of file Colors.hpp.

Referenced by ven::Gui::rendererSection().

#### 7.7.2.13 CYAN_3

```
glm::vec3 ven::Colors::CYAN_3 = glm::vec3(0.0F, COLOR_MAX, COLOR_MAX) / COLOR_MAX [static],
[constexpr]
```

Definition at line 50 of file Colors.hpp.

#### 7.7.2.14 CYAN_4

```
glm::vec4 ven::Colors::CYAN_4 = { 0.0F, 1.0F, 1.0F, 1.0F } [static], [constexpr]
```

Definition at line 51 of file Colors.hpp.

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

#### 7.7.2.15 CYAN_V

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

Definition at line 52 of file Colors.hpp.

#### 7.7.2.16 FUCHSIA_3

```
glm::vec3 ven::Colors::FUCHSIA_3 = glm::vec3(COLOR_MAX, 0.0F, COLOR_MAX) / COLOR_MAX [static],
[constexpr]
```

Definition at line 90 of file Colors.hpp.

#### 7.7.2.17 FUCHSIA_4

```
glm::vec4 ven::Colors::FUCHSIA_4 = { 1.0F, 0.0F, 1.0F, 1.0F } [static], [constexpr]
```

Definition at line 91 of file Colors.hpp.

#### 7.7.2.18 FUCHSIA_V

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

Definition at line 92 of file Colors.hpp.

#### 7.7.2.19 GRAY_3

```
glm::vec3 ven::Colors::GRAY_3 = glm::vec3(128.0F, 128.0F, 128.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 62 of file Colors.hpp.

#### 7.7.2.20 GRAY_4

```
glm::vec4 ven::Colors::GRAY_4 = { 0.5F, 0.5F, 0.5F, 1.0F } [static], [constexpr]
```

Definition at line 63 of file Colors.hpp.

Referenced by ven::Gui::objectsSection().

#### 7.7.2.21 GRAY_V

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

Definition at line 64 of file Colors.hpp.

#### 7.7.2.22 GREEN_3

```
glm::vec3 ven::Colors::GREEN_3 = glm::vec3(0.0F, COLOR_MAX, 0.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 38 of file Colors.hpp.

#### 7.7.2.23 GREEN_4

```
glm::vec4 ven::Colors::GREEN_4 = { 0.0F, 1.0F, 0.0F, 1.0F } [static], [constexpr]
```

Definition at line 39 of file Colors.hpp.

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

#### 7.7.2.24 GREEN_V

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

Definition at line 40 of file Colors.hpp.

#### 7.7.2.25 LIME_3

```
glm::vec3 ven::Colors::LIME_3 = glm::vec3(0.0F, COLOR_MAX, 0.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 74 of file Colors.hpp.

#### 7.7.2.26 LIME_4

```
glm::vec4 ven::Colors::LIME_4 = { 0.0F, 1.0F, 0.0F, 1.0F } [static], [constexpr]
```

Definition at line 75 of file Colors.hpp.

#### 7.7.2.27 LIME_V

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

Definition at line 76 of file Colors.hpp.

#### 7.7.2.28 MAGENTA_3

```
glm::vec3 ven::Colors::MAGENTA_3 = glm::vec3(COLOR_MAX, 0.0F, COLOR_MAX) / COLOR_MAX [static],
[constexpr]
```

Definition at line 54 of file Colors.hpp.

#### 7.7.2.29 MAGENTA_4

```
glm::vec4 ven::Colors::MAGENTA_4 = { 1.0F, 0.0F, 1.0F, 1.0F } [static], [constexpr]
```

Definition at line 55 of file Colors.hpp.

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

#### 7.7.2.30 MAGENTA_V

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

Definition at line 56 of file Colors.hpp.

#### 7.7.2.31 MAROON_3

```
glm::vec3 ven::Colors::MAROON_3 = glm::vec3(128.0F, 0.0F, 0.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 66 of file Colors.hpp.

#### 7.7.2.32 MAROON_4

```
glm::vec4 ven::Colors::MAROON_4 = { 0.5F, 0.0F, 0.0F, 1.0F } [static], [constexpr]
```

Definition at line 67 of file Colors.hpp.

#### 7.7.2.33 MAROON_V

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

Definition at line 68 of file Colors.hpp.

#### 7.7.2.34 NAVY_3

```
glm::vec3 ven::Colors::NAVY_3 = glm::vec3(0.0F, 0.0F, 128.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 86 of file Colors.hpp.

#### 7.7.2.35 NAVY_4

```
glm::vec4 ven::Colors::NAVY_4 = { 0.0F, 0.0F, 0.5F, 1.0F } [static], [constexpr]
```

Definition at line 87 of file Colors.hpp.

#### 7.7.2.36 NAVY_V

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

Definition at line 88 of file Colors.hpp.

#### 7.7.2.37 NIGHT_BLUE_3

```
glm::vec3 ven::Colors::NIGHT_BLUE_3 = glm::vec3(25.0F, 25.0F, 112.0F) / COLOR_MAX [static],
[constexpr]
```

Definition at line 94 of file Colors.hpp.

#### 7.7.2.38 NIGHT_BLUE_4

```
glm::vec4 ven::Colors::NIGHT_BLUE_4 = { 0.098F, 0.098F, 0.439F, 1.0F } [static], [constexpr]
```

Definition at line 95 of file Colors.hpp.

#### 7.7.2.39 NIGHT_BLUE_V

```
VkClearColorValue ven::Colors::NIGHT_BLUE_V = { { 0.098F, 0.098F, 0.439F, 1.0F } } [static],
[constexpr]
```

Definition at line 96 of file Colors.hpp.

#### 7.7.2.40 OLIVE_3

```
glm::vec3 ven::Colors::OLIVE_3 = glm::vec3(128.0F, 128.0F, 0.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 70 of file Colors.hpp.

#### 7.7.2.41 OLIVE_4

```
glm::vec4 ven::Colors::OLIVE_4 = { 0.5F, 0.5F, 0.0F, 1.0F } [static], [constexpr]
```

Definition at line 71 of file Colors.hpp.

#### 7.7.2.42 OLIVE_V

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

Definition at line 72 of file Colors.hpp.

#### 7.7.2.43 RED_3

```
glm::vec3 ven::Colors::RED_3 = glm::vec3(COLOR_MAX, 0.0F, 0.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 34 of file Colors.hpp.

#### 7.7.2.44 RED_4

```
glm::vec4 ven::Colors::RED_4 = { 1.0F, 0.0F, 0.0F, 1.0F } [static], [constexpr]
```

Definition at line 35 of file Colors.hpp.

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

#### 7.7.2.45 RED_V

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

Definition at line 36 of file Colors.hpp.

#### 7.7.2.46 SILVER_3

```
glm::vec3 ven::Colors::SILVER_3 = glm::vec3(192.0F, 192.0F, 192.0F) / COLOR_MAX [static],
[constexpr]
```

Definition at line 58 of file Colors.hpp.

#### 7.7.2.47 SILVER_4

```
glm::vec4 ven::Colors::SILVER_4 = { 0.75F, 0.75F, 0.75F, 1.0F } [static], [constexpr]
```

Definition at line 59 of file Colors.hpp.

#### 7.7.2.48 SILVER_V

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

Definition at line 60 of file Colors.hpp.

#### 7.7.2.49 SKY_BLUE_3

```
glm::vec3 ven::Colors::SKY_BLUE_3 = glm::vec3(102.0F, 178.0F, 255.0F) / COLOR_MAX [static],
[constexpr]
```

Definition at line 98 of file Colors.hpp.

#### 7.7.2.50 SKY_BLUE_4

```
glm::vec4 ven::Colors::SKY_BLUE_4 = { 0.4F, 0.698F, 1.0F, 1.0F } [static], [constexpr]
```

Definition at line 99 of file Colors.hpp.

#### 7.7.2.51 SKY_BLUE_V

```
VkClearColorValue ven::Colors::SKY_BLUE_V = { { 0.4F, 0.698F, 1.0F, 1.0F } } [static], [constexpr]
```

Definition at line 100 of file Colors.hpp.

#### 7.7.2.52 SUNSET_3

```
glm::vec3 ven::Colors::SUNSET_3 = glm::vec3(255.0F, 128.0F, 0.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 102 of file Colors.hpp.

#### 7.7.2.53 SUNSET_4

```
glm::vec4 ven::Colors::SUNSET_4 = { 1.0F, 0.5F, 0.0F, 1.0F } [static], [constexpr]
```

Definition at line 103 of file Colors.hpp.

#### 7.7.2.54 SUNSET_V

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

Definition at line 104 of file Colors.hpp.

#### 7.7.2.55 TEAL_3

```
glm::vec3 ven::Colors::TEAL_3 = glm::vec3(0.0F, 128.0F, 128.0F) / COLOR_MAX [static], [constexpr]
```

Definition at line 82 of file Colors.hpp.

#### 7.7.2.56 TEAL_4

```
glm::vec4 ven::Colors::TEAL_4 = { 0.0F, 0.5F, 0.5F, 1.0F } [static], [constexpr]
```

Definition at line 83 of file Colors.hpp.

#### 7.7.2.57 TEAL_V

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

Definition at line 84 of file Colors.hpp.

#### 7.7.2.58 WHITE_3

```
glm::vec3 ven::Colors::WHITE_3 = glm::vec3(COLOR_MAX) / COLOR_MAX [static], [constexpr]
```

Definition at line 26 of file Colors.hpp.

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

#### 7.7.2.59 WHITE_4

```
glm::vec4 ven::Colors::WHITE_4 = { 1.0F, 1.0F, 1.0F, 1.0F } [static], [constexpr]
```

Definition at line 27 of file Colors.hpp.

#### 7.7.2.60 WHITE_V

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

Definition at line 28 of file Colors.hpp.

#### 7.7.2.61 YELLOW_3

```
glm::vec3 ven::Colors::YELLOW_3 = glm::vec3(COLOR_MAX, COLOR_MAX, 0.0F) / COLOR_MAX [static],
[constexpr]
```

Definition at line 46 of file Colors.hpp.

#### 7.7.2.62 YELLOW_4

```
glm::vec4 ven::Colors::YELLOW_4 = { 1.0F, 1.0F, 0.0F, 1.0F } [static], [constexpr]
```

Definition at line 47 of file Colors.hpp.

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

#### 7.7.2.63 YELLOW_V

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

Definition at line 48 of file Colors.hpp.

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

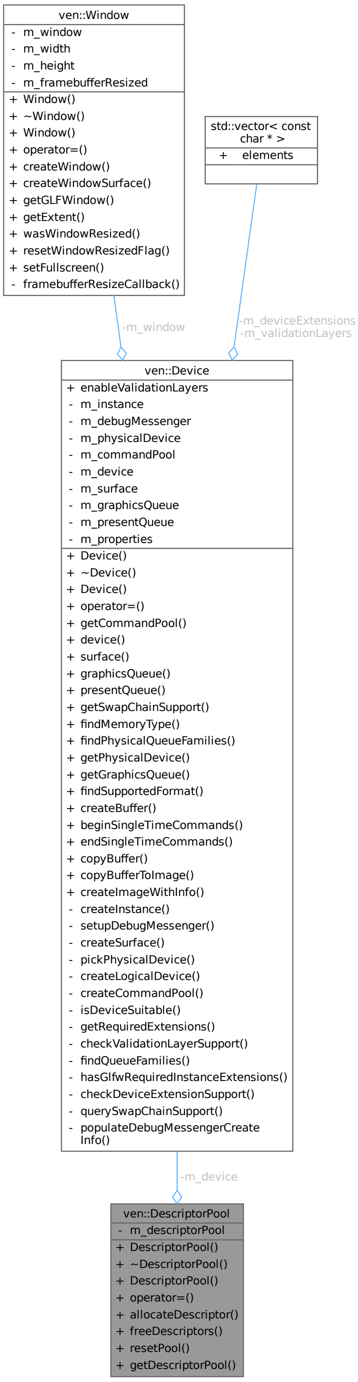
- /home/runner/work/VEngine/VEngine/include/VEngine/Colors.hpp

## 7.8 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

### 7.8.1 Detailed Description

Class for descriptor pool.

Definition at line 22 of file DescriptorPool.hpp.

### 7.8.2 Constructor & Destructor Documentation

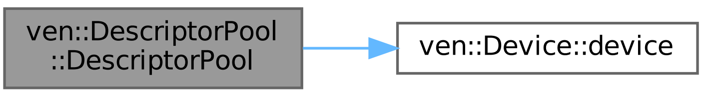
#### 7.8.2.1 DescriptorPool() [1/2]

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

Definition at line 3 of file descriptorPool.cpp.

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

Here is the call graph for this function:



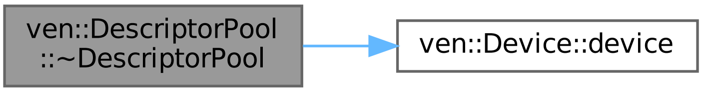
#### 7.8.2.2 ∼DescriptorPool()

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

Definition at line 48 of file DescriptorPool.hpp.

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

Here is the call graph for this function:




#### 7.8.2.3 DescriptorPool() [2/2]

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

### 7.8.3 Member Function Documentation

#### 7.8.3.1 allocateDescriptor()

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

Definition at line 18 of file descriptorPool.cpp.

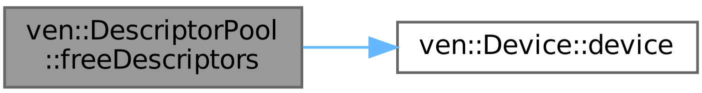
#### 7.8.3.2 freeDescriptors()

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

Definition at line 54 of file DescriptorPool.hpp.

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

Here is the call graph for this function:



#### 7.8.3.3 getDescriptorPool()

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

Definition at line 57 of file DescriptorPool.hpp.

References m_descriptorPool.

#### 7.8.3.4 operator=()

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

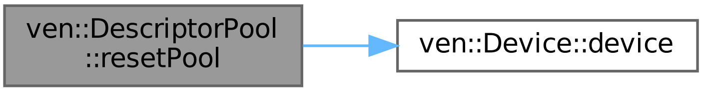
#### 7.8.3.5 resetPool()

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

Definition at line 55 of file DescriptorPool.hpp.

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

Here is the call graph for this function:




### 7.8.4 Friends And Related Symbol Documentation

#### 7.8.4.1 DescriptorWriter

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

Definition at line 63 of file DescriptorPool.hpp.

### 7.8.5 Member Data Documentation

#### 7.8.5.1 m_descriptorPool

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

Definition at line 62 of file DescriptorPool.hpp.

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

#### 7.8.5.2 m_device

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

Definition at line 61 of file DescriptorPool.hpp.

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

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

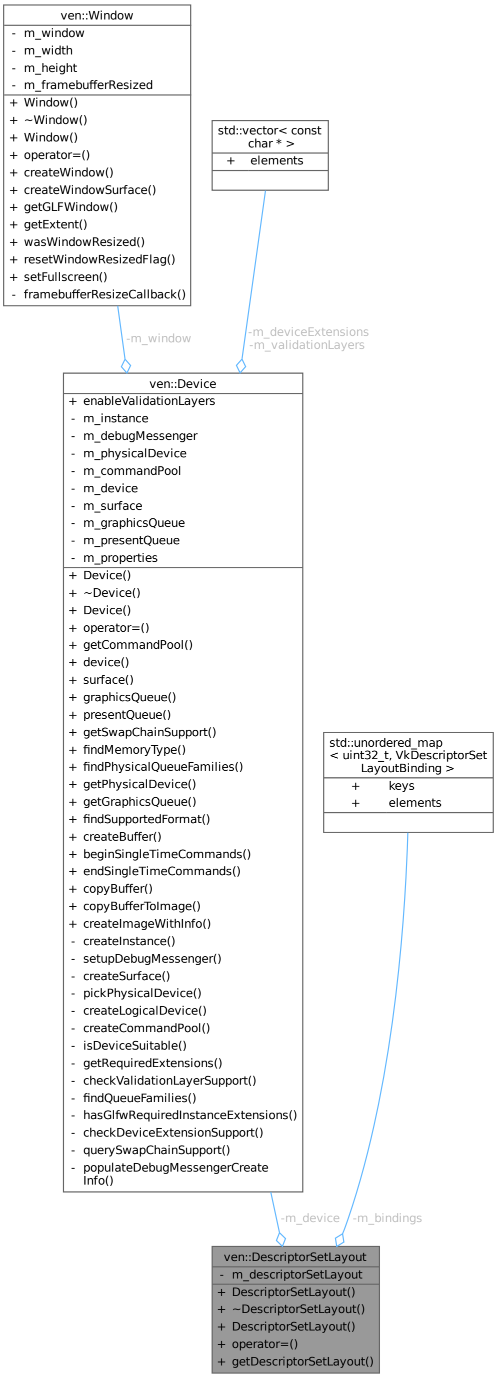
- /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorPool.hpp
- /home/runner/work/VEngine/VEngine/src/descriptors/descriptorPool.cpp

## 7.9 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

## 7.9.1 Detailed Description

Class for descriptor set layout.

Definition at line 21 of file DescriptorSetLayout.hpp.

## 7.9.2 Constructor & Destructor Documentation

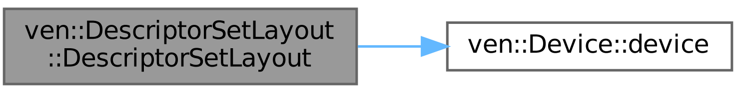
### 7.9.2.1 DescriptorSetLayout() [1/2]

```
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:



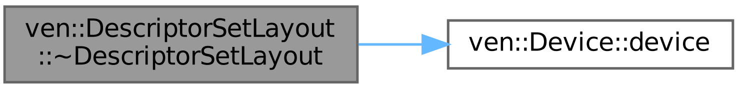
#### 7.9.2.2 ∼DescriptorSetLayout()

```
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:




#### 7.9.2.3 DescriptorSetLayout() [2/2]

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

### 7.9.3 Member Function Documentation

#### 7.9.3.1 getDescriptorSetLayout()

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

Definition at line 47 of file DescriptorSetLayout.hpp.

References m_descriptorSetLayout.

#### 7.9.3.2 operator=()

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

### 7.9.4 Friends And Related Symbol Documentation

#### 7.9.4.1 DescriptorWriter

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

Definition at line 55 of file DescriptorSetLayout.hpp.

### 7.9.5 Member Data Documentation

#### 7.9.5.1 m_bindings

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

Definition at line 53 of file DescriptorSetLayout.hpp.

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

#### 7.9.5.2 m_descriptorSetLayout

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

Definition at line 52 of file DescriptorSetLayout.hpp.

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

#### 7.9.5.3 m_device

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

Definition at line 51 of file DescriptorSetLayout.hpp.

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

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

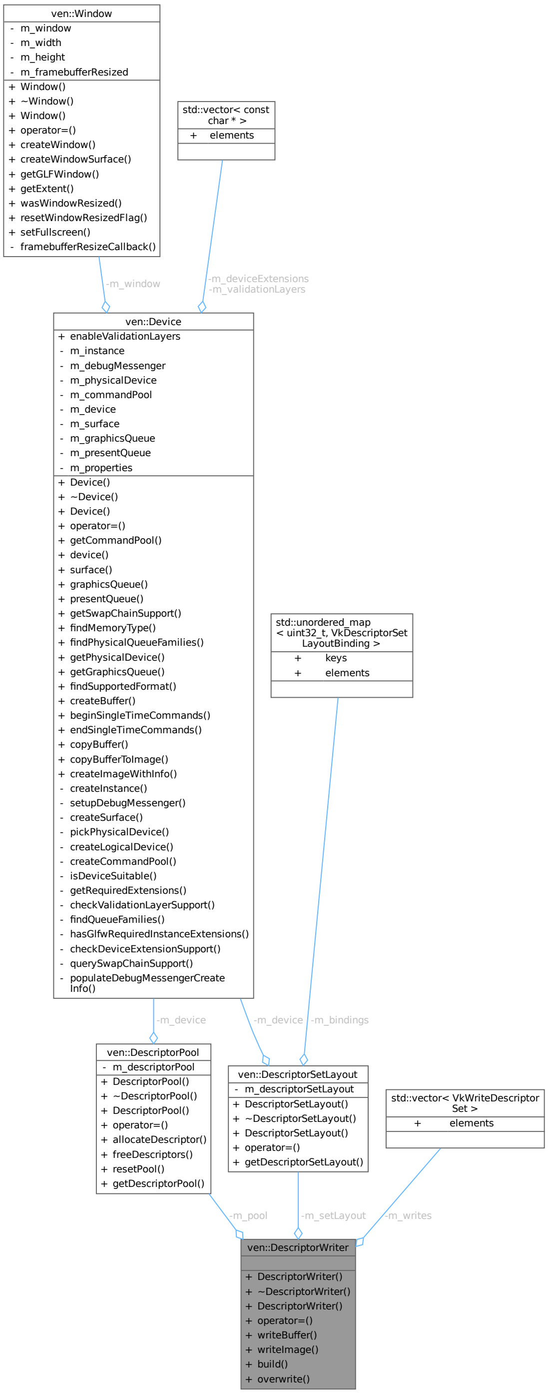
- /home/runner/work/VEngine/VEngine/include/VEngine/Descriptors/DescriptorSetLayout.hpp
- /home/runner/work/VEngine/VEngine/src/descriptors/descriptorSetLayout.cpp

## 7.10 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 ()=default
- DescriptorWriter (const DescriptorWriter &)=delete
- DescriptorWriter & operator= (const DescriptorWriter &)=delete
- 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

## 7.10.1 Detailed Description

Class for descriptor writer.

Definition at line 21 of file DescriptorWriter.hpp.

## 7.10.2 Constructor & Destructor Documentation

### 7.10.2.1 DescriptorWriter() [1/2]

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

Definition at line 25 of file DescriptorWriter.hpp.

### 7.10.2.2 ∼DescriptorWriter()

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

### 7.10.2.3 DescriptorWriter() [2/2]

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

## 7.10.3 Member Function Documentation

### 7.10.3.1 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:

#### 7.10.3.2 operator=()

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

#### 7.10.3.3 overwrite()

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

Definition at line 52 of file descriptorWriter.cpp.

#### 7.10.3.4 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:


#### 7.10.3.5 writeImage()

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

Definition at line 24 of file descriptorWriter.cpp.

### 7.10.4 Member Data Documentation

#### 7.10.4.1 m_pool

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

Definition at line 40 of file DescriptorWriter.hpp.

#### 7.10.4.2 m_setLayout

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

Definition at line 39 of file DescriptorWriter.hpp.

Referenced by writeBuffer().

#### 7.10.4.3 m_writes

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

Definition at line 41 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

## 7.11 ven::Device Class Reference

Class for device.

`#include <Device.hpp>`

Collaboration diagram for ven::Device:


## Public Member Functions

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

### Public Attributes

- const bool enableValidationLayers = true

### 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
- VkPhysicalDeviceProperties m_properties
- const std::vector< const char * > m_validationLayers = {"VK_LAYER_KHRONOS_validation"}
- const std::vector< const char * > m_deviceExtensions = {VK_KHR_SWAPCHAIN_EXTENSION_NAME}

## 7.11.1 Detailed Description

Class for device.

Definition at line 34 of file Device.hpp.

## 7.11.2 Constructor & Destructor Documentation

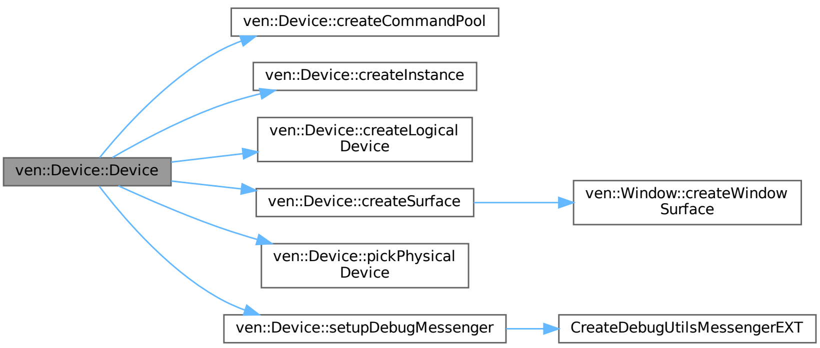
### 7.11.2.1 Device() [1/2]

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

Definition at line 32 of file device.cpp.

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

Here is the call graph for this function:



#### 7.11.2.2 ∼Device()

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

Definition at line 42 of file device.cpp.

References DestroyDebugUtilsMessengerEXT().

Here is the call graph for this function:


#### 7.11.2.3 Device() [2/2]

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

### 7.11.3 Member Function Documentation

#### 7.11.3.1 beginSingleTimeCommands()

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

Definition at line 408 of file device.cpp.

#### 7.11.3.2 checkDeviceExtensionSupport()

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

Definition at line 286 of file device.cpp.

#### 7.11.3.3 checkValidationLayerSupport()

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

Definition at line 223 of file device.cpp.

#### 7.11.3.4 copyBuffer()

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

Definition at line 442 of file device.cpp.

#### 7.11.3.5 copyBufferToImage()

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

Definition at line 455 of file device.cpp.

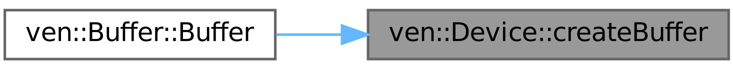
#### 7.11.3.6 createBuffer()

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

Definition at line 381 of file device.cpp.

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

Here is the caller graph for this function:



#### 7.11.3.7 createCommandPool()

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

Definition at line 169 of file device.cpp.

References ven::QueueFamilyIndices::graphicsFamily.

Referenced by Device().

Here is the caller graph for this function:


#### 7.11.3.8 createImageWithInfo()

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

Definition at line 476 of file device.cpp.

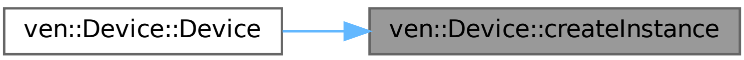
#### 7.11.3.9 createInstance()

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

Definition at line 55 of file device.cpp.

Referenced by Device().

Here is the caller graph for this function:



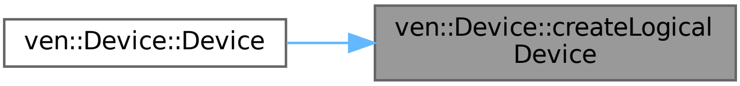
#### 7.11.3.10 createLogicalDevice()

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

Definition at line 122 of file device.cpp.

Referenced by Device().

Here is the caller graph for this function:




#### 7.11.3.11 createSurface()

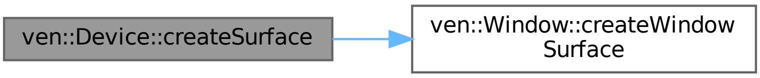
```
void ven::Device::createSurface ( ) [inline], [private]
```

Definition at line 77 of file Device.hpp.

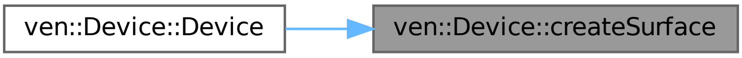
References ven::Window::createWindowSurface(), m_instance, m_surface, and m_window.

Referenced by Device().

Here is the call graph for this function:




Here is the caller graph for this function:



#### 7.11.3.12 device()

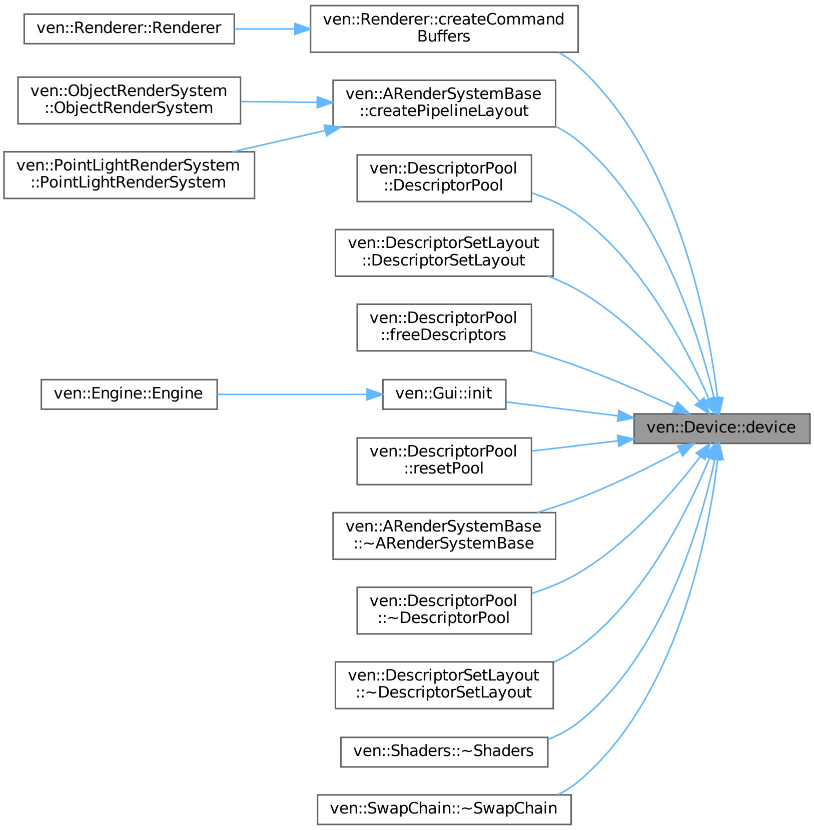
```
VkDevice ven::Device::device () const  [inline], [nodiscard]
```

Definition at line 51 of file Device.hpp.

References m_device.

Referenced by ven::Renderer::createCommandBuffers(), ven::ARenderSystemBase::createPipelineLayout(), ven::DescriptorPool::DescriptorPool(), ven::DescriptorSetLayout::DescriptorSetLayout(), ven::DescriptorPool::freeDescriptors(), ven::Gui::init(), ven::DescriptorPool::resetPool(), ven::ARenderSystemBase::~ARenderSystemBase(), ven::DescriptorPool::~Descrip ven::DescriptorSetLayout::~DescriptorSetLayout(), ven::Shaders::~Shaders(), and ven::SwapChain::~SwapChain().

Here is the caller graph for this function:




#### 7.11.3.13 endSingleTimeCommands()

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

Definition at line 427 of file device.cpp.

#### 7.11.3.14 findMemoryType()

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

Definition at line 366 of file device.cpp.

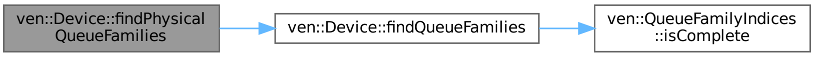
#### 7.11.3.15 findPhysicalQueueFamilies()

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

Definition at line 58 of file Device.hpp.

References findQueueFamilies(), and m_physicalDevice.

Here is the call graph for this function:




#### 7.11.3.16 findQueueFamilies()

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

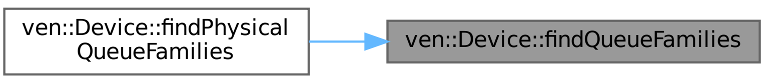
Definition at line 302 of file device.cpp.

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:



### 7.11.3.17 findSupportedFormat()

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

Definition at line 352 of file device.cpp.

### 7.11.3.18 getCommandPool()

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

Definition at line 50 of file Device.hpp.

References m_commandPool.

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

Here is the caller graph for this function:


### 7.11.3.19 getGraphicsQueue()

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

Definition at line 60 of file Device.hpp.

References m_graphicsQueue.

### 7.11.3.20 getPhysicalDevice()

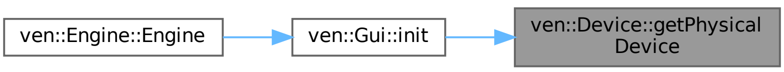
```
VkPhysicalDevice ven::Device::getPhysicalDevice () const  [inline], [nodiscard]
```

Definition at line 59 of file Device.hpp.

References m_physicalDevice.

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

Here is the caller graph for this function:



#### 7.11.3.21 getRequiredExtensions()

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

Definition at line 248 of file device.cpp.

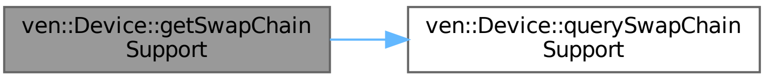
#### 7.11.3.22 getSwapChainSupport()

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

Definition at line 56 of file Device.hpp.

References m_physicalDevice, and querySwapChainSupport().

Here is the call graph for this function:




#### 7.11.3.23 graphicsQueue()

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

Definition at line 53 of file Device.hpp.

References m_graphicsQueue.

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

Here is the caller graph for this function:


#### 7.11.3.24 hasGlfwRequiredInstanceExtensions()

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

Definition at line 263 of file device.cpp.

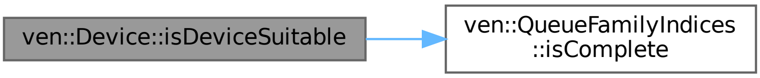
#### 7.11.3.25 isDeviceSuitable()

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

Definition at line 183 of file device.cpp.

References ven::QueueFamilyIndices::isComplete().

Here is the call graph for this function:




#### 7.11.3.26 operator=()

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

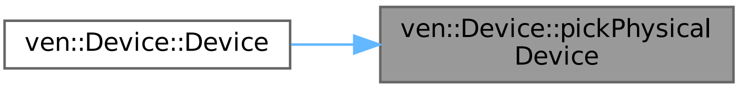
#### 7.11.3.27 pickPhysicalDevice()

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

Definition at line 96 of file device.cpp.

Referenced by Device().

Here is the caller graph for this function:



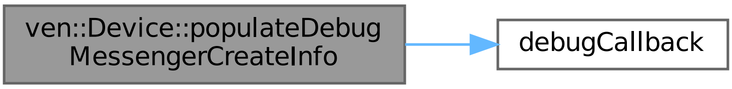
#### 7.11.3.28 populateDebugMessengerCreateInfo()

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

Definition at line 200 of file device.cpp.

References debugCallback().

Here is the call graph for this function:




#### 7.11.3.29 presentQueue()

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

Definition at line 54 of file Device.hpp.

References m_presentQueue.

#### 7.11.3.30 querySwapChainSupport()

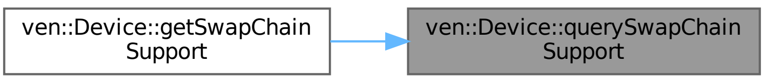
```
ven::SwapChainSupportDetails ven::Device::querySwapChainSupport (
            VkPhysicalDevice device) const  [private]
```

Definition at line 331 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:



#### 7.11.3.31 setupDebugMessenger()

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

Definition at line 213 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:


#### 7.11.3.32 surface()

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

Definition at line 52 of file Device.hpp.

References m_surface.

### 7.11.4 Member Data Documentation

#### 7.11.4.1 enableValidationLayers

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

Definition at line 41 of file Device.hpp.

#### 7.11.4.2 m_commandPool

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

Definition at line 96 of file Device.hpp.

Referenced by getCommandPool().

#### 7.11.4.3 m_debugMessenger

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

Definition at line 93 of file Device.hpp.

#### 7.11.4.4 m_device

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

Definition at line 98 of file Device.hpp.

Referenced by device().

#### 7.11.4.5 m_deviceExtensions

```
const std::vector<const char *> ven::Device::m_deviceExtensions = {VK_KHR_SWAPCHAIN_EXTENSION↩
_NAME} [private]
```

Definition at line 105 of file Device.hpp.

#### 7.11.4.6 m_graphicsQueue

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

Definition at line 100 of file Device.hpp.

Referenced by getGraphicsQueue(), and graphicsQueue().

#### 7.11.4.7 m_instance

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

Definition at line 92 of file Device.hpp.

Referenced by createSurface().

#### 7.11.4.8 m_physicalDevice

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

Definition at line 94 of file Device.hpp.

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

#### 7.11.4.9 m_presentQueue

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

Definition at line 101 of file Device.hpp.

Referenced by presentQueue().

#### 7.11.4.10 m_properties

```
VkPhysicalDeviceProperties ven::Device::m_properties [private]
```

Definition at line 102 of file Device.hpp.

#### 7.11.4.11 m_surface

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

Definition at line 99 of file Device.hpp.

Referenced by createSurface(), and surface().

#### 7.11.4.12 m_validationLayers

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

Definition at line 104 of file Device.hpp.

#### 7.11.4.13 m_window

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

Definition at line 95 of file Device.hpp.

Referenced by createSurface().

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

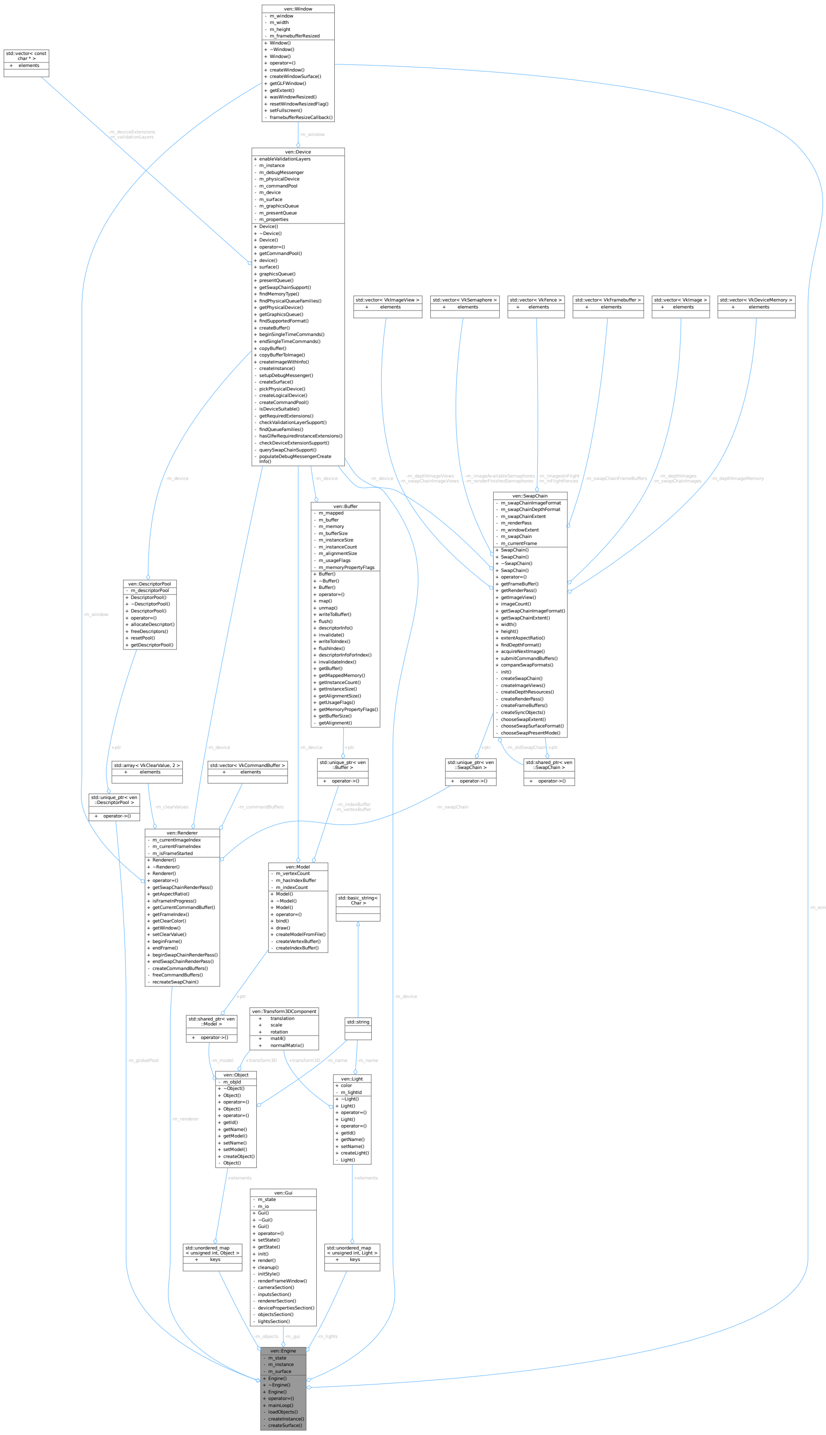
- /home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp
- /home/runner/work/VEngine/VEngine/src/device.cpp

# 7.12 ven::Engine Class Reference

Class for engine.

```
#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

- ENGINE_STATE m_state {EXIT}
- Window m_window
- Device m_device {m_window}
- Renderer m_renderer {m_window, m_device}
- Gui m_gui
- std::unique_ptr< DescriptorPool > m_globalPool
- Object::Map m_objects
- Light::Map m_lights
- VkInstance m_instance {nullptr}
- VkSurfaceKHR m_surface {nullptr}

## 7.12.1 Detailed Description

Class for engine.

Definition at line 33 of file Engine.hpp.

## 7.12.2 Constructor & Destructor Documentation

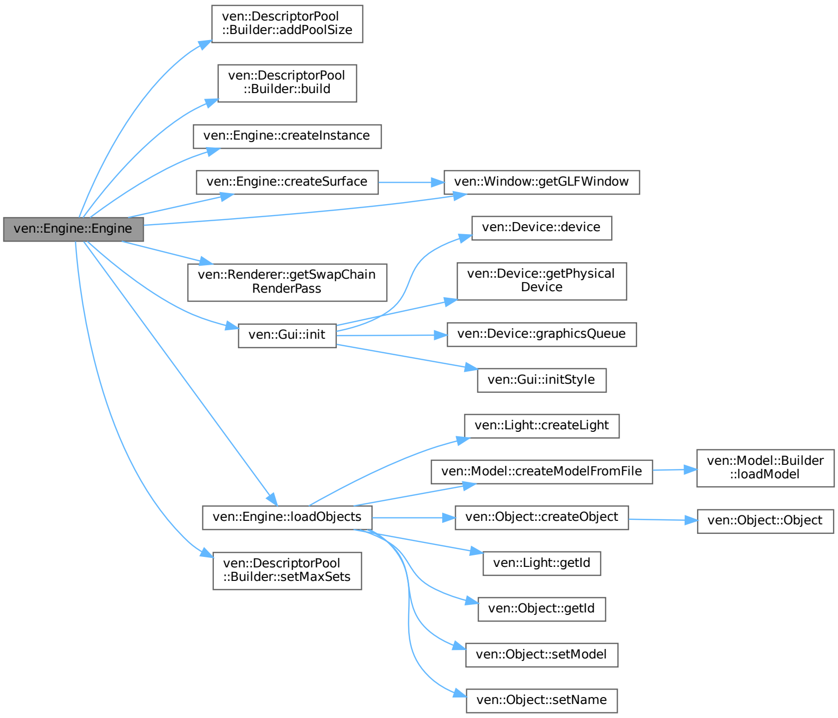
### 7.12.2.1 Engine() [1/2]

```
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::EDITOR, ven::Window::getGLFWindow(), ven::Renderer::getSwapChainRenderPass(), ven::Gui::init(), loadObjects(), m_device, m_globalPool, m_instance, m_renderer, m_window, ven::MAX_FRAMES_IN_FLIGHT, and ven::DescriptorPool::Builder::setMaxSets().

Here is the call graph for this function:




#### 7.12.2.2 ∼Engine()

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

#### 7.12.2.3 Engine() [2/2]

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

### 7.12.3 Member Function Documentation

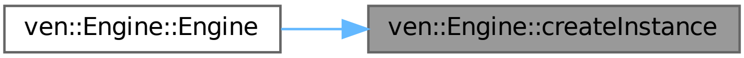
#### 7.12.3.1 createInstance()

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

Definition at line 23 of file engine.cpp.

Referenced by Engine().

Here is the caller graph for this function:




#### 7.12.3.2 createSurface()

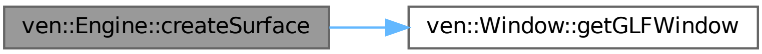
```
void ven::Engine::createSurface ()  [inline], [private]
```

Definition at line 63 of file Engine.hpp.

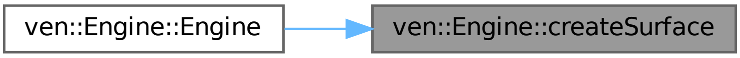
References ven::Window::getGLFWindow(), m_instance, m_surface, and m_window.

Referenced by Engine().

Here is the call graph for this function:




Here is the caller graph for this function:




#### 7.12.3.3 loadObjects()

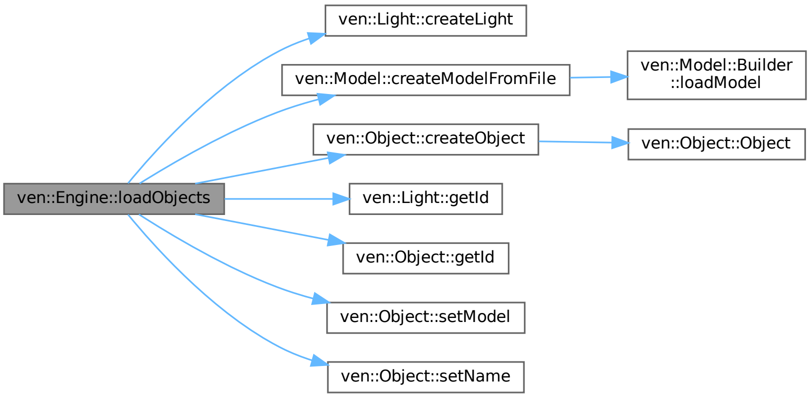
```
void ven::Engine::loadObjects ()  [private]
```

Definition at line 42 of file engine.cpp.

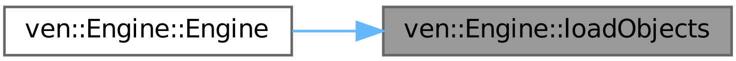
References ven::Colors::BLUE_4, ven::Light::color, ven::Light::createLight(), ven::Model::createModelFromFile(), ven::Object::createObject(), ven::Colors::CYAN_4, ven::Light::getId(), ven::Object::getId(), ven::Colors::GREEN_4, ven::Colors::MAGENTA_4, ven::Colors::RED_4, ven::Transform3DComponent::scale, ven::Object::setModel(), ven::Object::setName(), ven::Light::transform3D, ven::Object::transform3D, ven::Transform3DComponent::translation, and ven::Colors::YELLOW_4.

Referenced by Engine().

Here is the call graph for this function:




Here is the caller graph for this function:




### 7.12.3.4 mainLoop()

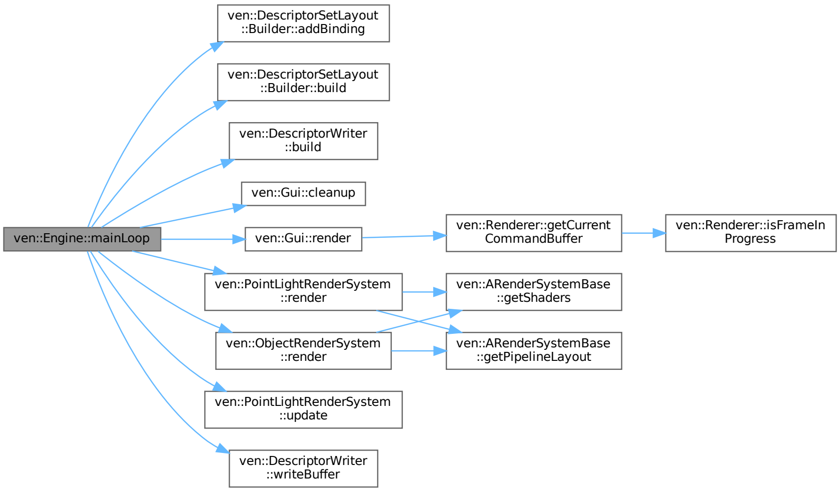
```
void ven::Engine::mainLoop ()
```

Definition at line 87 of file engine.cpp.

References ven::DescriptorSetLayout::Builder::addBinding(), ven::DescriptorSetLayout::Builder::build(), ven::DescriptorWriter::build(), ven::Gui::cleanup(), ven::EXIT, ven::FrameInfo::frameIndex, ven::MAX_FRAMES_IN_FLIGHT, ven::Gui::render(), ven::ObjectRenderSystem::render(), ven::PointLightRenderSystem::render(), ven::PointLightRenderSystem::update(), ven::VISIBLE, and ven::DescriptorWriter::writeBuffer().

Referenced by main().

Here is the call graph for this function:




Here is the caller graph for this function:


#### 7.12.3.5 operator=()

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

### 7.12.4 Member Data Documentation

#### 7.12.4.1 m_device

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

Definition at line 52 of file Engine.hpp.

Referenced by Engine().

#### 7.12.4.2 m_globalPool

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

Definition at line 55 of file Engine.hpp.

Referenced by Engine().

#### 7.12.4.3 m_gui

```
Gui ven::Engine::m_gui [private]
```

Definition at line 54 of file Engine.hpp.

#### 7.12.4.4 m_instance

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

Definition at line 59 of file Engine.hpp.

Referenced by createSurface(), and Engine().

#### 7.12.4.5 m_lights

```
Light::Map ven::Engine::m_lights [private]
```

Definition at line 57 of file Engine.hpp.

#### 7.12.4.6 m_objects

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

Definition at line 56 of file Engine.hpp.

#### 7.12.4.7 m_renderer

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

Definition at line 53 of file Engine.hpp.

Referenced by Engine().

#### 7.12.4.8 m_state

```
ENGINE_STATE ven::Engine::m_state {EXIT} [private]
```

Definition at line 49 of file Engine.hpp.

#### 7.12.4.9 m_surface

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

Definition at line 60 of file Engine.hpp.

Referenced by createSurface().

#### 7.12.4.10 m_window

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

Definition at line 51 of file Engine.hpp.

Referenced by createSurface(), and Engine().

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

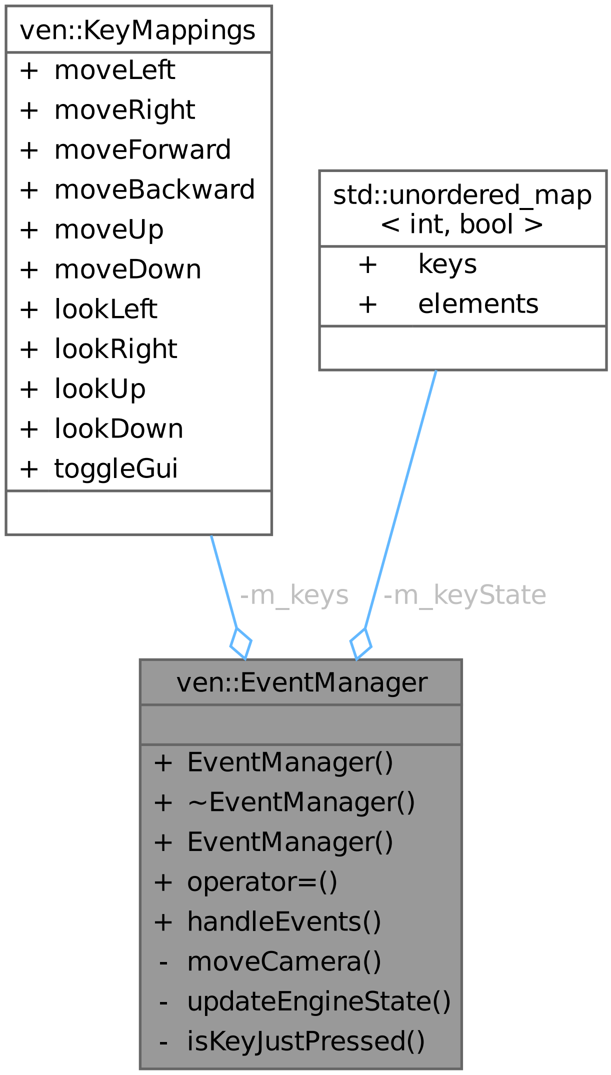
- /home/runner/work/VEngine/VEngine/include/VEngine/Engine.hpp
- /home/runner/work/VEngine/VEngine/src/engine.cpp

## 7.13 ven::EventManager Class Reference

Class for event manager.

`#include <EventManager.hpp>`

Collaboration diagram for ven::EventManager:




### Public Member Functions

- EventManager ()=default
- ∼EventManager ()=default
- EventManager (const EventManager &)=delete
- EventManager & operator= (const EventManager &)=delete
- void handleEvents (GLFWwindow ∗window, ENGINE_STATE ∗engineState, Camera &camera, Gui &gui, float dt) const

### Private Member Functions

- void moveCamera (GLFWwindow ∗window, Camera &camera, Gui &gui, float dt) const

**Static Private Member Functions**

- static void updateEngineState (ENGINE_STATE ∗engineState, const ENGINE_STATE newState)
- static bool isKeyJustPressed (GLFWwindow ∗window, int key, std::unordered_map< int, bool > &keyStates)

**Private Attributes**

- KeyMappings m_keys {}
- std::unordered_map< int, bool > m_keyState

## 7.13.1 Detailed Description

Class for event manager.

Definition at line 38 of file EventManager.hpp.

## 7.13.2 Constructor & Destructor Documentation

### 7.13.2.1 EventManager() [1/2]

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

### 7.13.2.2 ∼EventManager()

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

### 7.13.2.3 EventManager() [2/2]

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

## 7.13.3 Member Function Documentation

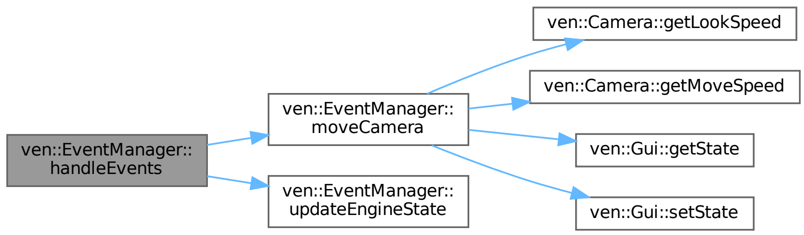
### 7.13.3.1 handleEvents()

```
void ven::EventManager::handleEvents (
            GLFWwindow * window,
            ENGINE_STATE * engineState,
            Camera & camera,
            Gui & gui,
            float dt) const
```

Definition at line 4 of file eventManager.cpp.

References ven::EXIT, moveCamera(), and updateEngineState().

Here is the call graph for this function:




#### 7.13.3.2 isKeyJustPressed()

```
bool ven::EventManager::isKeyJustPressed (
            GLFWwindow * window,
            int key,
            std::unordered_map< int, bool > & keyStates)  [static], [private]
```

Definition at line 12 of file eventManager.cpp.

#### 7.13.3.3 moveCamera()

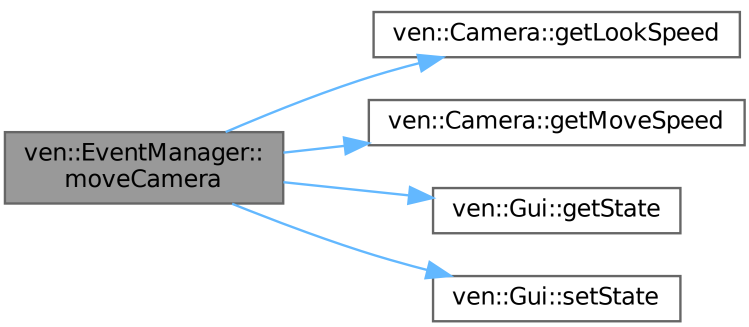
```
void ven::EventManager::moveCamera (
            GLFWwindow * window,
            Camera & camera,
            Gui & gui,
            float dt) const  [private]
```

Definition at line 22 of file eventManager.cpp.

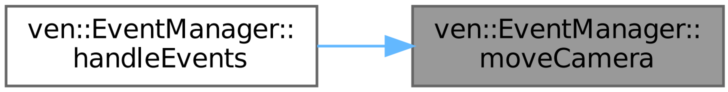
References ven::Camera::getLookSpeed(), ven::Camera::getMoveSpeed(), ven::Gui::getState(), ven::HIDDEN, ven::Transform3DComponent::rotation, ven::Gui::setState(), ven::Camera::transform3D, ven::Transform3DComponent::translation, and ven::VISIBLE.

Referenced by handleEvents().

Here is the call graph for this function:



Here is the caller graph for this function:




#### 7.13.3.4 operator=()

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

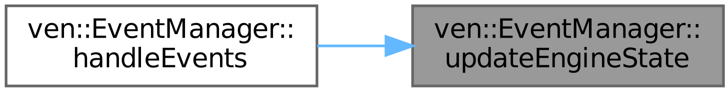
#### 7.13.3.5 updateEngineState()

```
static void ven::EventManager::updateEngineState (
            ENGINE_STATE * engineState,
            const ENGINE_STATE newState)  [inline], [static], [private]
```

Definition at line 53 of file EventManager.hpp.

Referenced by handleEvents().

Here is the caller graph for this function:




### 7.13.4 Member Data Documentation

#### 7.13.4.1 m_keys

```
KeyMappings ven::EventManager::m_keys {}  [private]
```

Definition at line 56 of file EventManager.hpp.

#### 7.13.4.2 m_keyState

```
std::unordered_map<int, bool> ven::EventManager::m_keyState  [mutable], [private]
```

Definition at line 57 of file EventManager.hpp.

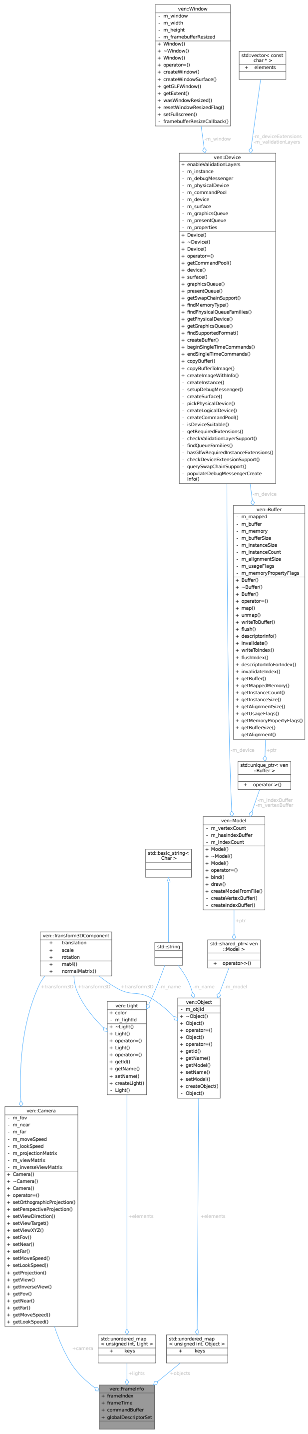
The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/EventManager.hpp
- /home/runner/work/VEngine/VEngine/src/eventManager.cpp

## 7.14 ven::FrameInfo Struct Reference

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

Collaboration diagram for ven::FrameInfo:



## Public Attributes

- unsigned long frameIndex
- float frameTime
- VkCommandBuffer commandBuffer
- Camera & camera
- VkDescriptorSet globalDescriptorSet
- Object::Map & objects
- Light::Map & lights

## 7.14.1 Detailed Description

Definition at line 38 of file FrameInfo.hpp.

## 7.14.2 Member Data Documentation

### 7.14.2.1 camera

`Camera& ven::FrameInfo::camera`

Definition at line 43 of file FrameInfo.hpp.

### 7.14.2.2 commandBuffer

`VkCommandBuffer ven::FrameInfo::commandBuffer`

Definition at line 42 of file FrameInfo.hpp.

Referenced by ven::ObjectRenderSystem::render(), and ven::PointLightRenderSystem::render().

### 7.14.2.3 frameIndex

`unsigned long ven::FrameInfo::frameIndex`

Definition at line 40 of file FrameInfo.hpp.

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

### 7.14.2.4 frameTime

`float ven::FrameInfo::frameTime`

Definition at line 41 of file FrameInfo.hpp.

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

### 7.14.2.5 globalDescriptorSet

`VkDescriptorSet ven::FrameInfo::globalDescriptorSet`

Definition at line 44 of file FrameInfo.hpp.

Referenced by ven::ObjectRenderSystem::render(), and ven::PointLightRenderSystem::render().

#### 7.14.2.6 lights

`Light::Map& ven::FrameInfo::lights`

Definition at line 46 of file FrameInfo.hpp.

Referenced by ven::PointLightRenderSystem::render(), and ven::PointLightRenderSystem::update().

#### 7.14.2.7 objects

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

Definition at line 45 of file FrameInfo.hpp.

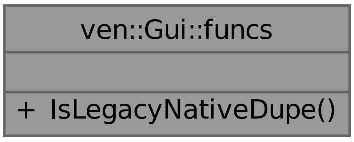
Referenced by ven::ObjectRenderSystem::render().

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

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

## 7.15 ven::Gui::funcs Struct Reference

Collaboration diagram for ven::Gui::funcs:




**Static Public Member Functions**

- static bool IsLegacyNativeDupe (const ImGuiKey key)

### 7.15.1 Detailed Description

Definition at line 58 of file Gui.hpp.

### 7.15.2 Member Function Documentation

#### 7.15.2.1 IsLegacyNativeDupe()

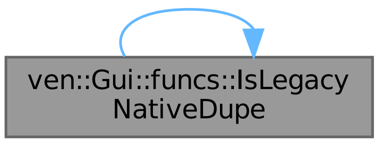
```
static bool ven::Gui::funcs::IsLegacyNativeDupe (
            const ImGuiKey key) [inline], [static]
```

Definition at line 58 of file Gui.hpp.

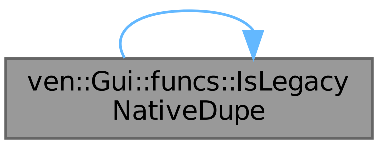
References IsLegacyNativeDupe().

Referenced by IsLegacyNativeDupe().

Here is the call graph for this function:




Here is the caller graph for this function:



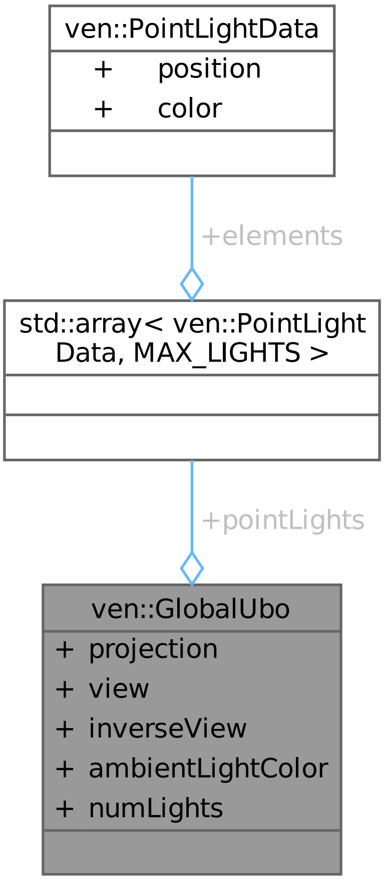

The documentation for this struct was generated from the following file:

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

## 7.16 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 {DEFAULT_AMBIENT_LIGHT_COLOR}
- std::array< PointLightData, MAX_LIGHTS > pointLights
- uint16_t numLights

### 7.16.1 Detailed Description

Definition at line 28 of file FrameInfo.hpp.

## 7.16.2 Member Data Documentation

### 7.16.2.1 ambientLightColor

```
glm::vec4 ven::GlobalUbo::ambientLightColor {DEFAULT_AMBIENT_LIGHT_COLOR}
```

Definition at line 33 of file FrameInfo.hpp.

Referenced by ven::Gui::rendererSection().

### 7.16.2.2 inverseView

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

Definition at line 32 of file FrameInfo.hpp.

### 7.16.2.3 numLights

```
uint16_t ven::GlobalUbo::numLights
```

Definition at line 35 of file FrameInfo.hpp.

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

### 7.16.2.4 pointLights

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

Definition at line 34 of file FrameInfo.hpp.

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

### 7.16.2.5 projection

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

Definition at line 30 of file FrameInfo.hpp.

### 7.16.2.6 view

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

Definition at line 31 of file FrameInfo.hpp.

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

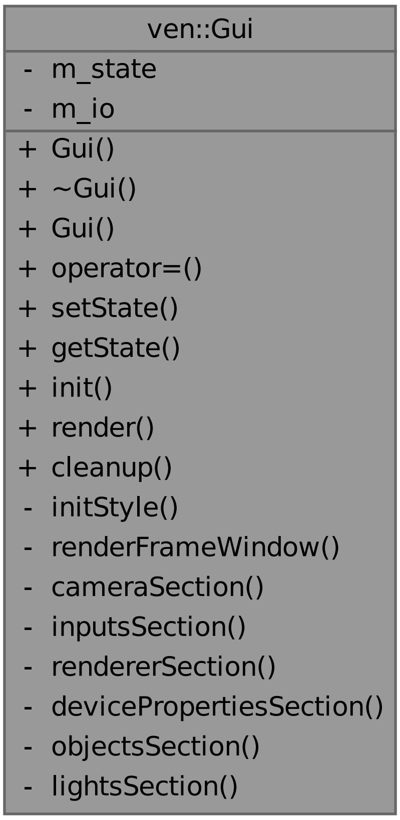
- /home/runner/work/VEngine/VEngine/include/VEngine/FrameInfo.hpp

## 7.17 ven::Gui Class Reference

Class for Gui.

```
#include <Gui.hpp>
```

Collaboration diagram for ven::Gui:




### Classes

- struct funcs

### Public Member Functions

- Gui ()=default
- ∼Gui ()=default
- Gui (const Gui &)=delete
- Gui & operator= (const Gui &)=delete
- void setState (const GUI_STATE state)
- GUI_STATE getState () const

### Static Public Member Functions

- static void init (GLFWwindow *window, VkInstance instance, const Device *device, VkRenderPass render↩ Pass)
- static void render (Renderer *renderer, std::unordered_map< unsigned int, Object > &objects, std↩ ::unordered_map< unsigned int, Light > &lights, Camera &camera, VkPhysicalDevice physicalDevice, GlobalUbo &ubo)
- static void cleanup ()

### Static Private Member Functions

- static void initStyle ()
- static void renderFrameWindow ()
- static void cameraSection (Camera &camera)
- static void inputsSection (const ImGuiIO *io)
- static void rendererSection (Renderer *renderer, GlobalUbo &ubo)
- static void devicePropertiesSection (VkPhysicalDeviceProperties deviceProperties)
- static void objectsSection (std::unordered_map< unsigned int, Object > &objects)
- static void lightsSection (std::unordered_map< unsigned int, Light > &lights)

### Private Attributes

- GUI_STATE m_state {VISIBLE}

### Static Private Attributes

- static ImGuiIO * m_io = nullptr

## 7.17.1 Detailed Description

Class for Gui.

Definition at line 28 of file Gui.hpp.

## 7.17.2 Constructor & Destructor Documentation

### 7.17.2.1 Gui() [1/2]

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

### 7.17.2.2 ~Gui()

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

#### 7.17.2.3 Gui() [2/2]

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

### 7.17.3 Member Function Documentation

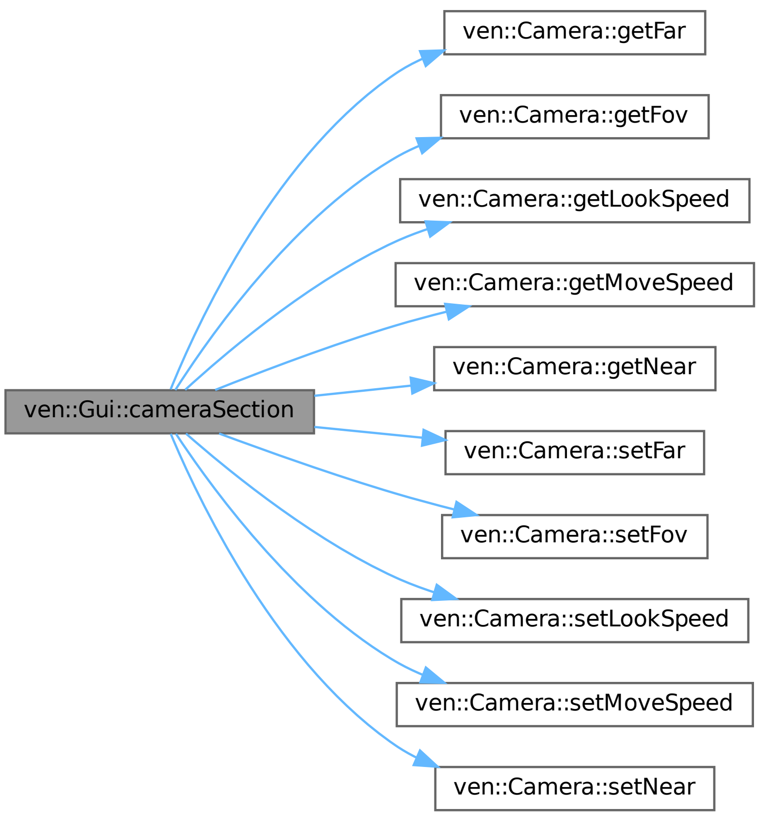
#### 7.17.3.1 cameraSection()

```
void ven::Gui::cameraSection (
            Camera & camera)  [static], [private]
```

Definition at line 110 of file render.cpp.

References ven::DEFAULT_FAR, ven::DEFAULT_FOV, ven::DEFAULT_LOOK_SPEED, ven::DEFAULT_MOVE_SPEED, ven::DEFAULT_NEAR, ven::DEFAULT_POSITION, ven::DEFAULT_ROTATION, ven::Camera::getFar(), ven::Camera::getFov(), ven::Camera::getLookSpeed(), ven::Camera::getMoveSpeed(), ven::Camera::getNear(), ven::Transform3DComponent::rotation, ven::Camera::setFar(), ven::Camera::setFov(), ven::Camera::setLookSpeed(), ven::Camera::setMoveSpeed(), ven::Camera::setNear(), ven::Camera::transform3D, and ven::Transform3DComponent::translation.

Here is the call graph for this function:



#### 7.17.3.2 cleanup()

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

Definition at line 9 of file render.cpp.

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

Here is the caller graph for this function:


#### 7.17.3.3 devicePropertiesSection()

```
void ven::Gui::devicePropertiesSection (
            VkPhysicalDeviceProperties deviceProperties)  [static], [private]
```

Definition at line 244 of file render.cpp.

#### 7.17.3.4 getState()

```
GUI_STATE ven::Gui::getState ( ) const  [inline], [nodiscard]
```

Definition at line 44 of file Gui.hpp.

References m_state.

Referenced by ven::EventManager::moveCamera().

Here is the caller graph for this function:

#### 7.17.3.5 init()

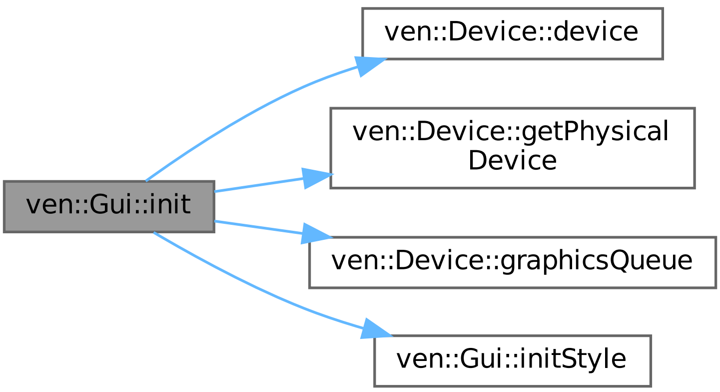
```
void ven::Gui::init (
            GLFWwindow * window,
            VkInstance instance,
            const Device * device,
            VkRenderPass renderPass)  [static]
```

Definition at line 11 of file init.cpp.

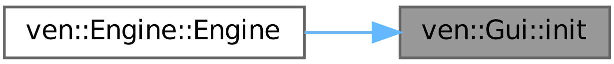
References DESCRIPTOR_COUNT, ven::Device::device(), ven::Device::getPhysicalDevice(), ven::Device::graphicsQueue(), initStyle(), and m_io.

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

Here is the call graph for this function:




Here is the caller graph for this function:



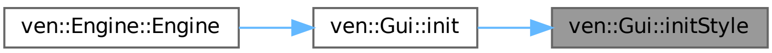
#### 7.17.3.6 initStyle()

```
void ven::Gui::initStyle ( ) [static], [private]
```

Definition at line 62 of file init.cpp.

Referenced by init().

Here is the caller graph for this function:




#### 7.17.3.7 inputsSection()

```
void ven::Gui::inputsSection (
            const ImGuiIO * io) [static], [private]
```

Definition at line 222 of file render.cpp.

#### 7.17.3.8 lightsSection()

```
void ven::Gui::lightsSection (
            std::unordered_map< unsigned int, Light > & lights) [static], [private]
```

Definition at line 178 of file render.cpp.

References ven::Colors::COLOR_PRESETS_3, and ven::DEFAULT_LIGHT_INTENSITY.

#### 7.17.3.9 objectsSection()

```
void ven::Gui::objectsSection (
            std::unordered_map< unsigned int, Object > & objects) [static], [private]
```

Definition at line 159 of file render.cpp.

References ven::Colors::GRAY_4.

#### 7.17.3.10 operator=()

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

#### 7.17.3.11 render()

```
void ven::Gui::render (
            Renderer * renderer,
            std::unordered_map< unsigned int, Object > & objects,
            std::unordered_map< unsigned int, Light > & lights,
            Camera & camera,
            VkPhysicalDevice physicalDevice,
            GlobalUbo & ubo) [static]
```

Definition at line 16 of file render.cpp.

References ven::Renderer::getCurrentCommandBuffer().

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

Here is the call graph for this function:


Here is the caller graph for this function:

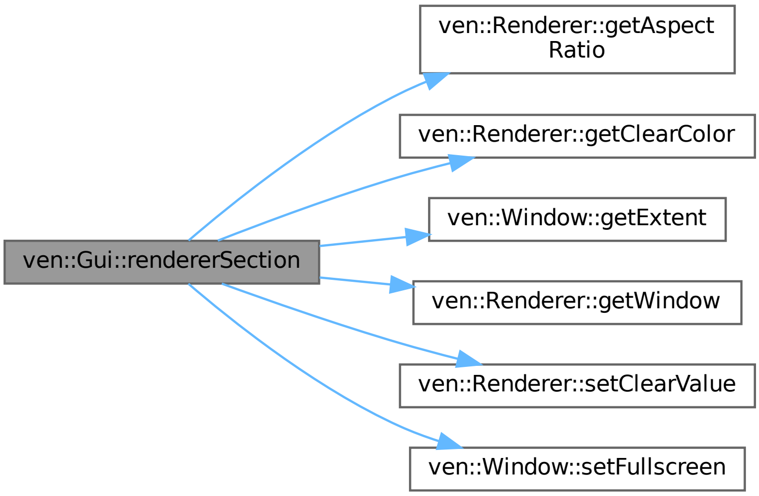

#### 7.17.3.12 rendererSection()

```
void ven::Gui::rendererSection (
            Renderer * renderer,
            GlobalUbo & ubo) [static], [private]
```

Definition at line 51 of file render.cpp.

References ven::GlobalUbo::ambientLightColor, ven::Colors::COLOR_PRESETS_4, ven::Colors::COLOR_PRESETS_VK, ven::DEFAULT_AMBIENT_LIGHT_INTENSITY, ven::Renderer::getAspectRatio(), ven::Renderer::getClearColor(), ven::Window::getExtent(), ven::Renderer::getWindow(), ven::Renderer::setClearValue(), and ven::Window::setFullscreen().

Here is the call graph for this function:




#### 7.17.3.13 renderFrameWindow()

`void ven::Gui::renderFrameWindow () [static], [private]`

Definition at line 40 of file render.cpp.

#### 7.17.3.14 setState()

```
void ven::Gui::setState (
            const GUI_STATE state) [inline]
```

Definition at line 43 of file Gui.hpp.

References m_state.

Referenced by ven::EventManager::moveCamera().

Here is the caller graph for this function:

### 7.17.4 Member Data Documentation

#### 7.17.4.1 m_io

```
ImGuiIO * ven::Gui::m_io = nullptr  [static], [private]
```

Definition at line 60 of file Gui.hpp.

Referenced by init().

#### 7.17.4.2 m_state

```
GUI_STATE ven::Gui::m_state {VISIBLE}  [private]
```

Definition at line 61 of file Gui.hpp.

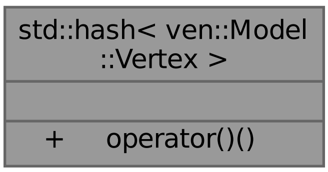
Referenced by getState(), and setState().

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

- /home/runner/work/VEngine/VEngine/include/VEngine/Gui.hpp
- /home/runner/work/VEngine/VEngine/src/gui/init.cpp
- /home/runner/work/VEngine/VEngine/src/gui/render.cpp

## 7.18 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 noexcept

### 7.18.1 Detailed Description

Definition at line 17 of file model.cpp.

## 7.18.2 Member Function Documentation

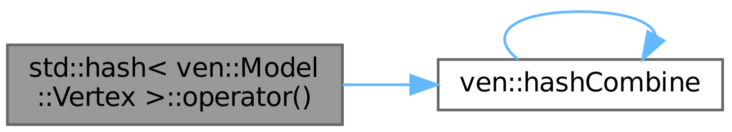
### 7.18.2.1 operator()()

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

Definition at line 18 of file model.cpp.

References ven::hashCombine().

Here is the call graph for this function:



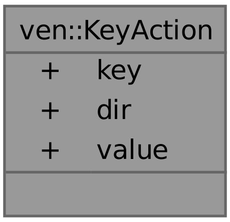

The documentation for this struct was generated from the following file:

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

## 7.19 ven::KeyAction Struct Reference

```
#include <EventManager.hpp>
```

Collaboration diagram for ven::KeyAction:



**Public Attributes**

- uint16_t key
- glm::vec3 * dir
- glm::vec3 value

## 7.19.1 Detailed Description

Definition at line 13 of file EventManager.hpp.

## 7.19.2 Member Data Documentation

### 7.19.2.1 dir

```
glm::vec3* ven::KeyAction::dir
```

Definition at line 15 of file EventManager.hpp.

### 7.19.2.2 key

```
uint16_t ven::KeyAction::key
```

Definition at line 14 of file EventManager.hpp.

### 7.19.2.3 value

```
glm::vec3 ven::KeyAction::value
```

Definition at line 16 of file EventManager.hpp.

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

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

## 7.20 ven::KeyMappings Struct Reference

```
#include <EventManager.hpp>
```

Collaboration diagram for ven::KeyMappings:


**Public Attributes**

- uint16_t moveLeft = GLFW_KEY_A
- uint16_t moveRight = GLFW_KEY_D
- uint16_t moveForward = GLFW_KEY_W
- uint16_t moveBackward = GLFW_KEY_S
- uint16_t moveUp = GLFW_KEY_SPACE
- uint16_t moveDown = GLFW_KEY_LEFT_SHIFT
- uint16_t lookLeft = GLFW_KEY_LEFT
- uint16_t lookRight = GLFW_KEY_RIGHT
- uint16_t lookUp = GLFW_KEY_UP
- uint16_t lookDown = GLFW_KEY_DOWN
- uint16_t toggleGui = GLFW_KEY_F1

### 7.20.1 Detailed Description

Definition at line 19 of file EventManager.hpp.

## 7.20.2 Member Data Documentation

### 7.20.2.1 lookDown

```
uint16_t ven::KeyMappings::lookDown = GLFW_KEY_DOWN
```

Definition at line 29 of file EventManager.hpp.

### 7.20.2.2 lookLeft

```
uint16_t ven::KeyMappings::lookLeft = GLFW_KEY_LEFT
```

Definition at line 26 of file EventManager.hpp.

### 7.20.2.3 lookRight

```
uint16_t ven::KeyMappings::lookRight = GLFW_KEY_RIGHT
```

Definition at line 27 of file EventManager.hpp.

### 7.20.2.4 lookUp

```
uint16_t ven::KeyMappings::lookUp = GLFW_KEY_UP
```

Definition at line 28 of file EventManager.hpp.

### 7.20.2.5 moveBackward

```
uint16_t ven::KeyMappings::moveBackward = GLFW_KEY_S
```

Definition at line 23 of file EventManager.hpp.

### 7.20.2.6 moveDown

```
uint16_t ven::KeyMappings::moveDown = GLFW_KEY_LEFT_SHIFT
```

Definition at line 25 of file EventManager.hpp.

### 7.20.2.7 moveForward

```
uint16_t ven::KeyMappings::moveForward = GLFW_KEY_W
```

Definition at line 22 of file EventManager.hpp.

#### 7.20.2.8 moveLeft

```
uint16_t ven::KeyMappings::moveLeft = GLFW_KEY_A
```

Definition at line 20 of file EventManager.hpp.

#### 7.20.2.9 moveRight

```
uint16_t ven::KeyMappings::moveRight = GLFW_KEY_D
```

Definition at line 21 of file EventManager.hpp.

#### 7.20.2.10 moveUp

```
uint16_t ven::KeyMappings::moveUp = GLFW_KEY_SPACE
```

Definition at line 24 of file EventManager.hpp.

#### 7.20.2.11 toggleGui

```
uint16_t ven::KeyMappings::toggleGui = GLFW_KEY_F1
```

Definition at line 30 of file EventManager.hpp.

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

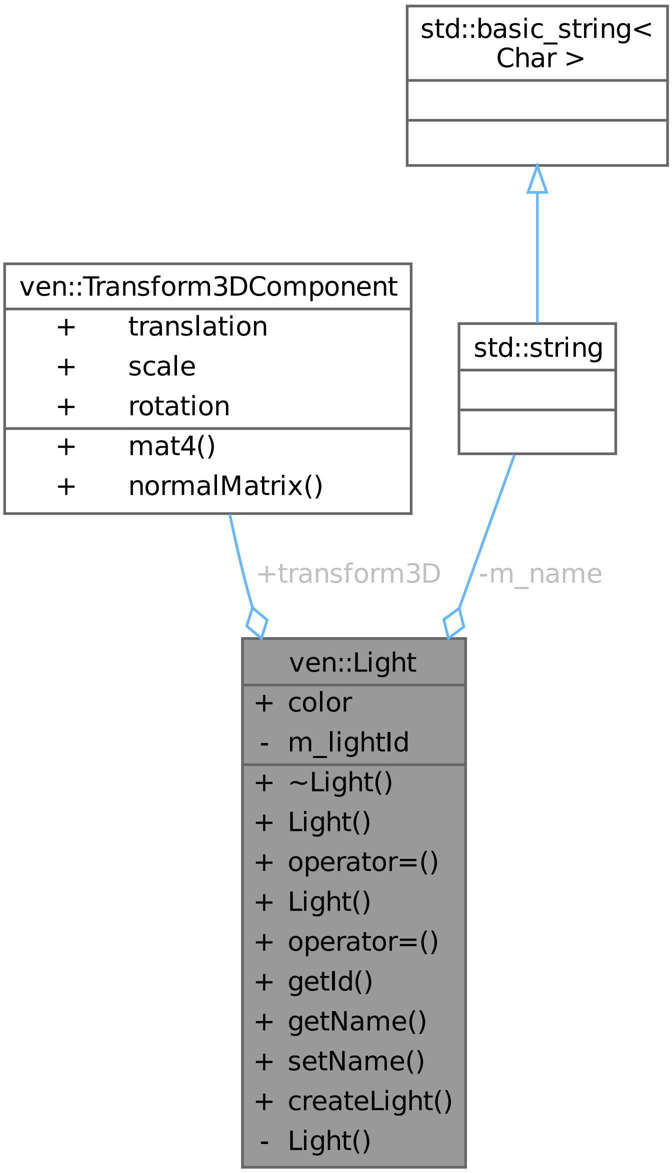
- /home/runner/work/VEngine/VEngine/include/VEngine/EventManager.hpp

## 7.21 ven::Light Class Reference

Class for light.

```
#include <Light.hpp>
```

Collaboration diagram for ven::Light:




## Public Types

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

## Public Member Functions

- ~Light ()=default
- Light (const Light &)=delete
- Light & operator= (const Light &)=delete

- Light (Light &&)=default
- Light & operator= (Light &&)=default
- unsigned int getId () const
- std::string getName () const
- void setName (const std::string &name)

### Static Public Member Functions

- static Light createLight (float radius=DEFAULT_LIGHT_RADIUS, glm::vec4 color=DEFAULT_LIGHT_COLOR)

### Public Attributes

- glm::vec4 color {DEFAULT_LIGHT_COLOR}
- Transform3DComponent transform3D {}

### Private Member Functions

- Light (const unsigned int lightId)

### Private Attributes

- unsigned int m_lightId
- std::string m_name {"point light"}

## 7.21.1 Detailed Description

Class for light.

Definition at line 27 of file Light.hpp.

## 7.21.2 Member Typedef Documentation

### 7.21.2.1 Map

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

Definition at line 31 of file Light.hpp.

## 7.21.3 Constructor & Destructor Documentation

### 7.21.3.1 ∼Light()

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

#### 7.21.3.2 Light() [1/3]

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

#### 7.21.3.3 Light() [2/3]

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

#### 7.21.3.4 Light() [3/3]

```
ven::Light::Light (
            const unsigned int lightId)  [inline], [explicit], [private]
```

Definition at line 52 of file Light.hpp.

### 7.21.4 Member Function Documentation

#### 7.21.4.1 createLight()

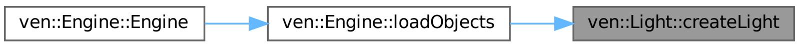
```
ven::Light ven::Light::createLight (
            float radius = DEFAULT_LIGHT_RADIUS,
            glm::vec4 color = DEFAULT_LIGHT_COLOR)  [static]
```

Definition at line 3 of file light.cpp.

References color, ven::Transform3DComponent::scale, and transform3D.

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

Here is the caller graph for this function:




#### 7.21.4.2 getId()

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

Definition at line 45 of file Light.hpp.

References m_lightId.

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

Here is the caller graph for this function:

#### 7.21.4.3 getName()

```
std::string ven::Light::getName () const  [inline], [nodiscard]
```

Definition at line 46 of file Light.hpp.

References m_name.

#### 7.21.4.4 operator=() [1/2]

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

#### 7.21.4.5 operator=() [2/2]

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

#### 7.21.4.6 setName()

```
void ven::Light::setName (
            const std::string & name)  [inline]
```

Definition at line 48 of file Light.hpp.

References m_name.

### 7.21.5 Member Data Documentation

#### 7.21.5.1 color

```
glm::vec4 ven::Light::color {DEFAULT_LIGHT_COLOR}
```

Definition at line 42 of file Light.hpp.

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

#### 7.21.5.2 m_lightId

```
unsigned int ven::Light::m_lightId  [private]
```

Definition at line 54 of file Light.hpp.

Referenced by getId().

#### 7.21.5.3 m_name

```
std::string ven::Light::m_name {"point light"} [private]
```

Definition at line 55 of file Light.hpp.

Referenced by getName(), and setName().

#### 7.21.5.4 transform3D

```
Transform3DComponent ven::Light::transform3D {}
```

Definition at line 43 of file Light.hpp.

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

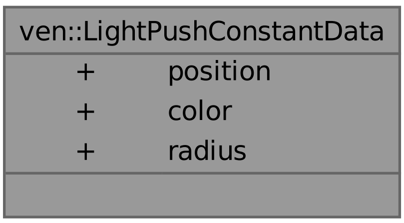
The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/Light.hpp
- /home/runner/work/VEngine/VEngine/src/light.cpp

## 7.22 ven::LightPushConstantData Struct Reference

```
#include <PointLightRenderSystem.hpp>
```

Collaboration diagram for ven::LightPushConstantData:




### Public Attributes

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

### 7.22.1 Detailed Description

Definition at line 14 of file PointLightRenderSystem.hpp.

### 7.22.2 Member Data Documentation

#### 7.22.2.1 color

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

Definition at line 16 of file PointLightRenderSystem.hpp.

#### 7.22.2.2 position

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

Definition at line 15 of file PointLightRenderSystem.hpp.

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

#### 7.22.2.3 radius

```
float ven::LightPushConstantData::radius
```

Definition at line 17 of file PointLightRenderSystem.hpp.

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

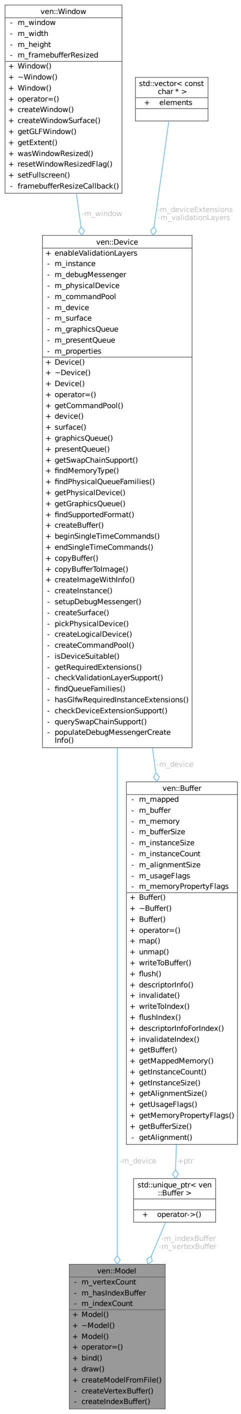
- /home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/PointLightRenderSystem.hpp

## 7.23 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 ()=default
- 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

## 7.23.1 Detailed Description

Class for model.

Definition at line 25 of file Model.hpp.

## 7.23.2 Constructor & Destructor Documentation

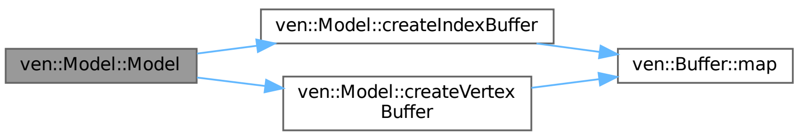
### 7.23.2.1 Model() [1/2]

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

Definition at line 25 of file model.cpp.

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

Here is the call graph for this function:



#### 7.23.2.2 ∼Model()

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

#### 7.23.2.3 Model() [2/2]

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

### 7.23.3 Member Function Documentation

#### 7.23.3.1 bind()

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

Definition at line 78 of file model.cpp.

#### 7.23.3.2 createIndexBuffer()

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

Definition at line 48 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:

#### 7.23.3.3 createModelFromFile()

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

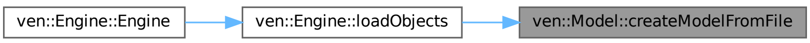
Definition at line 89 of file model.cpp.

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:




#### 7.23.3.4 createVertexBuffer()

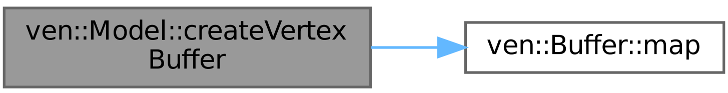
```
void ven::Model::createVertexBuffer (
            const std::vector< Vertex > & vertices)  [private]
```

Definition at line 31 of file model.cpp.

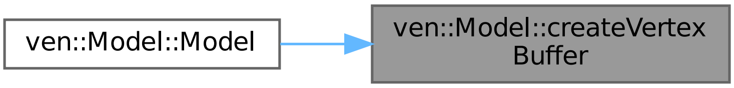
References ven::Buffer::map().

Referenced by Model().

Here is the call graph for this function:



Here is the caller graph for this function:




#### 7.23.3.5 draw()

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

Definition at line 69 of file model.cpp.

#### 7.23.3.6 operator=()

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

### 7.23.4 Member Data Documentation

#### 7.23.4.1 m_device

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

Definition at line 68 of file Model.hpp.

#### 7.23.4.2 m_hasIndexBuffer

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

Definition at line 72 of file Model.hpp.

#### 7.23.4.3 m_indexBuffer

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

Definition at line 73 of file Model.hpp.

#### 7.23.4.4 m_indexCount

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

Definition at line 74 of file Model.hpp.

#### 7.23.4.5 m_vertexBuffer

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

Definition at line 69 of file Model.hpp.

#### 7.23.4.6 m_vertexCount

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

Definition at line 70 of file Model.hpp.

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

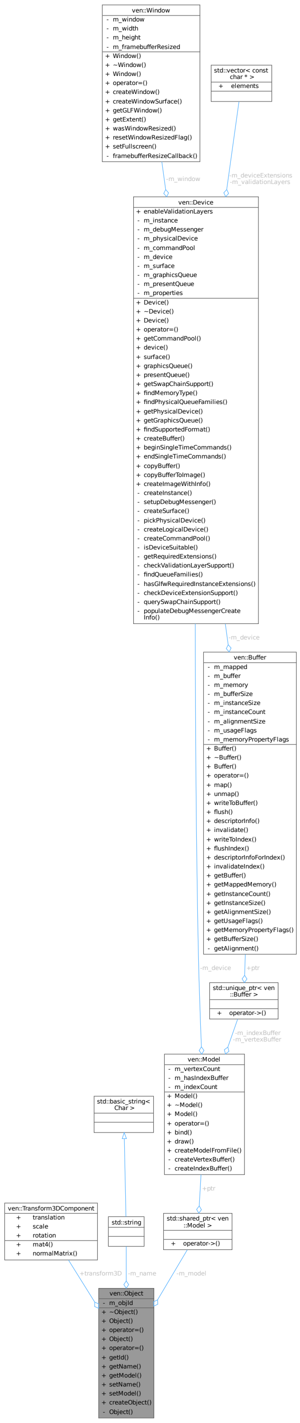
- /home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp
- /home/runner/work/VEngine/VEngine/src/model.cpp

## 7.24 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
- std::string getName () const
- std::shared_ptr< Model > getModel () const
- void setName (const std::string &name)
- void setModel (const std::shared_ptr< Model > &model)

#### Static Public Member Functions

- static Object createObject ()

#### Public Attributes

- Transform3DComponent transform3D {}

#### Private Member Functions

- Object (const unsigned int objId)

#### Private Attributes

- unsigned int m_objId
- std::string m_name
- std::shared_ptr< Model > m_model

## 7.24.1 Detailed Description

Class for object.

Definition at line 24 of file Object.hpp.

## 7.24.2 Member Typedef Documentation

### 7.24.2.1 Map

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

Definition at line 28 of file Object.hpp.

## 7.24.3 Constructor & Destructor Documentation

### 7.24.3.1 ~Object()

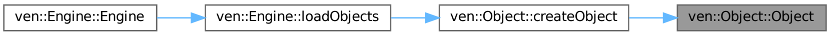
```
ven::Object::~Object () [default]
```

#### 7.24.3.2 Object() [1/3]

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

Referenced by createObject().

Here is the caller graph for this function:




#### 7.24.3.3 Object() [2/3]

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

#### 7.24.3.4 Object() [3/3]

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

Definition at line 50 of file Object.hpp.

### 7.24.4 Member Function Documentation

#### 7.24.4.1 createObject()

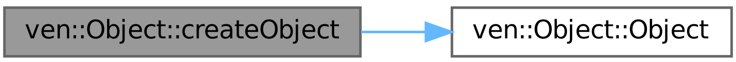
```
static Object ven::Object::createObject ()  [inline], [static]
```

Definition at line 37 of file Object.hpp.

References Object().

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

Here is the call graph for this function:




Here is the caller graph for this function:

#### 7.24.4.2 getId()

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

Definition at line 39 of file Object.hpp.

References m_objId.

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

Here is the caller graph for this function:


#### 7.24.4.3 getModel()

`std::shared_ptr< Model > ven::Object::getModel () const [inline], [nodiscard]`

Definition at line 41 of file Object.hpp.

References m_model.

#### 7.24.4.4 getName()

`std::string ven::Object::getName () const [inline], [nodiscard]`

Definition at line 40 of file Object.hpp.

References m_name.

#### 7.24.4.5 operator=() [1/2]

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

#### 7.24.4.6 operator=() [2/2]

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

#### 7.24.4.7 setModel()

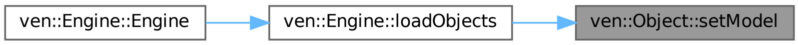
```
void ven::Object::setModel (
            const std::shared_ptr< Model > & model)  [inline]
```

Definition at line 44 of file Object.hpp.

References m_model.

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

Here is the caller graph for this function:




#### 7.24.4.8 setName()

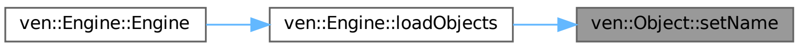
```
void ven::Object::setName (
            const std::string & name)  [inline]
```

Definition at line 43 of file Object.hpp.

References m_name.

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

Here is the caller graph for this function:




### 7.24.5 Member Data Documentation

#### 7.24.5.1 m_model

```
std::shared_ptr<Model> ven::Object::m_model  [private]
```

Definition at line 54 of file Object.hpp.

Referenced by getModel(), and setModel().

#### 7.24.5.2 m_name

```
std::string ven::Object::m_name [private]
```

Definition at line 53 of file Object.hpp.

Referenced by getName(), and setName().

#### 7.24.5.3 m_objId

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

Definition at line 52 of file Object.hpp.

Referenced by getId().

#### 7.24.5.4 transform3D

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

Definition at line 46 of file Object.hpp.

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

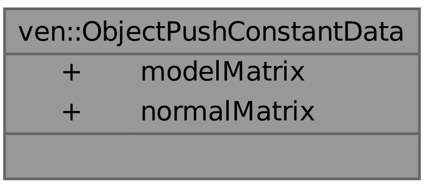
The documentation for this class was generated from the following file:

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

## 7.25 ven::ObjectPushConstantData Struct Reference

```
#include <ObjectRenderSystem.hpp>
```

Collaboration diagram for ven::ObjectPushConstantData:



**Public Attributes**

- glm::mat4 modelMatrix {}
- glm::mat4 normalMatrix {}

### 7.25.1 Detailed Description

Definition at line 14 of file ObjectRenderSystem.hpp.

### 7.25.2 Member Data Documentation

#### 7.25.2.1 modelMatrix

```
glm::mat4 ven::ObjectPushConstantData::modelMatrix {}
```

Definition at line 15 of file ObjectRenderSystem.hpp.

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

#### 7.25.2.2 normalMatrix

```
glm::mat4 ven::ObjectPushConstantData::normalMatrix {}
```

Definition at line 16 of file ObjectRenderSystem.hpp.

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

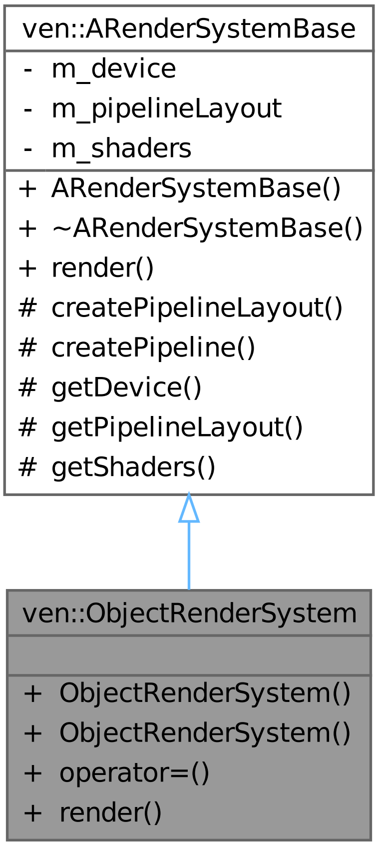
- /home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/ObjectRenderSystem.hpp

## 7.26 ven::ObjectRenderSystem Class Reference

Class for object render system.

```
#include <ObjectRenderSystem.hpp>
```

Inheritance diagram for ven::ObjectRenderSystem:



Collaboration diagram for ven::ObjectRenderSystem:


## Public Member Functions

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

## Public Member Functions inherited from ven::ARenderSystemBase

- ARenderSystemBase (Device &device)
- virtual ∼ARenderSystemBase ()

#### Additional Inherited Members

## Protected Member Functions inherited from ven::ARenderSystemBase

- void createPipelineLayout (VkDescriptorSetLayout globalSetLayout, uint32_t pushConstantSize)
- void createPipeline (VkRenderPass renderPass, const std::string &shadersVertPath, const std::string &shadersFragPath, bool isLight)
- Device & getDevice () const
- VkPipelineLayout getPipelineLayout () const
- const std::unique_ptr< Shaders > & getShaders () const

## 7.26.1 Detailed Description

Class for object render system.

Definition at line 24 of file ObjectRenderSystem.hpp.

## 7.26.2 Constructor & Destructor Documentation

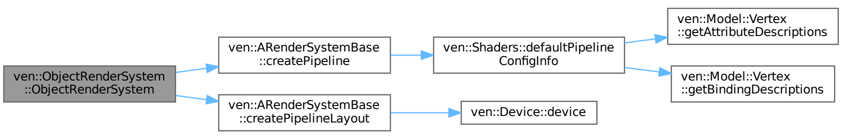
### 7.26.2.1 ObjectRenderSystem() [1/2]

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

Definition at line 28 of file ObjectRenderSystem.hpp.

References ven::ARenderSystemBase::createPipeline(), ven::ARenderSystemBase::createPipelineLayout(), and ven::SHADERS_BIN_PATH.

Here is the call graph for this function:




### 7.26.2.2 ObjectRenderSystem() [2/2]

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

## 7.26.3 Member Function Documentation

### 7.26.3.1 operator=()

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

### 7.26.3.2 render()

```
void ven::ObjectRenderSystem::render (
            const FrameInfo & frameInfo) const  [override], [virtual]
```

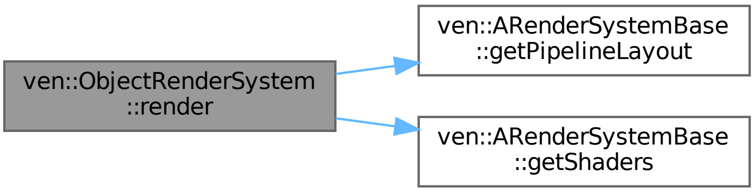
Implements ven::ARenderSystemBase.

Definition at line 5 of file objectRenderSystem.cpp.

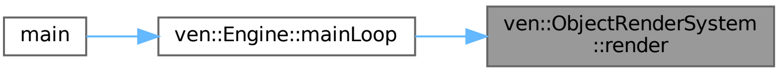
References ven::FrameInfo::commandBuffer, ven::ARenderSystemBase::getPipelineLayout(), ven::ARenderSystemBase::getShaders
ven::FrameInfo::globalDescriptorSet, ven::ObjectPushConstantData::modelMatrix, and ven::FrameInfo::objects.

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

Here is the call graph for this function:




Here is the caller graph for this function:



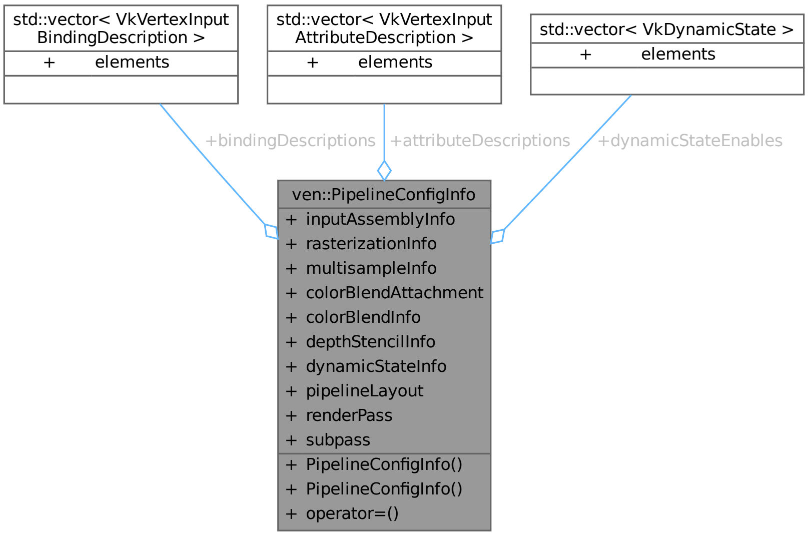

The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/ObjectRenderSystem.hpp
- /home/runner/work/VEngine/VEngine/src/system/objectRenderSystem.cpp

## 7.27 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

### 7.27.1 Detailed Description

Definition at line 19 of file Shaders.hpp.

## 7.27.2 Constructor & Destructor Documentation

### 7.27.2.1 PipelineConfigInfo() [1/2]

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

### 7.27.2.2 PipelineConfigInfo() [2/2]

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

## 7.27.3 Member Function Documentation

### 7.27.3.1 operator=()

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

## 7.27.4 Member Data Documentation

### 7.27.4.1 attributeDescriptions

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

Definition at line 25 of file Shaders.hpp.

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

### 7.27.4.2 bindingDescriptions

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

Definition at line 24 of file Shaders.hpp.

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

### 7.27.4.3 colorBlendAttachment

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

Definition at line 29 of file Shaders.hpp.

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

#### 7.27.4.4 colorBlendInfo

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

Definition at line 30 of file Shaders.hpp.

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

#### 7.27.4.5 depthStencilInfo

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

Definition at line 31 of file Shaders.hpp.

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

#### 7.27.4.6 dynamicStateEnables

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

Definition at line 32 of file Shaders.hpp.

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

#### 7.27.4.7 dynamicStateInfo

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

Definition at line 33 of file Shaders.hpp.

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

#### 7.27.4.8 inputAssemblyInfo

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

Definition at line 26 of file Shaders.hpp.

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

#### 7.27.4.9 multisampleInfo

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

Definition at line 28 of file Shaders.hpp.

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

#### 7.27.4.10 pipelineLayout

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

Definition at line 34 of file Shaders.hpp.

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

#### 7.27.4.11 rasterizationInfo

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

Definition at line 27 of file Shaders.hpp.

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

#### 7.27.4.12 renderPass

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

Definition at line 35 of file Shaders.hpp.

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

#### 7.27.4.13 subpass

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

Definition at line 36 of file Shaders.hpp.

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

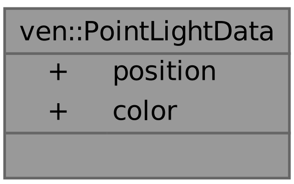
The documentation for this struct was generated from the following file:

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

## 7.28 ven::PointLightData Struct Reference

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

Collaboration diagram for ven::PointLightData:



**Public Attributes**

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

## 7.28.1 Detailed Description

Definition at line 22 of file FrameInfo.hpp.

## 7.28.2 Member Data Documentation

### 7.28.2.1 color

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

Definition at line 25 of file FrameInfo.hpp.

### 7.28.2.2 position

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

Definition at line 24 of file FrameInfo.hpp.

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

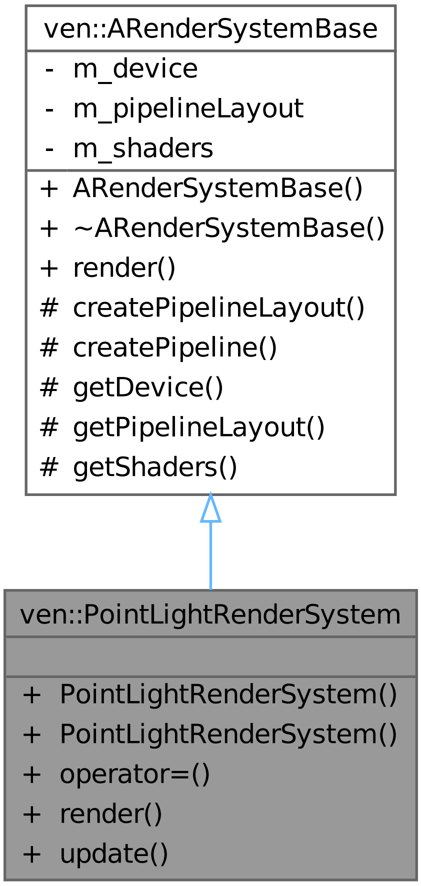
- /home/runner/work/VEngine/VEngine/include/VEngine/FrameInfo.hpp

# 7.29 ven::PointLightRenderSystem Class Reference

Class for point light system.

```
#include <PointLightRenderSystem.hpp>
```

Inheritance diagram for ven::PointLightRenderSystem:



Collaboration diagram for ven::PointLightRenderSystem:


### Public Member Functions

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

## Public Member Functions inherited from ven::ARenderSystemBase

- ARenderSystemBase (Device &device)
- virtual ∼ARenderSystemBase ()

### Static Public Member Functions

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

### Additional Inherited Members

## Protected Member Functions inherited from ven::ARenderSystemBase

- void createPipelineLayout (VkDescriptorSetLayout globalSetLayout, uint32_t pushConstantSize)
- void createPipeline (VkRenderPass renderPass, const std::string &shadersVertPath, const std::string &shadersFragPath, bool isLight)
- Device & getDevice () const
- VkPipelineLayout getPipelineLayout () const
- const std::unique_ptr< Shaders > & getShaders () const

## 7.29.1 Detailed Description

Class for point light system.

Definition at line 25 of file PointLightRenderSystem.hpp.

## 7.29.2 Constructor & Destructor Documentation

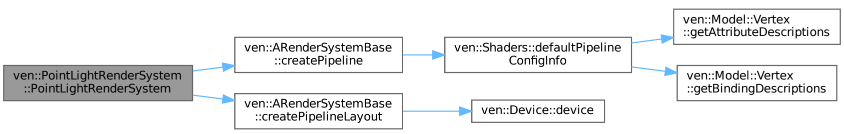
### 7.29.2.1 PointLightRenderSystem() [1/2]

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

Definition at line 29 of file PointLightRenderSystem.hpp.

References ven::ARenderSystemBase::createPipeline(), ven::ARenderSystemBase::createPipelineLayout(), and ven::SHADERS_BIN_PATH.

Here is the call graph for this function:



#### 7.29.2.2 PointLightRenderSystem() [2/2]

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

### 7.29.3 Member Function Documentation

#### 7.29.3.1 operator=()

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

#### 7.29.3.2 render()

```
void ven::PointLightRenderSystem::render (
            const FrameInfo & frameInfo) const  [override], [virtual]
```

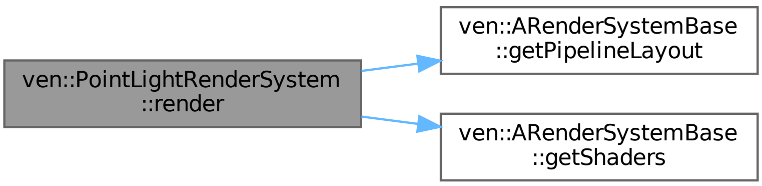
Implements ven::ARenderSystemBase.

Definition at line 5 of file pointLightRenderSystem.cpp.

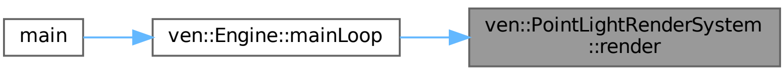
References ven::FrameInfo::commandBuffer, ven::ARenderSystemBase::getPipelineLayout(), ven::ARenderSystemBase::getShaders ven::FrameInfo::globalDescriptorSet, ven::FrameInfo::lights, and ven::LightPushConstantData::position.

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

Here is the call graph for this function:




Here is the caller graph for this function:



#### 7.29.3.3 update()

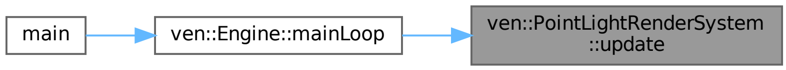
```
void ven::PointLightRenderSystem::update (
            const FrameInfo & frameInfo,
            GlobalUbo & ubo)  [static]
```

Definition at line 21 of file pointLightRenderSystem.cpp.

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

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

Here is the caller graph for this function:



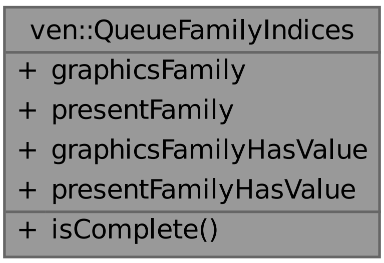

The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/PointLightRenderSystem.hpp
- /home/runner/work/VEngine/VEngine/src/system/pointLightRenderSystem.cpp

## 7.30 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

## 7.30.1 Detailed Description

Definition at line 21 of file Device.hpp.

## 7.30.2 Member Function Documentation

### 7.30.2.1 isComplete()

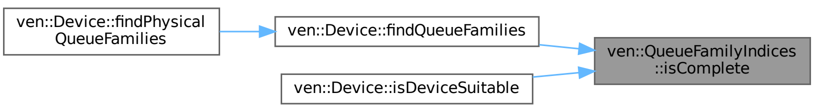
```
bool ven::QueueFamilyIndices::isComplete () const  [inline], [nodiscard]
```

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:




## 7.30.3 Member Data Documentation

### 7.30.3.1 graphicsFamily

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

Definition at line 22 of file Device.hpp.

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

### 7.30.3.2 graphicsFamilyHasValue

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

Definition at line 24 of file Device.hpp.

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

#### 7.30.3.3 presentFamily

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

Definition at line 23 of file Device.hpp.

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

#### 7.30.3.4 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:

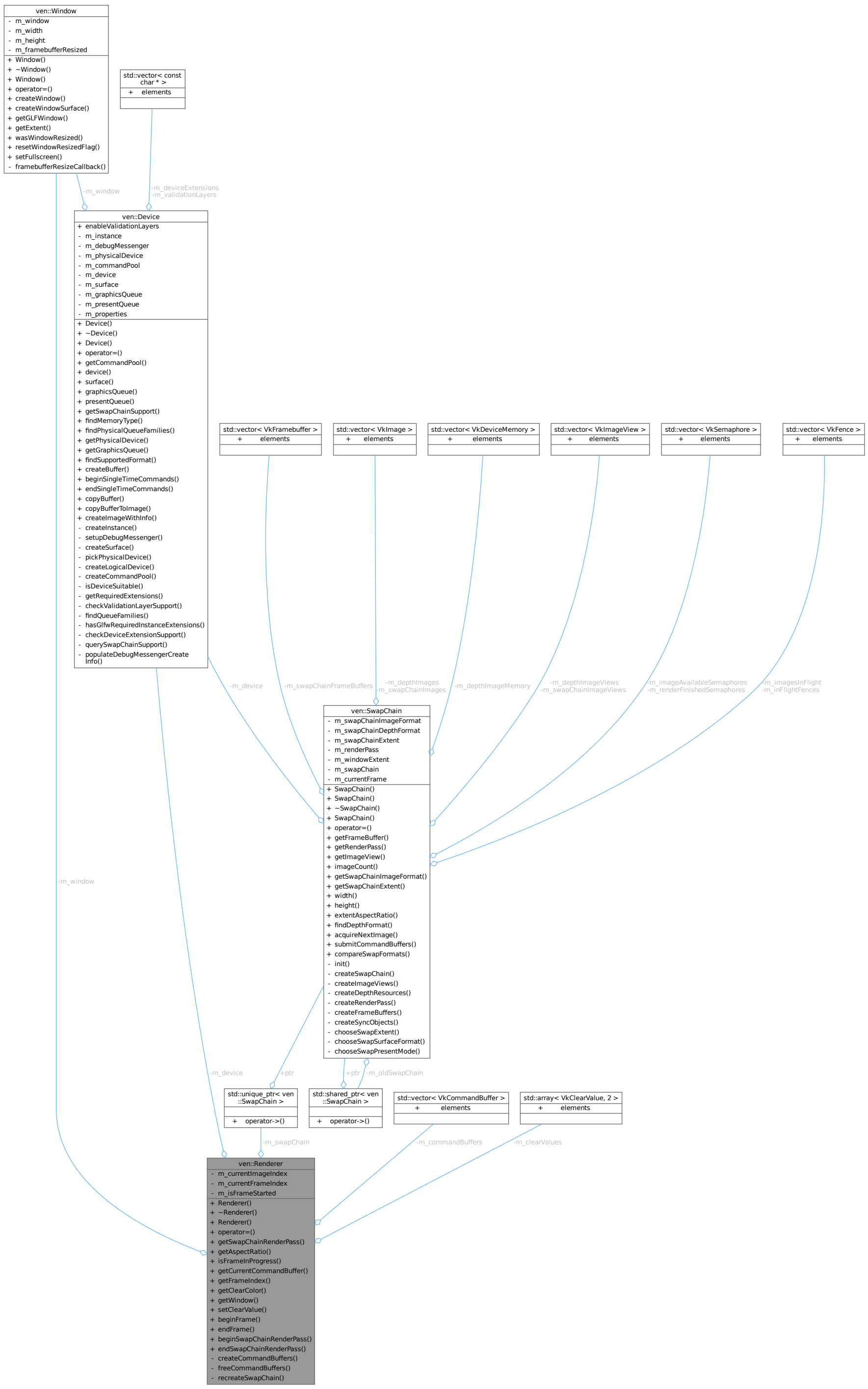
- /home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp

## 7.31 ven::Renderer Class Reference

Class for renderer.

```
#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
- unsigned long getFrameIndex () const
- std::array< float, 4 > getClearColor () const
- Window & getWindow () const
- void setClearValue (const VkClearColorValue clearColorValue=DEFAULT_CLEAR_COLOR, const VkClear←DepthStencilValue clearDepthValue=DEFAULT_CLEAR_DEPTH)
- VkCommandBuffer beginFrame ()
- void endFrame ()
- void beginSwapChainRenderPass (VkCommandBuffer commandBuffer) const
- void endSwapChainRenderPass (VkCommandBuffer commandBuffer) const

**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 {DEFAULT_CLEAR_COLOR, 1.0F, 0.F}
- uint32_t m_currentImageIndex {0}
- unsigned long m_currentFrameIndex {0}
- bool m_isFrameStarted {false}

## 7.31.1 Detailed Description

Class for renderer.

Definition at line 28 of file Renderer.hpp.

## 7.31.2 Constructor & Destructor Documentation

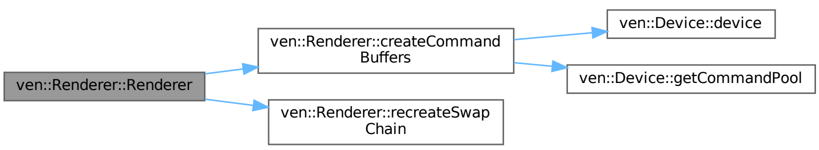
### 7.31.2.1 Renderer() [1/2]

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

Definition at line 32 of file Renderer.hpp.

References createCommandBuffers(), and recreateSwapChain().

Here is the call graph for this function:



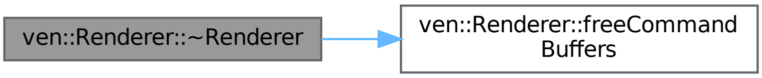
#### 7.31.2.2 ∼Renderer()

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

Definition at line 33 of file Renderer.hpp.

References freeCommandBuffers().

Here is the call graph for this function:




#### 7.31.2.3 Renderer() [2/2]

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

### 7.31.3 Member Function Documentation

#### 7.31.3.1 beginFrame()

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

Definition at line 43 of file renderer.cpp.

#### 7.31.3.2 beginSwapChainRenderPass()

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

Definition at line 89 of file renderer.cpp.

#### 7.31.3.3 createCommandBuffers()

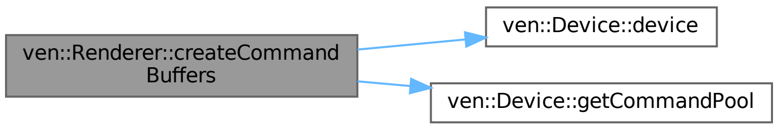
```
void ven::Renderer::createCommandBuffers ()  [private]
```

Definition at line 3 of file renderer.cpp.

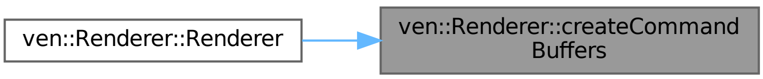
References ven::Device::device(), ven::Device::getCommandPool(), m_commandBuffers, m_device, and ven::MAX_FRAMES_IN_FLIGHT.

Referenced by Renderer().

Here is the call graph for this function:




Here is the caller graph for this function:




#### 7.31.3.4 endFrame()

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

Definition at line 69 of file renderer.cpp.

References ven::MAX_FRAMES_IN_FLIGHT.

#### 7.31.3.5 endSwapChainRenderPass()

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

Definition at line 119 of file renderer.cpp.

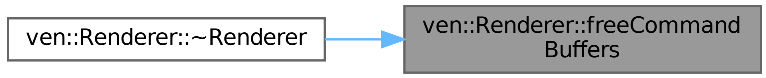
#### 7.31.3.6 freeCommandBuffers()

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

Definition at line 17 of file renderer.cpp.

Referenced by ∼Renderer().

Here is the caller graph for this function:




#### 7.31.3.7 getAspectRatio()

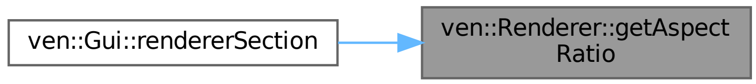
```
float ven::Renderer::getAspectRatio () const  [inline], [nodiscard]
```

Definition at line 39 of file Renderer.hpp.

References m_swapChain.

Referenced by ven::Gui::rendererSection().

Here is the caller graph for this function:




#### 7.31.3.8 getClearColor()

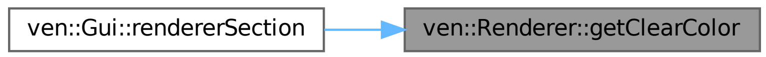
```
std::array< float, 4 > ven::Renderer::getClearColor () const  [inline], [nodiscard]
```

Definition at line 44 of file Renderer.hpp.

References m_clearValues.

Referenced by ven::Gui::rendererSection().

Here is the caller graph for this function:



#### 7.31.3.9 getCurrentCommandBuffer()

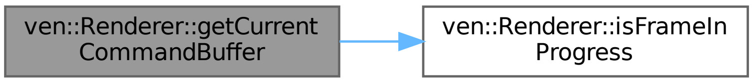
```
VkCommandBuffer ven::Renderer::getCurrentCommandBuffer () const  [inline], [nodiscard]
```

Definition at line 41 of file Renderer.hpp.

References isFrameInProgress(), m_commandBuffers, and m_currentFrameIndex.

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

Here is the call graph for this function:




Here is the caller graph for this function:

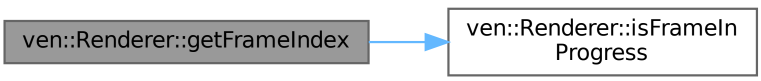

#### 7.31.3.10 getFrameIndex()

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

Definition at line 43 of file Renderer.hpp.

References isFrameInProgress(), and m_currentFrameIndex.

Here is the call graph for this function:



#### 7.31.3.11 getSwapChainRenderPass()

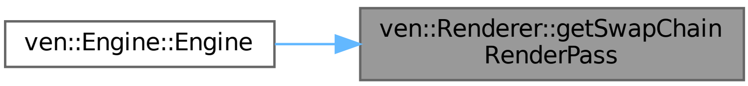
```
VkRenderPass ven::Renderer::getSwapChainRenderPass () const [inline], [nodiscard]
```

Definition at line 38 of file Renderer.hpp.

References m_swapChain.

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

Here is the caller graph for this function:




#### 7.31.3.12 getWindow()

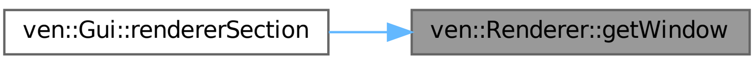
```
Window & ven::Renderer::getWindow () const [inline], [nodiscard]
```

Definition at line 51 of file Renderer.hpp.

References m_window.

Referenced by ven::Gui::rendererSection().

Here is the caller graph for this function:




#### 7.31.3.13 isFrameInProgress()

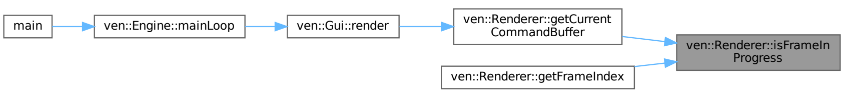
```
bool ven::Renderer::isFrameInProgress () const [inline], [nodiscard]
```

Definition at line 40 of file Renderer.hpp.

References m_isFrameStarted.

Referenced by getCurrentCommandBuffer(), and getFrameIndex().

Here is the caller graph for this function:



#### 7.31.3.14 operator=()

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

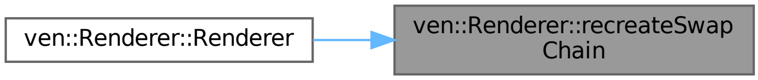
#### 7.31.3.15 recreateSwapChain()

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

Definition at line 23 of file renderer.cpp.

Referenced by Renderer().

Here is the caller graph for this function:




#### 7.31.3.16 setClearValue()

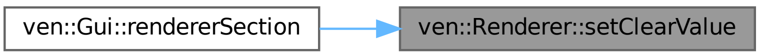
```
void ven::Renderer::setClearValue (
             const VkClearColorValue clearColorValue = DEFAULT_CLEAR_COLOR,
             const VkClearDepthStencilValue clearDepthValue = DEFAULT_CLEAR_DEPTH)  [inline]
```

Definition at line 53 of file Renderer.hpp.

References m_clearValues.

Referenced by ven::Gui::rendererSection().

Here is the caller graph for this function:



## 7.31.4 Member Data Documentation

### 7.31.4.1 m_clearValues

```
std::array<VkClearValue, 2> ven::Renderer::m_clearValues {DEFAULT_CLEAR_COLOR, 1.0F, 0.F}
[private]
```

Definition at line 69 of file Renderer.hpp.

Referenced by getClearColor(), and setClearValue().

### 7.31.4.2 m_commandBuffers

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

Definition at line 68 of file Renderer.hpp.

Referenced by createCommandBuffers(), and getCurrentCommandBuffer().

### 7.31.4.3 m_currentFrameIndex

```
unsigned long ven::Renderer::m_currentFrameIndex {0} [private]
```

Definition at line 72 of file Renderer.hpp.

Referenced by getCurrentCommandBuffer(), and getFrameIndex().

### 7.31.4.4 m_currentImageIndex

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

Definition at line 71 of file Renderer.hpp.

### 7.31.4.5 m_device

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

Definition at line 66 of file Renderer.hpp.

Referenced by createCommandBuffers().

### 7.31.4.6 m_isFrameStarted

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

Definition at line 73 of file Renderer.hpp.

Referenced by isFrameInProgress().

#### 7.31.4.7 m_swapChain

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

Definition at line 67 of file Renderer.hpp.

Referenced by getAspectRatio(), and getSwapChainRenderPass().

#### 7.31.4.8 m_window

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

Definition at line 65 of file Renderer.hpp.

Referenced by getWindow().

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

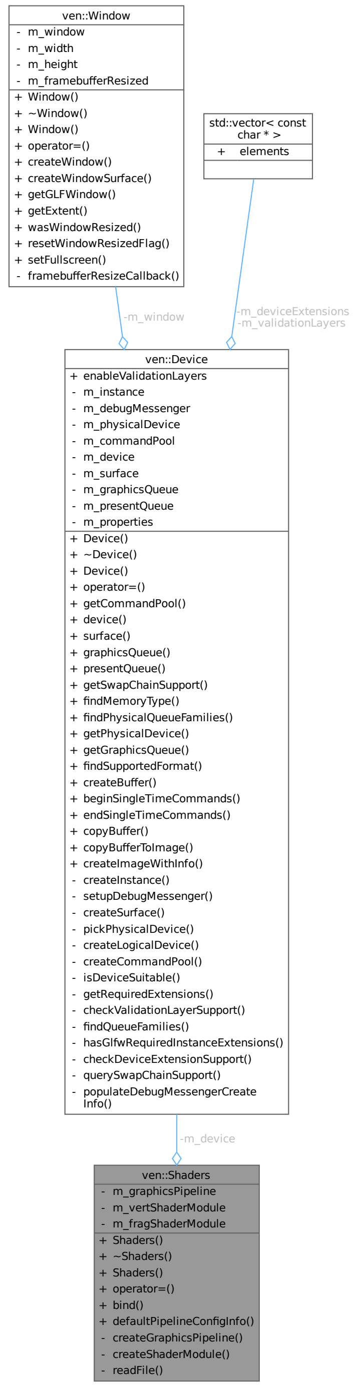
- /home/runner/work/VEngine/VEngine/include/VEngine/Renderer.hpp
- /home/runner/work/VEngine/VEngine/src/renderer.cpp

## 7.32 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}

## 7.32.1 Detailed Description

Class for shaders.

Definition at line 44 of file Shaders.hpp.

## 7.32.2 Constructor & Destructor Documentation

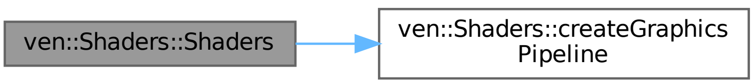
### 7.32.2.1 Shaders() [1/2]

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

Definition at line 48 of file Shaders.hpp.

References createGraphicsPipeline().

Here is the call graph for this function:



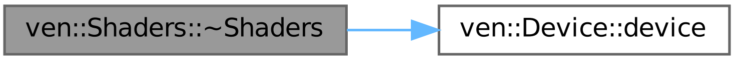
#### 7.32.2.2 ∼Shaders()

```
ven::Shaders::∼Shaders ()
```

Definition at line 8 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:




#### 7.32.2.3 Shaders() [2/2]

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

### 7.32.3 Member Function Documentation

#### 7.32.3.1 bind()

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

Definition at line 55 of file Shaders.hpp.

References m_graphicsPipeline.

#### 7.32.3.2 createGraphicsPipeline()

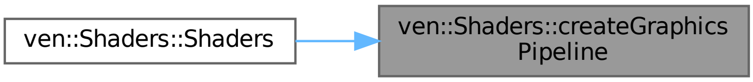
```
void ven::Shaders::createGraphicsPipeline (
            const std::string & vertFilepath,
            const std::string & fragFilepath,
            const PipelineConfigInfo & configInfo)  [private]
```

Definition at line 28 of file shaders.cpp.

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

Referenced by Shaders().

Here is the caller graph for this function:




#### 7.32.3.3 createShaderModule()

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

Definition at line 97 of file shaders.cpp.

#### 7.32.3.4 defaultPipelineConfigInfo()

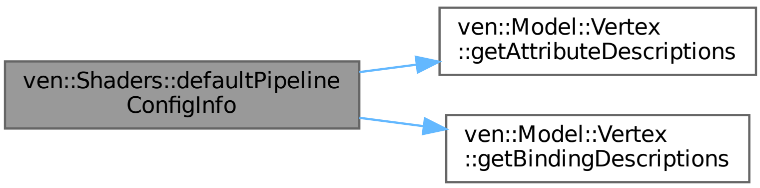
```
void ven::Shaders::defaultPipelineConfigInfo (
            PipelineConfigInfo & configInfo)  [static]
```

Definition at line 109 of file shaders.cpp.

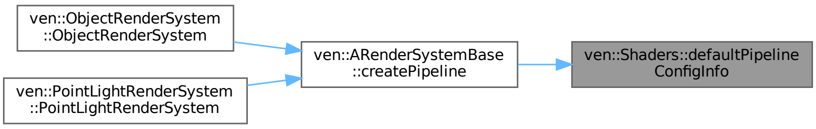
References ven::PipelineConfigInfo::attributeDescriptions, ven::PipelineConfigInfo::bindingDescriptions, ven::PipelineConfigInfo::colo
ven::PipelineConfigInfo::colorBlendInfo, ven::PipelineConfigInfo::depthStencilInfo, ven::PipelineConfigInfo::dynamicStateEnables, ven::PipelineConfigInfo::dynamicStateInfo,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

Referenced by ven::ARenderSystemBase::createPipeline().

Here is the call graph for this function:



Here is the caller graph for this function:




#### 7.32.3.5 operator=()

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

#### 7.32.3.6 readFile()

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

Definition at line 15 of file shaders.cpp.

### 7.32.4 Member Data Documentation

#### 7.32.4.1 m_device

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

Definition at line 63 of file Shaders.hpp.

Referenced by ∼Shaders().

#### 7.32.4.2 m_fragShaderModule

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

Definition at line 66 of file Shaders.hpp.

Referenced by ∼Shaders().

#### 7.32.4.3 m_graphicsPipeline

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

Definition at line 64 of file Shaders.hpp.

Referenced by bind(), and ∼Shaders().

#### 7.32.4.4 m_vertShaderModule

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

Definition at line 65 of file Shaders.hpp.

Referenced by ∼Shaders().

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

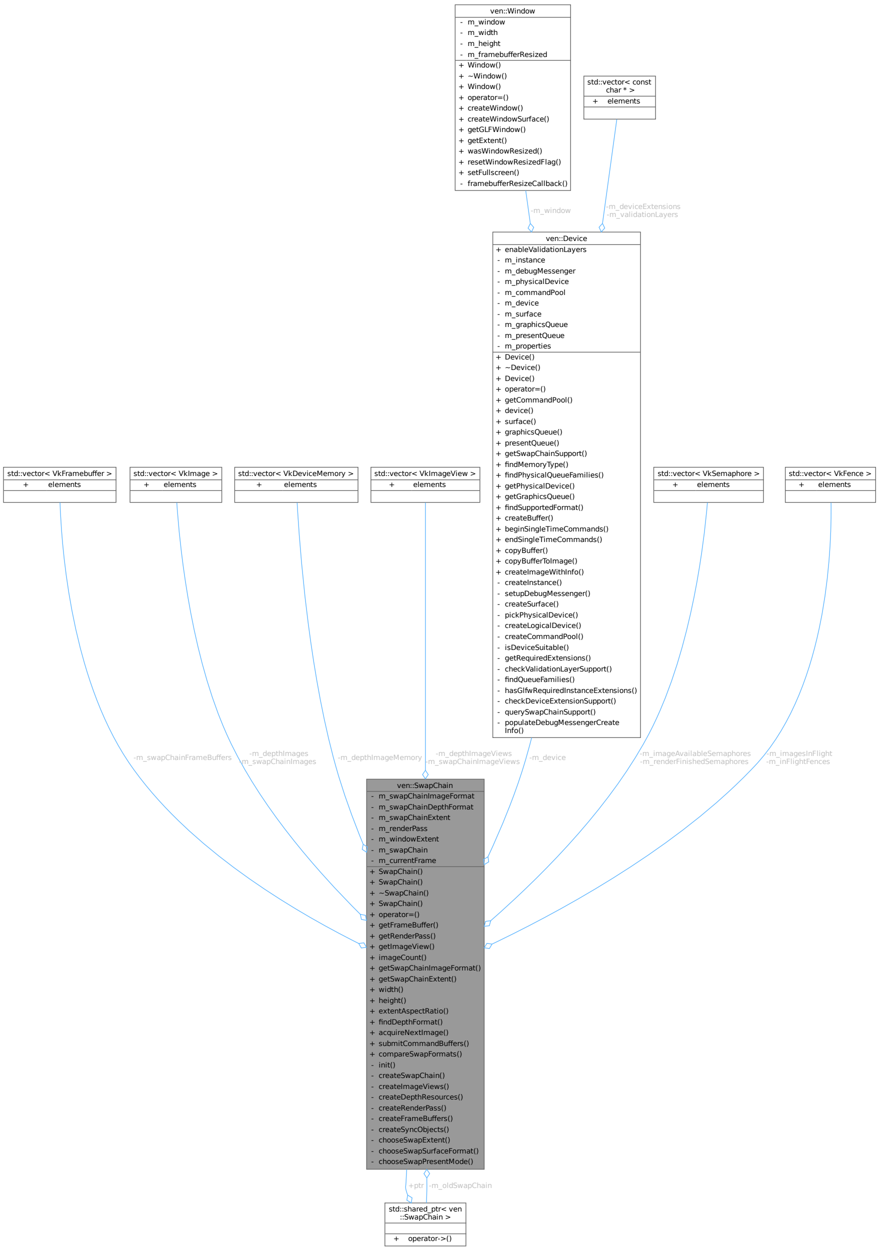
- /home/runner/work/VEngine/VEngine/include/VEngine/Shaders.hpp
- /home/runner/work/VEngine/VEngine/src/shaders.cpp

## 7.33 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 &swapChain) const

### 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 m_swapChainImageFormat {}
- VkFormat m_swapChainDepthFormat {}
- VkExtent2D m_swapChainExtent {}
- std::vector< VkFramebuffer > m_swapChainFrameBuffers
- VkRenderPass m_renderPass {}
- std::vector< VkImage > m_depthImages
- std::vector< VkDeviceMemory > m_depthImageMemory
- std::vector< VkImageView > m_depthImageViews
- std::vector< VkImage > m_swapChainImages
- std::vector< VkImageView > m_swapChainImageViews
- Device & m_device
- VkExtent2D m_windowExtent
- VkSwapchainKHR m_swapChain {}
- std::shared_ptr< SwapChain > m_oldSwapChain
- std::vector< VkSemaphore > m_imageAvailableSemaphores
- std::vector< VkSemaphore > m_renderFinishedSemaphores
- std::vector< VkFence > m_inFlightFences
- std::vector< VkFence > m_imagesInFlight
- size_t m_currentFrame {0}

### 7.33.1 Detailed Description

Class for swap chain.

Definition at line 23 of file SwapChain.hpp.

### 7.33.2 Constructor & Destructor Documentation

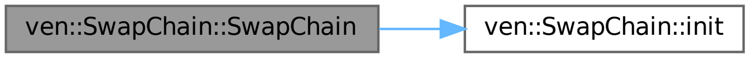
#### 7.33.2.1 SwapChain() [1/3]

```
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:



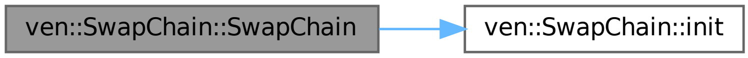

#### 7.33.2.2 SwapChain() [2/3]

```
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 m_oldSwapChain.

Here is the call graph for this function:



#### 7.33.2.3 ∼SwapChain()

```
ven::SwapChain::∼SwapChain ()
```

Definition at line 7 of file swapChain.cpp.

References ven::Device::device(), m_depthImageMemory, m_depthImages, m_depthImageViews, m_device, m_imageAvailableSemaphores, m_inFlightFences, m_renderFinishedSemaphores, m_renderPass, m_swapChain, m_swapChainFrameBuffers, m_swapChainImageViews, and ven::MAX_FRAMES_IN_FLIGHT.

Here is the call graph for this function:


#### 7.33.2.4 SwapChain() [3/3]

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

### 7.33.3 Member Function Documentation

#### 7.33.3.1 acquireNextImage()

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

Definition at line 49 of file swapChain.cpp.

#### 7.33.3.2 chooseSwapExtent()

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

Definition at line 362 of file swapChain.cpp.

#### 7.33.3.3 chooseSwapPresentMode()

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

Definition at line 342 of file swapChain.cpp.

#### 7.33.3.4 chooseSwapSurfaceFormat()

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

Definition at line 331 of file swapChain.cpp.

#### 7.33.3.5 compareSwapFormats()

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

Definition at line 49 of file SwapChain.hpp.

References m_swapChainDepthFormat, and m_swapChainImageFormat.

#### 7.33.3.6 createDepthResources()

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

Definition at line 262 of file swapChain.cpp.

#### 7.33.3.7 createFrameBuffers()

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

Definition at line 240 of file swapChain.cpp.

#### 7.33.3.8 createImageViews()

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

Definition at line 160 of file swapChain.cpp.

#### 7.33.3.9 createRenderPass()

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

Definition at line 181 of file swapChain.cpp.

#### 7.33.3.10 createSwapChain()

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

Definition at line 103 of file swapChain.cpp.

#### 7.33.3.11 createSyncObjects()

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

Definition at line 308 of file swapChain.cpp.

References ven::MAX_FRAMES_IN_FLIGHT.

#### 7.33.3.12 extentAspectRatio()

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

Definition at line 43 of file SwapChain.hpp.

References m_swapChainExtent.

#### 7.33.3.13 findDepthFormat()

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

Definition at line 374 of file swapChain.cpp.

#### 7.33.3.14 getFrameBuffer()

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

Definition at line 34 of file SwapChain.hpp.

References m_swapChainFrameBuffers.

#### 7.33.3.15 getImageView()

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

Definition at line 36 of file SwapChain.hpp.

References m_swapChainImageViews.

#### 7.33.3.16 getRenderPass()

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

Definition at line 35 of file SwapChain.hpp.

References m_renderPass.

#### 7.33.3.17 getSwapChainExtent()

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

Definition at line 39 of file SwapChain.hpp.

References m_swapChainExtent.

#### 7.33.3.18 getSwapChainImageFormat()

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

Definition at line 38 of file SwapChain.hpp.

References m_swapChainImageFormat.

#### 7.33.3.19 height()

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

Definition at line 41 of file SwapChain.hpp.

References m_swapChainExtent.

#### 7.33.3.20 imageCount()

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

Definition at line 37 of file SwapChain.hpp.

References m_swapChainImages.

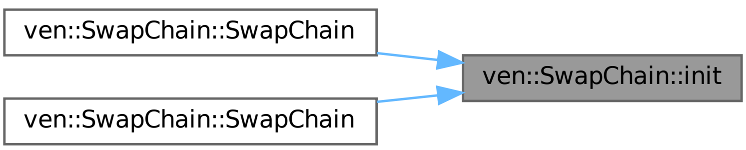
#### 7.33.3.21 init()

```
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:



#### 7.33.3.22 operator=()

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

#### 7.33.3.23 submitCommandBuffers()

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

Definition at line 56 of file swapChain.cpp.

References ven::MAX_FRAMES_IN_FLIGHT.

#### 7.33.3.24 width()

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

Definition at line 40 of file SwapChain.hpp.

References m_swapChainExtent.

### 7.33.4 Member Data Documentation

#### 7.33.4.1 m_currentFrame

```
size_t ven::SwapChain::m_currentFrame {0}  [private]
```

Definition at line 88 of file SwapChain.hpp.

#### 7.33.4.2 m_depthImageMemory

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

Definition at line 73 of file SwapChain.hpp.

Referenced by ∼SwapChain().

#### 7.33.4.3 m_depthImages

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

Definition at line 72 of file SwapChain.hpp.

Referenced by ∼SwapChain().

#### 7.33.4.4 m_depthImageViews

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

Definition at line 74 of file SwapChain.hpp.

Referenced by ∼SwapChain().

#### 7.33.4.5 m_device

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

Definition at line 78 of file SwapChain.hpp.

Referenced by ∼SwapChain().

#### 7.33.4.6 m_imageAvailableSemaphores

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

Definition at line 84 of file SwapChain.hpp.

Referenced by ∼SwapChain().

#### 7.33.4.7 m_imagesInFlight

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

Definition at line 87 of file SwapChain.hpp.

#### 7.33.4.8 m_inFlightFences

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

Definition at line 86 of file SwapChain.hpp.

Referenced by ∼SwapChain().

#### 7.33.4.9 m_oldSwapChain

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

Definition at line 82 of file SwapChain.hpp.

Referenced by SwapChain().

#### 7.33.4.10 m_renderFinishedSemaphores

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

Definition at line 85 of file SwapChain.hpp.

Referenced by ~SwapChain().

#### 7.33.4.11 m_renderPass

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

Definition at line 70 of file SwapChain.hpp.

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

#### 7.33.4.12 m_swapChain

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

Definition at line 81 of file SwapChain.hpp.

Referenced by ~SwapChain().

#### 7.33.4.13 m_swapChainDepthFormat

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

Definition at line 66 of file SwapChain.hpp.

Referenced by compareSwapFormats().

#### 7.33.4.14 m_swapChainExtent

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

Definition at line 67 of file SwapChain.hpp.

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

#### 7.33.4.15 m_swapChainFrameBuffers

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

Definition at line 69 of file SwapChain.hpp.

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

#### 7.33.4.16 m_swapChainImageFormat

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

Definition at line 65 of file SwapChain.hpp.

Referenced by compareSwapFormats(), and getSwapChainImageFormat().

#### 7.33.4.17 m_swapChainImages

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

Definition at line 75 of file SwapChain.hpp.

Referenced by imageCount().

#### 7.33.4.18 m_swapChainImageViews

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

Definition at line 76 of file SwapChain.hpp.

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

#### 7.33.4.19 m_windowExtent

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

Definition at line 79 of file SwapChain.hpp.

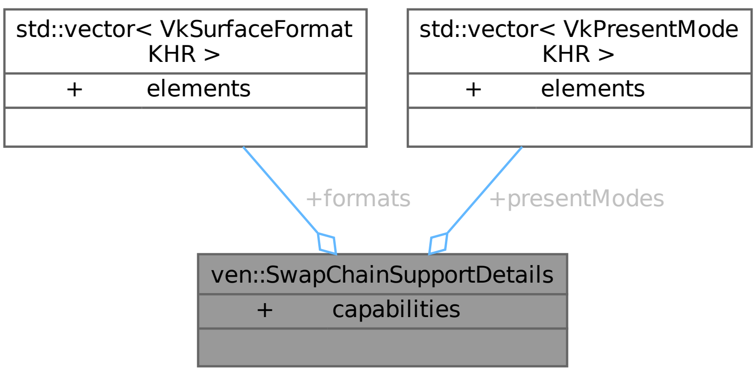
The documentation for this class was generated from the following files:

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

## 7.34 ven::SwapChainSupportDetails Struct Reference

`#include <Device.hpp>`

Collaboration diagram for ven::SwapChainSupportDetails:



**Public Attributes**

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

### 7.34.1 Detailed Description

Definition at line 15 of file Device.hpp.

### 7.34.2 Member Data Documentation

#### 7.34.2.1 capabilities

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

Definition at line 16 of file Device.hpp.

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

#### 7.34.2.2 formats

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

Definition at line 17 of file Device.hpp.

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

#### 7.34.2.3 presentModes

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

Definition at line 18 of file Device.hpp.

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

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

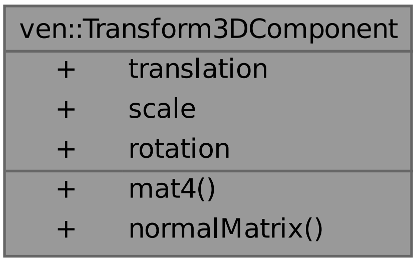
- /home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp

## 7.35 ven::Transform3DComponent Class Reference

Class for 3D transformation.

```
#include <Transform3DComponent.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 {}

### 7.35.1 Detailed Description

Class for 3D transformation.

Definition at line 18 of file Transform3DComponent.hpp.

### 7.35.2 Member Function Documentation

#### 7.35.2.1 mat4()

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

Definition at line 3 of file transform3DComponent.cpp.

References rotation, scale, and translation.

#### 7.35.2.2 normalMatrix()

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

Definition at line 38 of file transform3DComponent.cpp.

### 7.35.3 Member Data Documentation

#### 7.35.3.1 rotation

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

Definition at line 24 of file Transform3DComponent.hpp.

Referenced by ven::Gui::cameraSection(), mat4(), and ven::EventManager::moveCamera().

#### 7.35.3.2 scale

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

Definition at line 23 of file Transform3DComponent.hpp.

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

#### 7.35.3.3 translation

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

Definition at line 22 of file Transform3DComponent.hpp.

Referenced by ven::Gui::cameraSection(), ven::Engine::loadObjects(), mat4(), and ven::EventManager::moveCamera().

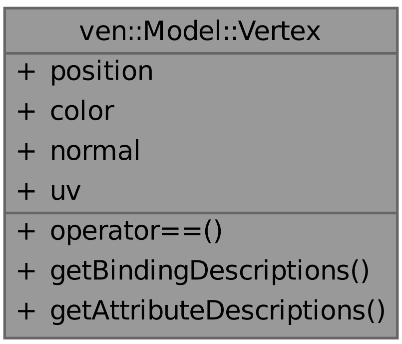
The documentation for this class was generated from the following files:

- /home/runner/work/VEngine/VEngine/include/VEngine/Transform3DComponent.hpp
- /home/runner/work/VEngine/VEngine/src/transform3DComponent.cpp

## 7.36 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 {}

## 7.36.1 Detailed Description

Definition at line 29 of file Model.hpp.

## 7.36.2 Member Function Documentation

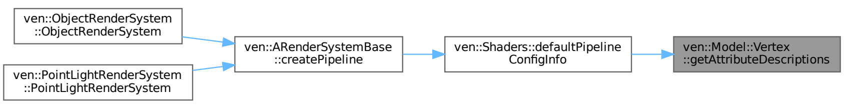
### 7.36.2.1 getAttributeDescriptions()

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

Definition at line 105 of file model.cpp.

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

Here is the caller graph for this function:



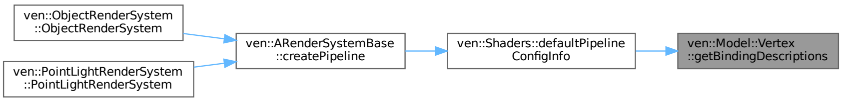
#### 7.36.2.2 getBindingDescriptions()

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

Definition at line 96 of file model.cpp.

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

Here is the caller graph for this function:




#### 7.36.2.3 operator==()

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

Definition at line 38 of file Model.hpp.

References color, normal, position, and uv.

### 7.36.3 Member Data Documentation

#### 7.36.3.1 color

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

Definition at line 31 of file Model.hpp.

Referenced by operator==().

#### 7.36.3.2 normal

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

Definition at line 32 of file Model.hpp.

Referenced by operator==().

#### 7.36.3.3 position

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

Definition at line 30 of file Model.hpp.

Referenced by operator==(), and ven::Model::Builder::processMesh().

#### 7.36.3.4 uv

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

Definition at line 33 of file Model.hpp.

Referenced by operator==().

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

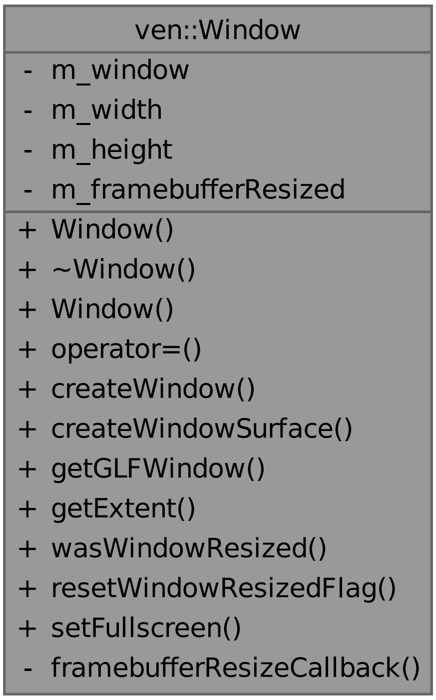
- /home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp
- /home/runner/work/VEngine/VEngine/src/model.cpp

## 7.37 ven::Window Class Reference

Class for window.

`#include <Window.hpp>`

Collaboration diagram for ven::Window:



**Public Member Functions**

- Window (const uint32_t width=DEFAULT_WIDTH, const uint32_t height=DEFAULT_HEIGHT, const std↩
::string &title=DEFAULT_TITLE.data())
- ∼Window ()
- Window (const Window &)=delete
- Window & operator= (const Window &)=delete
- 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 ()
- void setFullscreen (bool fullscreen, uint32_t width, uint32_t height)

**Static Private Member Functions**

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

**Private Attributes**

- GLFWwindow ∗ m_window {nullptr}
- uint32_t m_width {DEFAULT_WIDTH}
- uint32_t m_height {DEFAULT_HEIGHT}
- bool m_framebufferResized = false

## 7.37.1 Detailed Description

Class for window.

Definition at line 26 of file Window.hpp.

## 7.37.2 Constructor & Destructor Documentation

### 7.37.2.1 Window() [1/2]

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

Definition at line 30 of file Window.hpp.

### 7.37.2.2 ∼Window()

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

Definition at line 31 of file Window.hpp.

References m_window.

#### 7.37.2.3 Window() [2/2]

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

### 7.37.3 Member Function Documentation

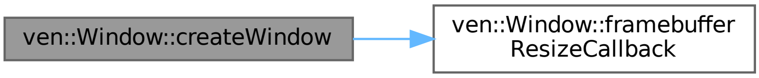
#### 7.37.3.1 createWindow()

```
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:




#### 7.37.3.2 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:

#### 7.37.3.3 framebufferResizeCallback()

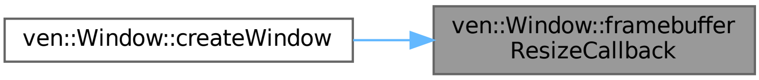
```
void ven::Window::framebufferResizeCallback (
            GLFWwindow * window,
            int width,
            int height) [static], [private]
```

Definition at line 31 of file window.cpp.

References m_framebufferResized.

Referenced by createWindow().

Here is the caller graph for this function:




#### 7.37.3.4 getExtent()

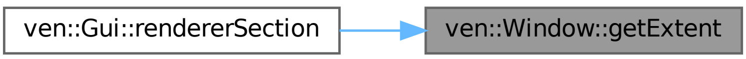
```
VkExtent2D ven::Window::getExtent () const [inline], [nodiscard]
```

Definition at line 41 of file Window.hpp.

References m_height, and m_width.

Referenced by ven::Gui::rendererSection().

Here is the caller graph for this function:



### 7.37.3.5 getGLFWindow()

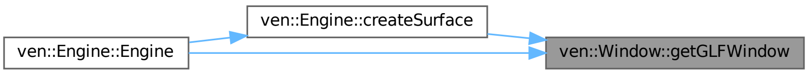
```
GLFWwindow * ven::Window::getGLFWindow () const  [inline], [nodiscard]
```

Definition at line 39 of file Window.hpp.

References m_window.

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

Here is the caller graph for this function:




### 7.37.3.6 operator=()

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

### 7.37.3.7 resetWindowResizedFlag()

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

Definition at line 43 of file Window.hpp.

References m_framebufferResized.

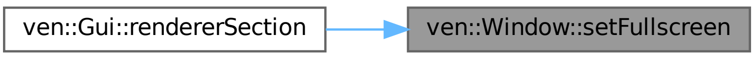
### 7.37.3.8 setFullscreen()

```
void ven::Window::setFullscreen (
            bool fullscreen,
            uint32_t width,
            uint32_t height)
```

Definition at line 39 of file window.cpp.

Referenced by ven::Gui::rendererSection().

Here is the caller graph for this function:



#### 7.37.3.9 wasWindowResized()

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

Definition at line 42 of file Window.hpp.

References m_framebufferResized.

### 7.37.4 Member Data Documentation

#### 7.37.4.1 m_framebufferResized

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

Definition at line 55 of file Window.hpp.

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

#### 7.37.4.2 m_height

```
uint32_t ven::Window::m_height {DEFAULT_HEIGHT}  [private]
```

Definition at line 53 of file Window.hpp.

Referenced by getExtent().

#### 7.37.4.3 m_width

```
uint32_t ven::Window::m_width {DEFAULT_WIDTH}  [private]
```

Definition at line 52 of file Window.hpp.

Referenced by getExtent().

#### 7.37.4.4 m_window

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

Definition at line 51 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 8

# File Documentation

## 8.1 /home/runner/work/VEngine/VEngine/assets/shaders/fragment_point_light.frag File Reference

## 8.2 fragment_point_light.frag

Go to the documentation of this file.

```
#version 450

layout(location = 0) in vec2 fragOffset;
layout(location = 0) out vec4 outColor;

struct PointLight {
    vec4 position; // ignore w
    vec4 color; // w is intensity
};

layout(set = 0, binding = 0) uniform GlobalUbo {
    mat4 projection;
    mat4 view;
    mat4 invView;
    vec4 ambientLightColor; // w is intensity
    PointLight pointLights[10];
    int numLights;
} ubo;

layout(push_constant) uniform Push {
    vec4 position;
    vec4 color;
    float radius;
} push;

const float M_PI = 3.1415926538;

void main() {
    float dis = length(fragOffset);
    if (dis >= 1.0) {
        discard;
    }

    float cosDis = 0.5 * (cos(dis * M_PI) + 1.0);
    outColor = vec4(push.color.rgb + 0.5 * cosDis, cosDis);
}
```

## 8.3 /home/runner/work/VEngine/VEngine/assets/shaders/fragment_shader.frag File Reference

## 8.4 fragment_shader.frag

Go to the documentation of this file.

```
#version 450

layout(location = 0) in vec3 fragColor;
layout(location = 1) in vec3 fragPosWorld;
layout(location = 2) in vec3 fragNormalWorld;

layout(location = 0) out vec4 outColor;

struct PointLight {
  vec4 position; // ignore w
  vec4 color; // w is intensity
};

layout(set = 0, binding = 0) uniform GlobalUbo {
  mat4 projection;
  mat4 view;
  mat4 invView;
  vec4 ambientLightColor; // w is intensity
  PointLight pointLights[10];
  int numLights;
} ubo;

layout(push_constant) uniform Push {
  mat4 modelMatrix;
  mat4 normalMatrix;
} push;

void main() {
  vec3 specularLight = vec3(0.0);
  vec3 surfaceNormal = normalize(gl_FrontFacing ? fragNormalWorld : -fragNormalWorld);
  vec3 diffuseLight = ubo.ambientLightColor.rgb * ubo.ambientLightColor.a;

  vec3 cameraPosWorld = ubo.invView[3].xyz;
  vec3 viewDirection = normalize(cameraPosWorld - fragPosWorld);

  for (int i = 0; i < ubo.numLights; i++) {
    PointLight light = ubo.pointLights[i];
    vec3 directionToLight = light.position.xyz - fragPosWorld;
    float distanceSquared = dot(directionToLight, directionToLight);
    float attenuation = 1.0 / distanceSquared; // distance squared
    directionToLight = normalize(directionToLight);

    float cosAngIncidence = max(dot(surfaceNormal, directionToLight), 0);
    vec3 intensity = light.color.rgb * light.color.a * attenuation;
    vec3 reflectionDirection = reflect(-directionToLight, surfaceNormal);
    float cosAngReflection = max(dot(viewDirection, reflectionDirection), 0);

    // diffuse lighting
    diffuseLight += intensity * cosAngIncidence;
    // specular lighting
    float specular = pow(cosAngReflection, 512);
    specularLight += intensity * specular * step(0.0, cosAngIncidence) * step(0.0, cosAngReflection);
  }

  outColor = vec4(fragColor * (diffuseLight + specularLight), 1.0);
}
```

## 8.5 /home/runner/work/VEngine/VEngine/assets/shaders/vertex_point_light.vert File Reference

## 8.6 vertex_point_light.vert

Go to the documentation of this file.

```
#version 450

const vec2 OFFSETS[6] = vec2[](
vec2(-1.0, -1.0),
vec2(-1.0, 1.0),
vec2(1.0, -1.0),
vec2(1.0, -1.0),
vec2(-1.0, 1.0),
vec2(1.0, 1.0)
);

layout(location = 0) out vec2 fragOffset;

struct PointLight {
```

```
    vec4 position; // ignore w
    vec4 color; // w is intensity
};

layout(set = 0, binding = 0) uniform GlobalUbo {
    mat4 projection;
    mat4 view;
    mat4 invView;
    vec4 ambientLightColor; // w is intensity
    PointLight pointLights[10];
    int numLights;
} ubo;

layout(push_constant) uniform Push {
    vec4 position;
    vec4 color;
    float radius;
} push;

void main() {
    fragOffset = OFFSETS[gl_VertexIndex];
    vec3 cameraRightWorld = vec3(ubo.view[0][0], ubo.view[1][0], ubo.view[2][0]);
    vec3 cameraUpWorld = vec3(ubo.view[0][1], ubo.view[1][1], ubo.view[2][1]);

    vec3 positionWorld = push.position.xyz
    + push.radius * fragOffset.x * cameraRightWorld
    + push.radius * fragOffset.y * cameraUpWorld;

    gl_Position = ubo.projection * ubo.view * vec4(positionWorld, 1.0);
}
```

## 8.7 /home/runner/work/VEngine/VEngine/assets/shaders/vertex_shader.vert File Reference

## 8.8 vertex_shader.vert

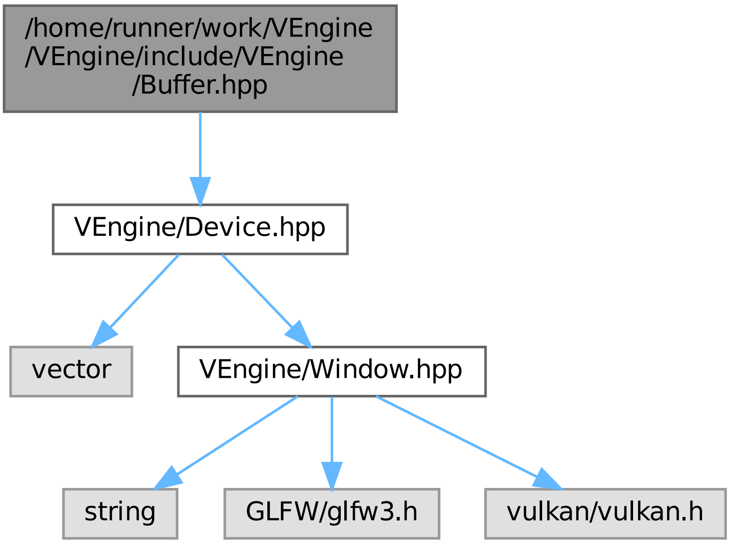
Go to the documentation of this file.

```
#version 450

layout(location = 0) in vec3 position;
layout(location = 1) in vec3 color;
layout(location = 2) in vec3 normal;
layout(location = 3) in vec2 uv;

layout(location = 0) out vec3 fragColor;
layout(location = 1) out vec3 fragPosWorld;
layout(location = 2) out vec3 fragNormalWorld;

struct PointLight {
  vec4 position; // ignore w
  vec4 color; // w is intensity
};

layout(set = 0, binding = 0) uniform GlobalUbo {
  mat4 projection;
  mat4 view;
  mat4 invView;
  vec4 ambientLightColor; // w is intensity
  PointLight pointLights[10];
  int numLights;
} ubo;

layout(push_constant) uniform Push {
  mat4 modelMatrix;
  mat4 normalMatrix;
} push;

void main() {
  vec4 positionWorld = push.modelMatrix * vec4(position, 1.0);
  gl_Position = ubo.projection * ubo.view * positionWorld;
  fragNormalWorld = normalize(mat3(push.normalMatrix) * normal);
  fragPosWorld = positionWorld.xyz;
  fragColor = color;
}
```

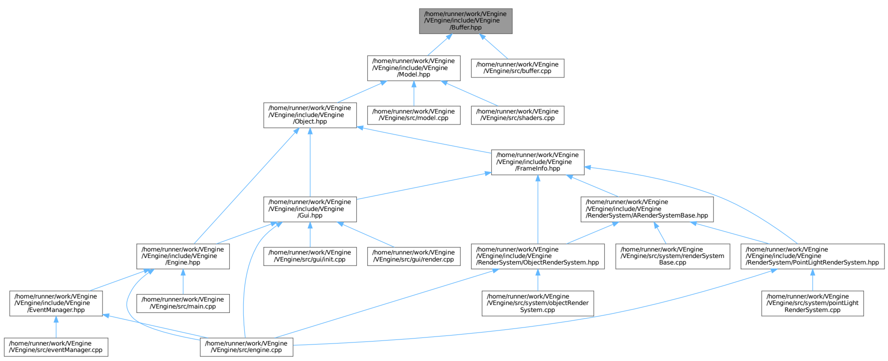
## 8.9 /home/runner/work/VEngine/VEngine/include/VEngine/Buffer.hpp File Reference

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

### 8.9.1 Detailed Description

This file contains the Buffer class.

Definition in file Buffer.hpp.

## 8.10 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; }
```

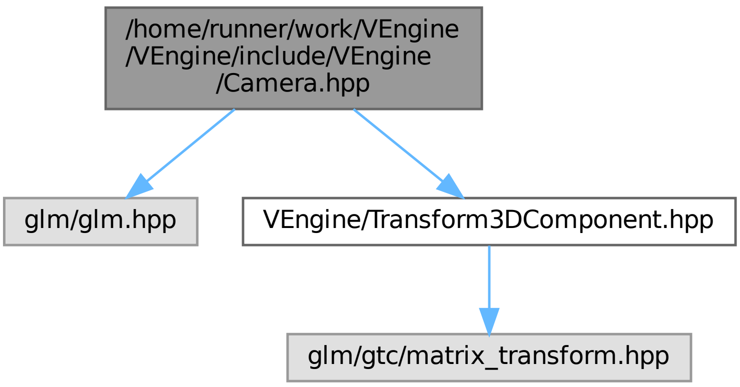
```
            [[nodiscard]] VkDeviceSize getAlignmentSize() const { return m_alignmentSize; }
            [[nodiscard]] VkBufferUsageFlags getUsageFlags() const { return m_usageFlags; }
            [[nodiscard]] VkMemoryPropertyFlags getMemoryPropertyFlags() const { return
m_memoryPropertyFlags; }
            [[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
```

## 8.11 /home/runner/work/VEngine/VEngine/include/VEngine/Camera.hpp File Reference

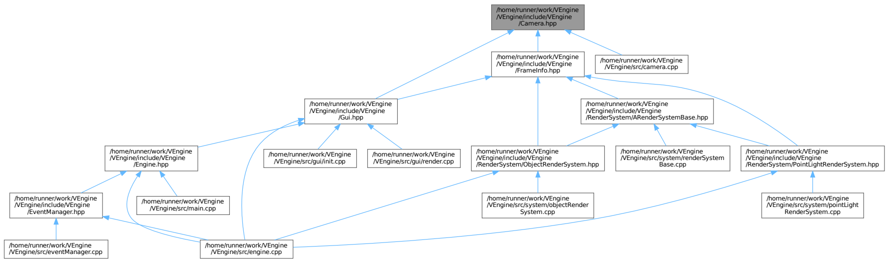
This file contains the Camera class.

```
#include <glm/glm.hpp>
#include "VEngine/Transform3DComponent.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
- static constexpr float ven::DEFAULT_MOVE_SPEED = 3.F
- static constexpr float ven::DEFAULT_LOOK_SPEED = 1.5F

## 8.11.1 Detailed Description

This file contains the Camera class.

Definition in file Camera.hpp.

# 8.12 Camera.hpp

Go to the documentation of this file.

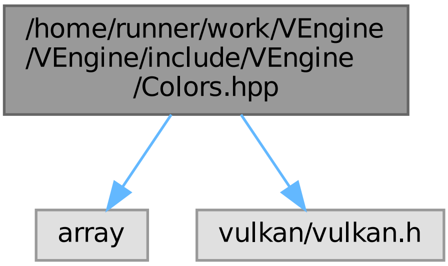
```
///
/// @file Camera.hpp
/// @brief This file contains the Camera class
/// @namespace ven
///

#pragma once

#include <glm/glm.hpp>

#include "VEngine/Transform3DComponent.hpp"

```

```cpp
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;

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

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

        public:

            Camera() = default;
            ~Camera() = default;

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

            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 = {0.F, -1.F, 0.F});
            void setViewTarget(const glm::vec3 position, const glm::vec3 target, const glm::vec3 up = {0.F, -1.F, 0.F}) { setViewDirection(position, target - position, up); }
            void setViewXYZ(glm::vec3 position, glm::vec3 rotation);
            void setFov(const float fov) { m_fov = fov; }
            void setNear(const float near) { m_near = near; }
            void setFar(const float far) { m_far = far; }
            void setMoveSpeed(const float moveSpeed) { m_moveSpeed = moveSpeed; }
            void setLookSpeed(const float lookSpeed) { m_lookSpeed = lookSpeed; }

            [[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; }
            [[nodiscard]] float getMoveSpeed() const { return m_moveSpeed; }
            [[nodiscard]] float getLookSpeed() const { return m_lookSpeed; }

            Transform3DComponent transform3D{DEFAULT_POSITION, {1.F, 1.F, 1.F}, DEFAULT_ROTATION};

        private:

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

    }; // class Camera

} // namespace ven
```

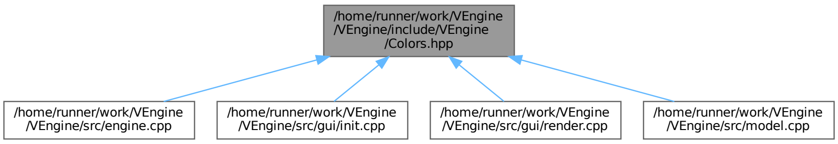
## 8.13 /home/runner/work/VEngine/VEngine/include/VEngine/Colors.hpp File Reference

```cpp
#include <array>
#include <vulkan/vulkan.h>
```

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

### Variables

- static constexpr float ven::COLOR_MAX = 255.0F

## 8.14 Colors.hpp

Go to the documentation of this file.

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

#pragma once
```

```
#include <array>

#include <vulkan/vulkan.h>

namespace ven {

    static constexpr float COLOR_MAX = 255.0F;

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

        public:

            static constexpr glm::vec3 WHITE_3 = glm::vec3(COLOR_MAX) / COLOR_MAX;
            static constexpr glm::vec4 WHITE_4 = { 1.0F, 1.0F, 1.0F, 1.0F };
            static constexpr VkClearColorValue WHITE_V = { { 1.0F, 1.0F, 1.0F, 1.0F } };

            static constexpr glm::vec3 BLACK_3 = glm::vec3(0.0F);
            static constexpr glm::vec4 BLACK_4 = { 0.0F, 0.0F, 0.0F, 1.0F };
            static constexpr VkClearColorValue BLACK_V = { { 0.0F, 0.0F, 0.0F, 1.0F } };

            static constexpr glm::vec3 RED_3 = glm::vec3(COLOR_MAX, 0.0F, 0.0F) / COLOR_MAX;
            static constexpr glm::vec4 RED_4 = { 1.0F, 0.0F, 0.0F, 1.0F };
            static constexpr VkClearColorValue RED_V = { { 1.0F, 0.0F, 0.0F, 1.0F } };

            static constexpr glm::vec3 GREEN_3 = glm::vec3(0.0F, COLOR_MAX, 0.0F) / COLOR_MAX;
            static constexpr glm::vec4 GREEN_4 = { 0.0F, 1.0F, 0.0F, 1.0F };
            static constexpr VkClearColorValue GREEN_V = { { 0.0F, 1.0F, 0.0F, 1.0F } };

            static constexpr glm::vec3 BLUE_3 = glm::vec3(0.0F, 0.0F, COLOR_MAX) / COLOR_MAX;
            static constexpr glm::vec4 BLUE_4 = { 0.0F, 0.0F, 1.0F, 1.0F };
            static constexpr VkClearColorValue BLUE_V = { { 0.0F, 0.0F, 1.0F, 1.0F } };

            static constexpr glm::vec3 YELLOW_3 = glm::vec3(COLOR_MAX, COLOR_MAX, 0.0F) / COLOR_MAX;
            static constexpr glm::vec4 YELLOW_4 = { 1.0F, 1.0F, 0.0F, 1.0F };
            static constexpr VkClearColorValue YELLOW_V = { { 1.0F, 1.0F, 0.0F, 1.0F } };

            static constexpr glm::vec3 CYAN_3 = glm::vec3(0.0F, COLOR_MAX, COLOR_MAX) / COLOR_MAX;
            static constexpr glm::vec4 CYAN_4 = { 0.0F, 1.0F, 1.0F, 1.0F };
            static constexpr VkClearColorValue CYAN_V = { { 0.0F, 1.0F, 1.0F, 1.0F } };

            static constexpr glm::vec3 MAGENTA_3 = glm::vec3(COLOR_MAX, 0.0F, COLOR_MAX) / COLOR_MAX;
            static constexpr glm::vec4 MAGENTA_4 = { 1.0F, 0.0F, 1.0F, 1.0F };
            static constexpr VkClearColorValue MAGENTA_V = { { 1.0F, 0.0F, 1.0F, 1.0F } };

            static constexpr glm::vec3 SILVER_3 = glm::vec3(192.0F, 192.0F, 192.0F) / COLOR_MAX;
            static constexpr glm::vec4 SILVER_4 = { 0.75F, 0.75F, 0.75F, 1.0F };
            static constexpr VkClearColorValue SILVER_V = { { 0.75F, 0.75F, 0.75F, 1.0F } };

            static constexpr glm::vec3 GRAY_3 = glm::vec3(128.0F, 128.0F, 128.0F) / COLOR_MAX;
            static constexpr glm::vec4 GRAY_4 = { 0.5F, 0.5F, 0.5F, 1.0F };
            static constexpr VkClearColorValue GRAY_V = { { 0.5F, 0.5F, 0.5F, 1.0F } };

            static constexpr glm::vec3 MAROON_3 = glm::vec3(128.0F, 0.0F, 0.0F) / COLOR_MAX;
            static constexpr glm::vec4 MAROON_4 = { 0.5F, 0.0F, 0.0F, 1.0F };
            static constexpr VkClearColorValue MAROON_V = { { 0.5F, 0.0F, 0.0F, 1.0F } };

            static constexpr glm::vec3 OLIVE_3 = glm::vec3(128.0F, 128.0F, 0.0F) / COLOR_MAX;
            static constexpr glm::vec4 OLIVE_4 = { 0.5F, 0.5F, 0.0F, 1.0F };
            static constexpr VkClearColorValue OLIVE_V = { { 0.5F, 0.5F, 0.0F, 1.0F } };

            static constexpr glm::vec3 LIME_3 = glm::vec3(0.0F, COLOR_MAX, 0.0F) / COLOR_MAX;
            static constexpr glm::vec4 LIME_4 = { 0.0F, 1.0F, 0.0F, 1.0F };
            static constexpr VkClearColorValue LIME_V = { { 0.0F, 1.0F, 0.0F, 1.0F } };

            static constexpr glm::vec3 AQUA_3 = glm::vec3(0.0F, COLOR_MAX, COLOR_MAX) / COLOR_MAX;
            static constexpr glm::vec4 AQUA_4 = { 0.0F, 1.0F, 1.0F, 1.0F };
            static constexpr VkClearColorValue AQUA_V = { { 0.0F, 1.0F, 1.0F, 1.0F } };

            static constexpr glm::vec3 TEAL_3 = glm::vec3(0.0F, 128.0F, 128.0F) / COLOR_MAX;
            static constexpr glm::vec4 TEAL_4 = { 0.0F, 0.5F, 0.5F, 1.0F };
            static constexpr VkClearColorValue TEAL_V = { { 0.0F, 0.5F, 0.5F, 1.0F } };

            static constexpr glm::vec3 NAVY_3 = glm::vec3(0.0F, 0.0F, 128.0F) / COLOR_MAX;
            static constexpr glm::vec4 NAVY_4 = { 0.0F, 0.0F, 0.5F, 1.0F };
            static constexpr VkClearColorValue NAVY_V = { { 0.0F, 0.0F, 0.5F, 1.0F } };

            static constexpr glm::vec3 FUCHSIA_3 = glm::vec3(COLOR_MAX, 0.0F, COLOR_MAX) / COLOR_MAX;
            static constexpr glm::vec4 FUCHSIA_4 = { 1.0F, 0.0F, 1.0F, 1.0F };
            static constexpr VkClearColorValue FUCHSIA_V = { { 1.0F, 0.0F, 1.0F, 1.0F } };

            static constexpr glm::vec3 NIGHT_BLUE_3 = glm::vec3(25.0F, 25.0F, 112.0F) / COLOR_MAX;
```

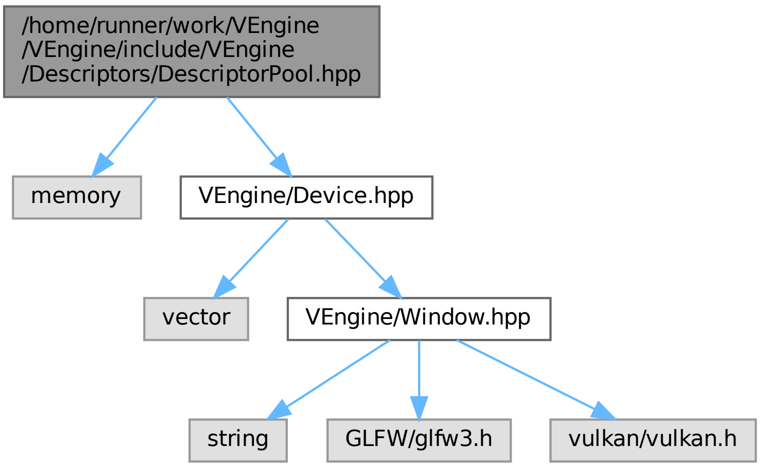
```
        static constexpr glm::vec4 NIGHT_BLUE_4 = { 0.098F, 0.098F, 0.439F, 1.0F };
        static constexpr VkClearColorValue NIGHT_BLUE_V = { { 0.098F, 0.098F, 0.439F, 1.0F } };

        static constexpr glm::vec3 SKY_BLUE_3 = glm::vec3(102.0F, 178.0F, 255.0F) / COLOR_MAX;
        static constexpr glm::vec4 SKY_BLUE_4 = { 0.4F, 0.698F, 1.0F, 1.0F };
        static constexpr VkClearColorValue SKY_BLUE_V = { { 0.4F, 0.698F, 1.0F, 1.0F } };

        static constexpr glm::vec3 SUNSET_3 = glm::vec3(255.0F, 128.0F, 0.0F) / COLOR_MAX;
        static constexpr glm::vec4 SUNSET_4 = { 1.0F, 0.5F, 0.0F, 1.0F };
        static constexpr VkClearColorValue SUNSET_V = { { 1.0F, 0.5F, 0.0F, 1.0F } };


        static constexpr std::array<std::pair<const char *, glm::vec3>, 20> COLOR_PRESETS_3 = {{
            {"White", WHITE_3},
            {"Black", BLACK_3},
            {"Red", RED_3},
            {"Green", GREEN_3},
            {"Blue", BLUE_3},
            {"Yellow", YELLOW_3},
            {"Cyan", CYAN_3},
            {"Magenta", MAGENTA_3},
            {"Silver", SILVER_3},
            {"Gray", GRAY_3},
            {"Maroon", MAROON_3},
            {"Olive", OLIVE_3},
            {"Lime", LIME_3},
            {"Aqua", AQUA_3},
            {"Teal", TEAL_3},
            {"Navy", NAVY_3},
            {"Fuchsia", FUCHSIA_3},
            {"Night Blue", NIGHT_BLUE_3},
            {"Sky Blue", SKY_BLUE_3},
            {"Sunset", SUNSET_3}
        }};

        static constexpr std::array<std::pair<const char *, glm::vec4>, 20> COLOR_PRESETS_4 = {{
            {"White", WHITE_4},
            {"Black", BLACK_4},
            {"Red", RED_4},
            {"Green", GREEN_4},
            {"Blue", BLUE_4},
            {"Yellow", YELLOW_4},
            {"Cyan", CYAN_4},
            {"Magenta", MAGENTA_4},
            {"Silver", SILVER_4},
            {"Gray", GRAY_4},
            {"Maroon", MAROON_4},
            {"Olive", OLIVE_4},
            {"Lime", LIME_4},
            {"Aqua", AQUA_4},
            {"Teal", TEAL_4},
            {"Navy", NAVY_4},
            {"Fuchsia", FUCHSIA_4},
            {"Night Blue", NIGHT_BLUE_4},
            {"Sky Blue", SKY_BLUE_4},
            {"Sunset", SUNSET_4}
        }};

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

    }; // class Colors

} // namespace ven
```

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

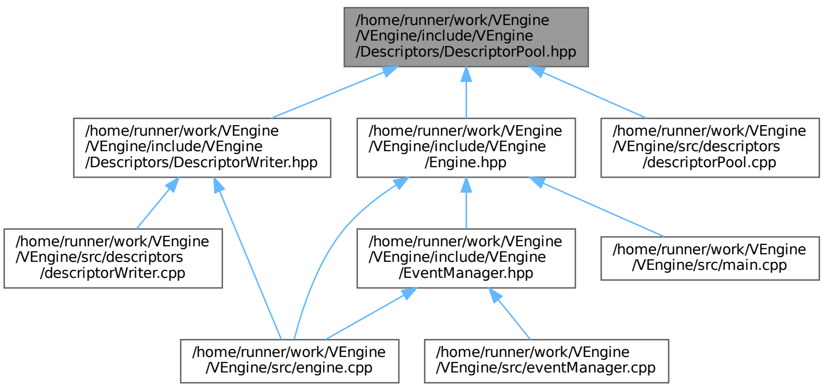
This file contains the DescriptorPool class.

```
#include <memory>
#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

### Variables

- static constexpr uint32_t ven::DEFAULT_MAX_SETS = 1000

### 8.15.1 Detailed Description

This file contains the DescriptorPool class.

Definition in file DescriptorPool.hpp.

## 8.16 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 "VEngine/Device.hpp"

namespace ven {

    static constexpr uint32_t DEFAULT_MAX_SETS = 1000;

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

        public:

            class Builder {

                public:

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

                    [[nodiscard]] std::unique_ptr<DescriptorPool> build() const { return std::make_unique<DescriptorPool>(m_device, m_maxSets, m_poolFlags, m_poolSizes); }

                    Builder &addPoolSize(const VkDescriptorType descriptorType, const uint32_t count) { m_poolSizes.push_back({descriptorType, count}); return *this; }
                    Builder &setPoolFlags(const VkDescriptorPoolCreateFlags flags) { m_poolFlags = flags; return *this; }
                    Builder &setMaxSets(const uint32_t count) { m_maxSets = count; return *this; }

                private:

                    Device &m_device;
                    std::vector<VkDescriptorPoolSize> m_poolSizes;
                    uint32_t m_maxSets{DEFAULT_MAX_SETS};
                    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); }

```

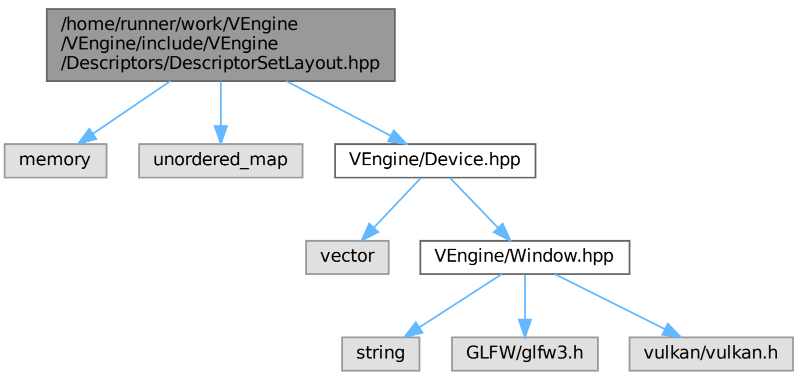
```
            DescriptorPool(const DescriptorPool &) = delete;
            DescriptorPool &operator=(const DescriptorPool &) = delete;

            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
```

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

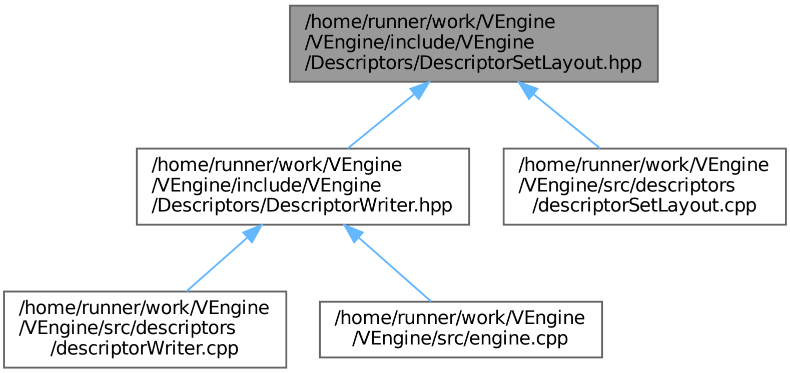
This file contains the DescriptorSetLayout class.

```
#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

### 8.17.1 Detailed Description

This file contains the DescriptorSetLayout class.

Definition in file DescriptorSetLayout.hpp.

## 8.18 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
```

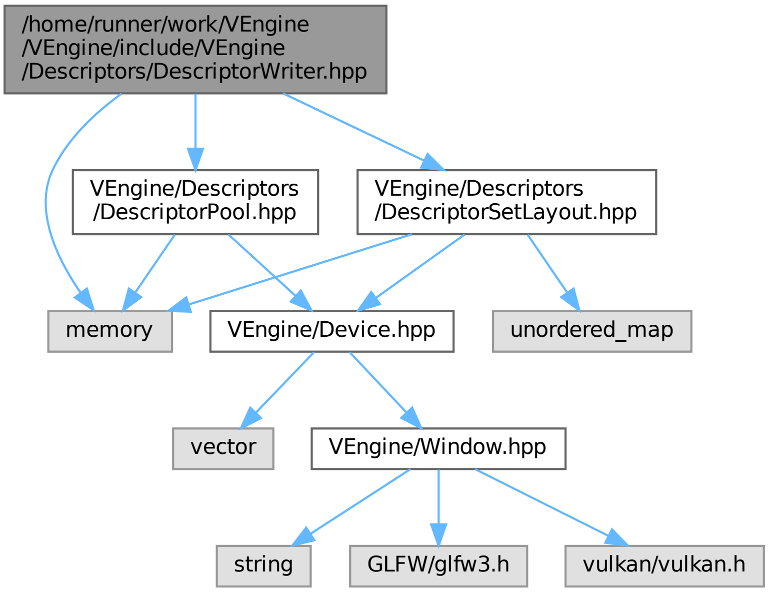
```
    /// @namespace ven
    ///
    class DescriptorSetLayout {

        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
```

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

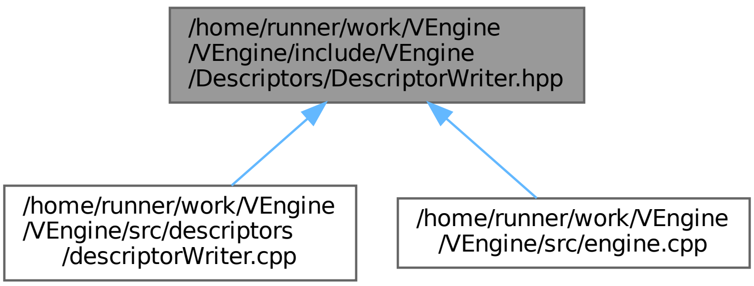
This file contains the DescriptorsWriter class.

```
#include <memory>
#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

### 8.19.1 Detailed Description

This file contains the DescriptorsWriter class.

Definition in file DescriptorWriter.hpp.

## 8.20 DescriptorWriter.hpp

Go to the documentation of this file.

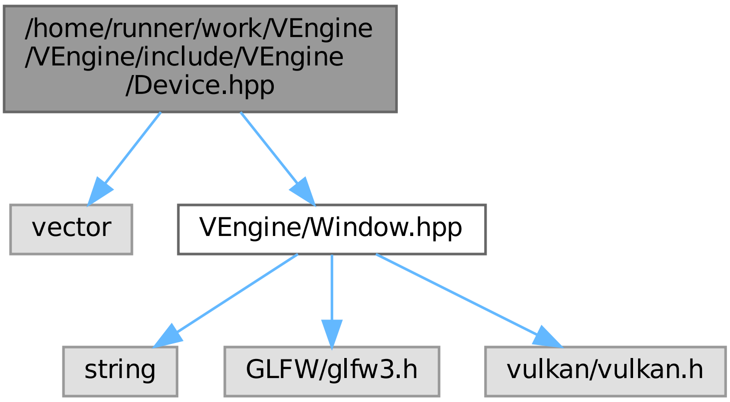
```
///
/// @file DescriptorWriter.hpp
/// @brief This file contains the DescriptorsWriter class
/// @namespace ven
///

#pragma once

#include <memory>

#include "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() = default;

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

            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
```

## 8.21 /home/runner/work/VEngine/VEngine/include/VEngine/Device.hpp File Reference

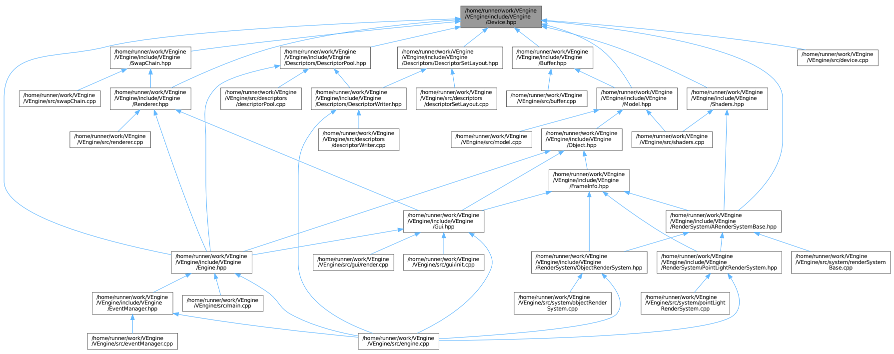
This file contains the Device class.

```
#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

    *Class for device.*

### Namespaces

- namespace ven

## 8.21.1 Detailed Description

This file contains the Device class.

Definition in file Device.hpp.

# 8.22 Device.hpp

Go to the documentation of this file.

```
///
/// @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
    /// @brief Class for device
    /// @namespace ven
    ///
    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;

            [[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 properties) const;
            [[nodiscard]] QueueFamilyIndices findPhysicalQueueFamilies() const { return findQueueFamilies(m_physicalDevice); }
            [[nodiscard]] VkPhysicalDevice getPhysicalDevice() const { return m_physicalDevice; }
            [[nodiscard]] VkQueue getGraphicsQueue() const { return m_graphicsQueue; }
            [[nodiscard]] VkFormat findSupportedFormat(const std::vector<VkFormat> &candidates, VkImageTiling tiling, VkFormatFeatureFlags features) const;

            // Buffer Helper Functions
            void createBuffer(VkDeviceSize size, VkBufferUsageFlags usage, VkMemoryPropertyFlags properties, 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;


        private:

```

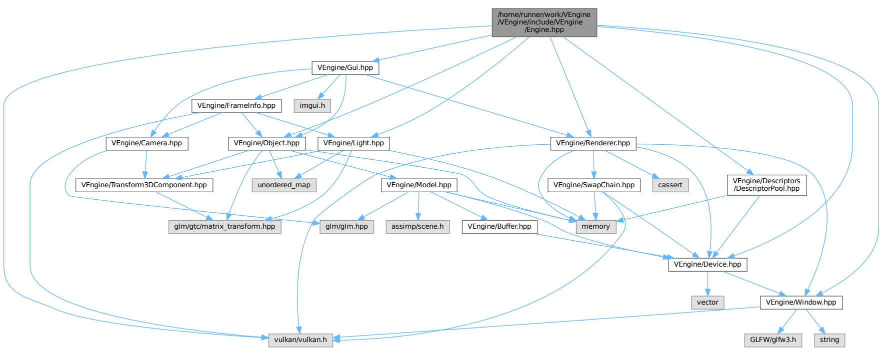
```
            void createInstance();
            void setupDebugMessenger();
            void createSurface() { m_window.createWindowSurface(m_instance, &m_surface); };
            void pickPhysicalDevice();
            void createLogicalDevice();
            void createCommandPool();

            // 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;
            VkPhysicalDeviceProperties m_properties;

            const std::vector<const char *> m_validationLayers = {"VK_LAYER_KHRONOS_validation"};
            const std::vector<const char *> m_deviceExtensions = {VK_KHR_SWAPCHAIN_EXTENSION_NAME};

    }; // class Device

} // namespace ven
```

## 8.23 /home/runner/work/VEngine/VEngine/include/VEngine/Engine.hpp File Reference

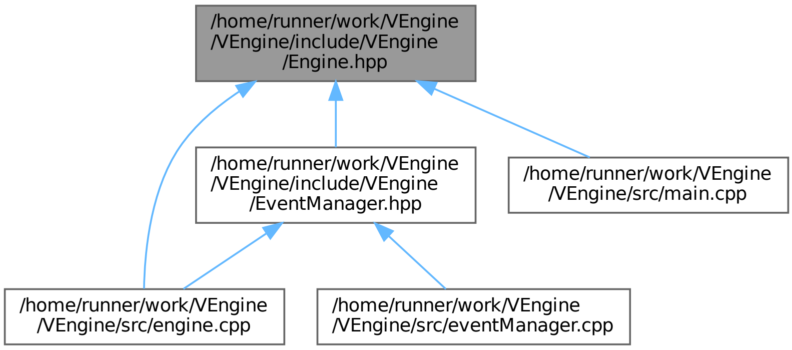
This file contains the Engine class.

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

Include dependency graph for Engine.hpp:



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




### Classes

- class ven::Engine

    *Class for engine.*

### Namespaces

- namespace ven

### Enumerations

- enum ven::ENGINE_STATE : uint8_t { ven::EDITOR = 0 , ven::GAME = 1 , ven::PAUSED = 2 , ven::EXIT = 3 }

## 8.23.1 Detailed Description

This file contains the Engine class.

Definition in file Engine.hpp.

# 8.24 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 "VEngine/Gui.hpp"
#include "VEngine/Window.hpp"
```

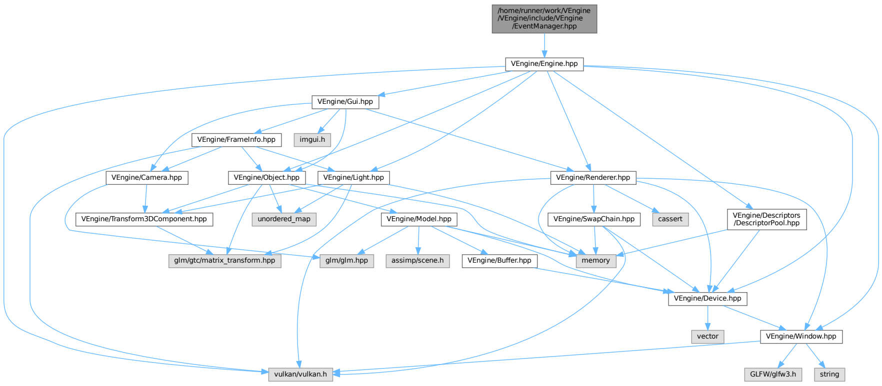
```cpp
#include "VEngine/Device.hpp"
#include "VEngine/Object.hpp"
#include "VEngine/Renderer.hpp"
#include "VEngine/Descriptors/DescriptorPool.hpp"
#include "VEngine/Light.hpp"

namespace ven {

    enum ENGINE_STATE : uint8_t {
        EDITOR = 0,
        GAME = 1,
        PAUSED = 2,
        EXIT = 3
    };

    ///
    /// @class Engine
    /// @brief Class for engine
    /// @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();

            ENGINE_STATE m_state{EXIT};

            Window m_window;
            Device m_device{m_window};
            Renderer m_renderer{m_window, m_device};
            Gui m_gui;
            std::unique_ptr<DescriptorPool> m_globalPool;
            Object::Map m_objects;
            Light::Map m_lights;

            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
```

## 8.25 /home/runner/work/VEngine/VEngine/include/VEngine/EventManager.hpp File Reference

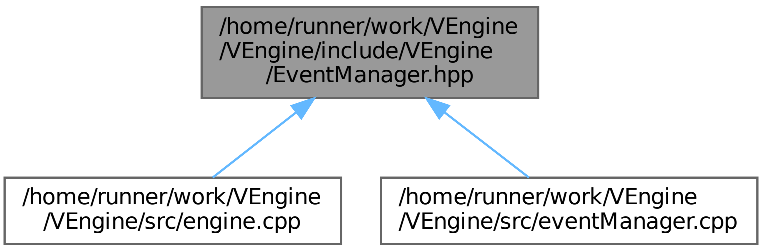
This file contains the EventManager class.

```
#include "VEngine/Engine.hpp"
```

Include dependency graph for EventManager.hpp:




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




### Classes

- struct ven::KeyAction
- struct ven::KeyMappings
- class ven::EventManager

    *Class for event manager.*

### Namespaces

- namespace ven

## 8.25.1 Detailed Description

This file contains the EventManager class.

Definition in file EventManager.hpp.

## 8.26 EventManager.hpp

Go to the documentation of this file.

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

#pragma once

#include "VEngine/Engine.hpp"

namespace ven {

    struct KeyAction {
        uint16_t key;
        glm::vec3* dir;
        glm::vec3 value;
    };

    struct KeyMappings {
        uint16_t moveLeft = GLFW_KEY_A;
        uint16_t moveRight = GLFW_KEY_D;
        uint16_t moveForward = GLFW_KEY_W;
        uint16_t moveBackward = GLFW_KEY_S;
        uint16_t moveUp = GLFW_KEY_SPACE;
        uint16_t moveDown = GLFW_KEY_LEFT_SHIFT;
        uint16_t lookLeft = GLFW_KEY_LEFT;
        uint16_t lookRight = GLFW_KEY_RIGHT;
        uint16_t lookUp = GLFW_KEY_UP;
        uint16_t lookDown = GLFW_KEY_DOWN;
        uint16_t toggleGui = GLFW_KEY_F1;
    };

    ///
    /// @class EventManager
    /// @brief Class for event manager
    /// @namespace ven
    ///
    class EventManager {

        public:

            EventManager() = default;
            ~EventManager() = default;

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

            void handleEvents(GLFWwindow *window, ENGINE_STATE *engineState, Camera& camera, Gui& gui,
    float dt) const;

        private:

            void moveCamera(GLFWwindow* window, Camera& camera, Gui& gui, float dt) const;
            static void updateEngineState(ENGINE_STATE *engineState, const ENGINE_STATE newState) {
    *engineState = newState; }
            static bool isKeyJustPressed(GLFWwindow* window, int key, std::unordered_map<int, bool>&
    keyStates);

            KeyMappings m_keys{};
            mutable std::unordered_map<int, bool> m_keyState;

    }; // class EventManager

} // namespace ven
```

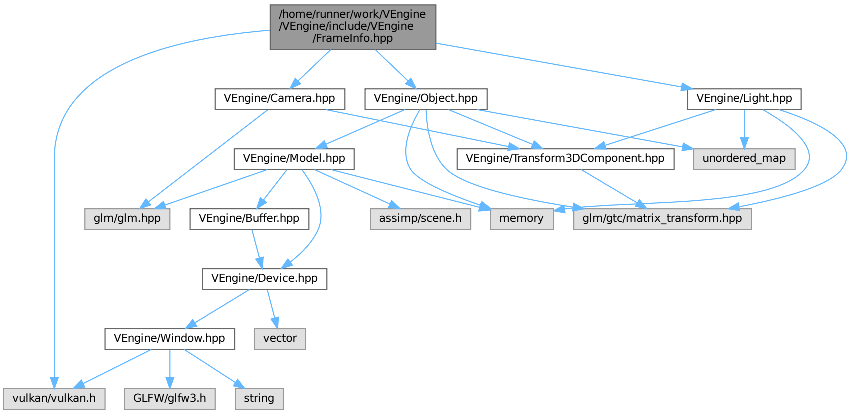
## 8.27 /home/runner/work/VEngine/VEngine/include/VEngine/FrameInfo.hpp File Reference

This file contains the FrameInfo class.

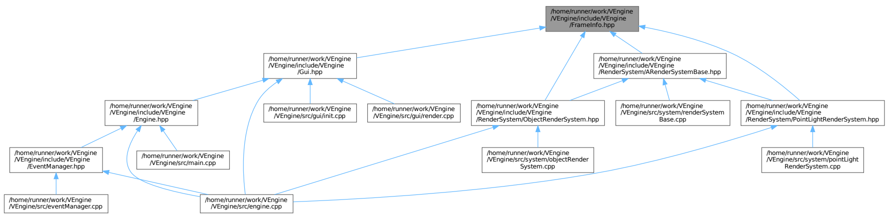
```
#include <vulkan/vulkan.h>
#include "VEngine/Camera.hpp"
#include "VEngine/Object.hpp"
```

```
#include "VEngine/Light.hpp"
```

Include dependency graph for FrameInfo.hpp:




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




## Classes

- struct ven::PointLightData
- struct ven::GlobalUbo
- struct ven::FrameInfo

## Namespaces

- namespace ven

## Variables

- static constexpr uint16_t ven::MAX_LIGHTS = 10
- static constexpr float ven::DEFAULT_AMBIENT_LIGHT_INTENSITY = .2F
- static constexpr glm::vec4 ven::DEFAULT_AMBIENT_LIGHT_COLOR = {glm::vec3(1.F), DEFAULT_AMBIENT_LIGHT_INTENS

### 8.27.1 Detailed Description

This file contains the FrameInfo class.

Definition in file FrameInfo.hpp.

## 8.28 FrameInfo.hpp

Go to the documentation of this file.

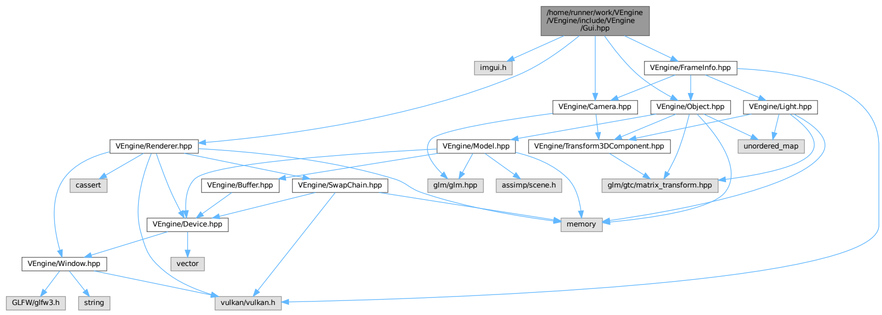
```
///
/// @file FrameInfo.hpp
/// @brief This file contains the FrameInfo class
/// @namespace ven
///

#pragma once

#include <vulkan/vulkan.h>

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

namespace ven {

static constexpr uint16_t MAX_LIGHTS = 10;

static constexpr float DEFAULT_AMBIENT_LIGHT_INTENSITY = .2F;
static constexpr glm::vec4 DEFAULT_AMBIENT_LIGHT_COLOR = {glm::vec3(1.F),
DEFAULT_AMBIENT_LIGHT_INTENSITY};

    struct PointLightData
    {
        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{DEFAULT_AMBIENT_LIGHT_COLOR};
        std::array<PointLightData, MAX_LIGHTS> pointLights;
        uint16_t numLights;
    };

    struct FrameInfo
    {
        unsigned long frameIndex;
        float frameTime;
        VkCommandBuffer commandBuffer;
        Camera &camera;
        VkDescriptorSet globalDescriptorSet;
        Object::Map &objects;
        Light::Map &lights;
    };

} // namespace ven
```

## 8.29 /home/runner/work/VEngine/VEngine/include/VEngine/Gui.hpp File Reference

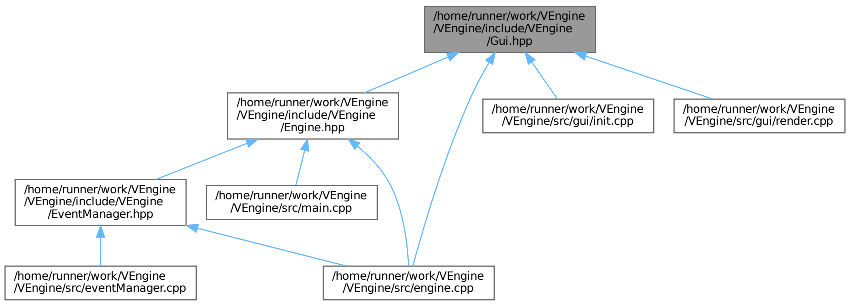
This file contains the ImGuiWindowManager class.

```
#include <imgui.h>
#include "VEngine/Object.hpp"
#include "VEngine/Renderer.hpp"
#include "VEngine/Camera.hpp"
#include "VEngine/FrameInfo.hpp"
```

Include dependency graph for Gui.hpp:




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




### Classes

- class ven::Gui

  *Class for Gui.*
- struct ven::Gui::funcs

### Namespaces

- namespace ven

### Enumerations

- enum ven::GUI_STATE : uint8_t { ven::VISIBLE = 0 , ven::HIDDEN = 1 }

## 8.29.1 Detailed Description

This file contains the ImGuiWindowManager class.

Definition in file Gui.hpp.

## 8.30 Gui.hpp

[Go to the documentation of this file.]

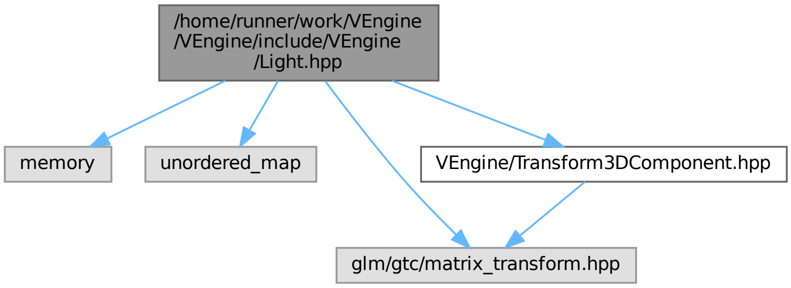
```cpp
///
/// @file Gui.hpp
/// @brief This file contains the ImGuiWindowManager class
/// @namespace ven
///

#pragma once

#include <imgui.h>

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

namespace ven {

    enum GUI_STATE : uint8_t {
        VISIBLE = 0,
        HIDDEN = 1
    };

    ///
    /// @class Gui
    /// @brief Class for Gui
    /// @namespace ven
    ///
    class Gui {

        public:

            Gui() = default;
            ~Gui() = default;

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

            static void init(GLFWwindow* window, VkInstance instance, const Device* device, VkRenderPass renderPass);

            static void render(Renderer *renderer, std::unordered_map<unsigned int, Object>& objects, std::unordered_map<unsigned int, Light>& lights, Camera& camera, VkPhysicalDevice physicalDevice, GlobalUbo& ubo);
            static void cleanup();

            void setState(const GUI_STATE state) { m_state = state; }
            [[nodiscard]] GUI_STATE getState() const { return m_state; }

        private:

            static void initStyle();
            static void renderFrameWindow();
            static void cameraSection(Camera& camera);
            static void inputsSection(const ImGuiIO *io);
            static void rendererSection(Renderer *renderer, GlobalUbo& ubo);
            static void devicePropertiesSection(VkPhysicalDeviceProperties deviceProperties);
            static void objectsSection(std::unordered_map<unsigned int, Object>& objects);
            static void lightsSection(std::unordered_map<unsigned int, Light>& lights);


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

            static ImGuiIO* m_io;
            GUI_STATE m_state{VISIBLE};

    }; // class Gui

} // namespace ven
```

## 8.31 /home/runner/work/VEngine/VEngine/include/VEngine/Light.hpp File Reference

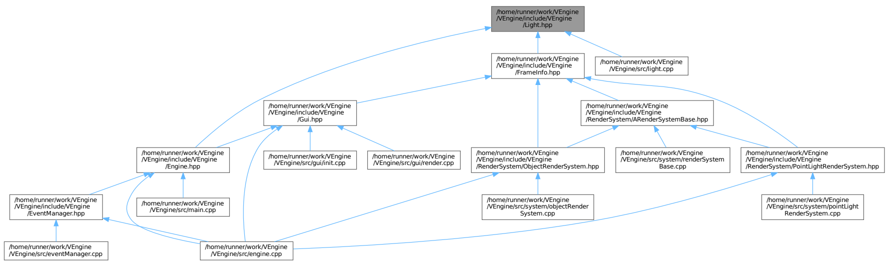
This file contains the Light class.

```
#include <memory>
#include <unordered_map>
#include <glm/gtc/matrix_transform.hpp>
#include "VEngine/Transform3DComponent.hpp"
```

Include dependency graph for Light.hpp:




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



### Classes

- class ven::Light

    *Class for light.*

### Namespaces

- namespace ven

### Variables

- static constexpr float ven::DEFAULT_LIGHT_INTENSITY = .2F
- static constexpr float ven::DEFAULT_LIGHT_RADIUS = 0.1F
- static constexpr glm::vec4 ven::DEFAULT_LIGHT_COLOR = {glm::vec3(1.F), DEFAULT_LIGHT_INTENSITY}

## 8.31.1 Detailed Description

This file contains the Light class.

Definition in file Light.hpp.

## 8.32 Light.hpp

Go to the documentation of this file.

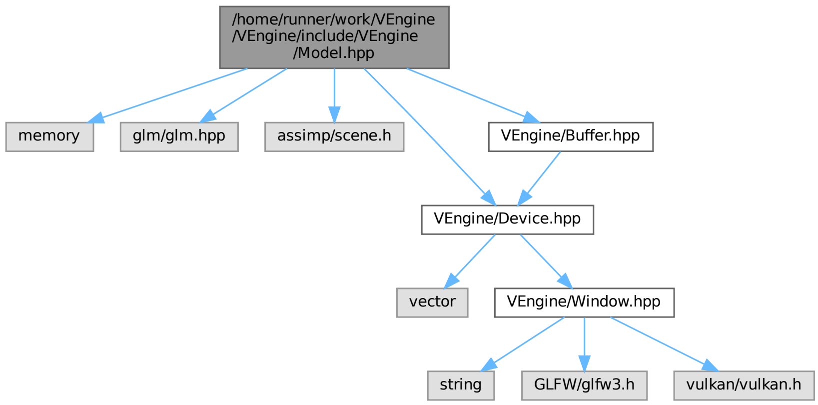
```cpp
///
/// @file Light.hpp
/// @brief This file contains the Light class
/// @namespace ven
///

#pragma once

#include <memory>
#include <unordered_map>

#include <glm/gtc/matrix_transform.hpp>

#include "VEngine/Transform3DComponent.hpp"

namespace ven {

    static constexpr float DEFAULT_LIGHT_INTENSITY = .2F;
    static constexpr float DEFAULT_LIGHT_RADIUS = 0.1F;
    static constexpr glm::vec4 DEFAULT_LIGHT_COLOR = {glm::vec3(1.F), DEFAULT_LIGHT_INTENSITY};

    ///
    /// @class Light
    /// @brief Class for light
    /// @namespace ven
    ///
    class Light {

        public:

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

            ~Light() = default;

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

            static Light createLight(float radius = DEFAULT_LIGHT_RADIUS, glm::vec4 color =
DEFAULT_LIGHT_COLOR);

            glm::vec4 color{DEFAULT_LIGHT_COLOR};
            Transform3DComponent transform3D{};

            [[nodiscard]] unsigned int getId() const { return m_lightId; }
            [[nodiscard]] std::string getName() const { return m_name; }

            void setName(const std::string &name) { m_name = name; }

        private:

            explicit Light(const unsigned int lightId) : m_lightId(lightId) {}

            unsigned int m_lightId;
            std::string m_name{"point light"};

    }; // class Light

} // namespace ven
```

## 8.33 /home/runner/work/VEngine/VEngine/include/VEngine/Model.hpp File Reference

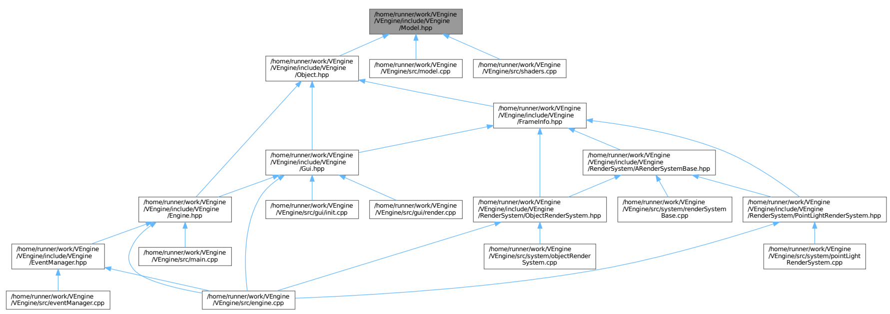
This file contains the Model class.

```cpp
#include <memory>
#include <glm/glm.hpp>
#include <assimp/scene.h>
#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

## 8.33.1 Detailed Description

This file contains the Model class.

Definition in file Model.hpp.

## 8.34 Model.hpp

Go to the documentation of this file.

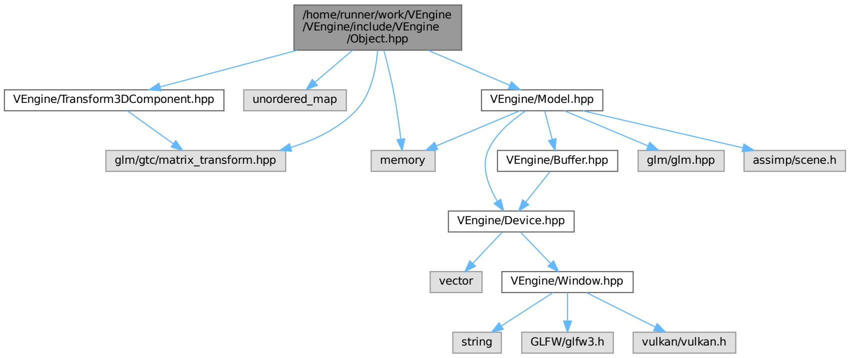
```cpp
///
/// @file Model.hpp
/// @brief This file contains the Model class
/// @namespace ven
///

#pragma once

#include <memory>

#include <glm/glm.hpp>

#include <assimp/scene.h>

#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);
                void processNode(const aiNode* node, const aiScene* scene);
                void processMesh(const aiMesh* mesh, const aiScene* scene);
            };

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

            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
```

## 8.35 /home/runner/work/VEngine/VEngine/include/VEngine/Object.hpp File Reference

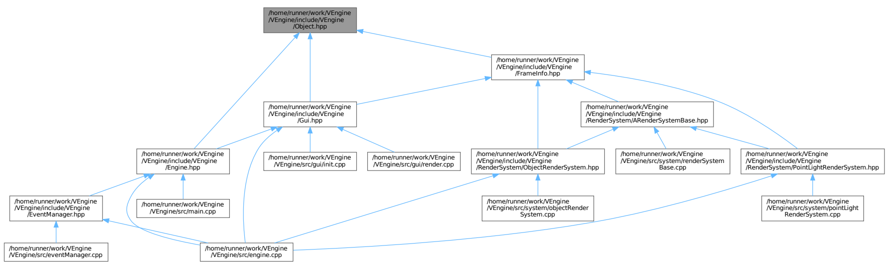
This file contains the Object class.

```
#include <memory>
#include <unordered_map>
#include <glm/gtc/matrix_transform.hpp>
#include "VEngine/Model.hpp"
#include "VEngine/Transform3DComponent.hpp"
```

Include dependency graph for Object.hpp:




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



### Classes

- class ven::Object

  *Class for object.*

### Namespaces

- namespace ven

### 8.35.1 Detailed Description

This file contains the Object class.

Definition in file Object.hpp.

## 8.36 Object.hpp

Go to the documentation of this file.

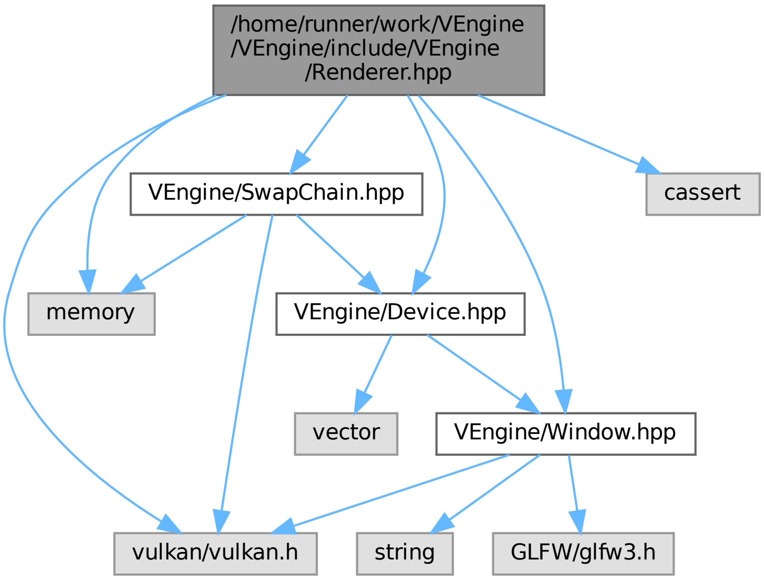
```
///
/// @file Object.hpp
/// @brief This file contains the Object class
/// @namespace ven
///

#pragma once

#include <memory>
#include <unordered_map>

#include <glm/gtc/matrix_transform.hpp>

#include "VEngine/Model.hpp"
#include "VEngine/Transform3DComponent.hpp"

namespace ven {

    ///
    /// @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++); }

            [[nodiscard]] unsigned int getId() const { return m_objId; }
            [[nodiscard]] std::string getName() const { return m_name; }
            [[nodiscard]] std::shared_ptr<Model> getModel() const { return m_model; }

            void setName(const std::string &name) { m_name = name; }
            void setModel(const std::shared_ptr<Model> &model) { m_model = model; }

            Transform3DComponent transform3D{};

        private:

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

            unsigned int m_objId;
            std::string m_name;
            std::shared_ptr<Model> m_model;

    }; // class Object

} // namespace ven
```

## 8.37 /home/runner/work/VEngine/VEngine/include/VEngine/Renderer.hpp File Reference

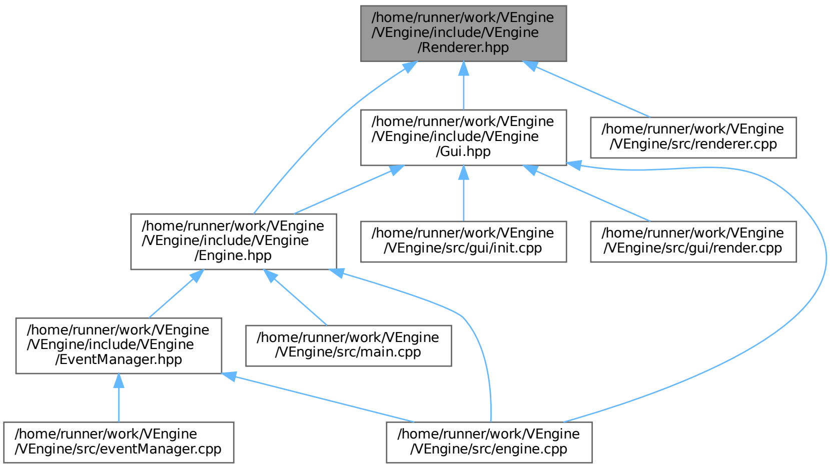
This file contains the Renderer class.

```
#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

*Class for 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}

### 8.37.1 Detailed Description

This file contains the Renderer class.

Definition in file Renderer.hpp.

## 8.38 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
    /// @brief Class for renderer
    /// @namespace ven
    ///
    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)]; }

```

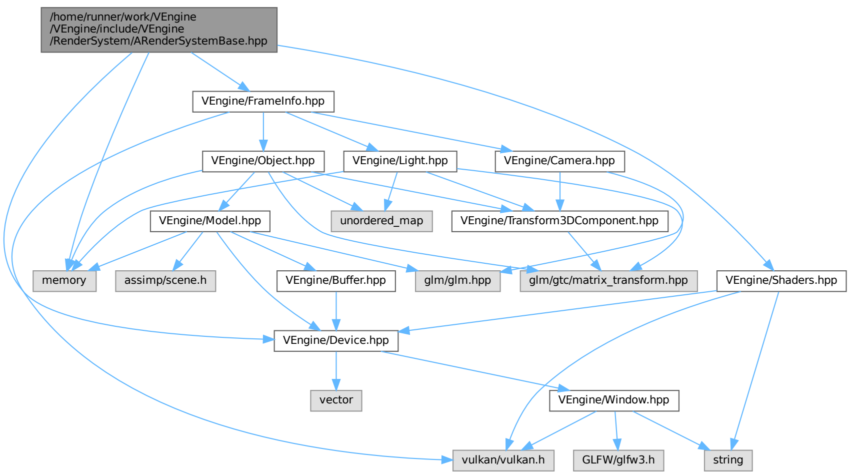
```
            [[nodiscard]] unsigned long getFrameIndex() const { assert(isFrameInProgress() && "cannot get frame index when frame not in progress"); return m_currentFrameIndex; }
            [[nodiscard]] std::array<float, 4> getClearColor() const { return {
                m_clearValues[0].color.float32[0],
                m_clearValues[0].color.float32[1],
                m_clearValues[0].color.float32[2],
                m_clearValues[0].color.float32[3]
            };}

            [[nodiscard]] Window& getWindow() const { return m_window; }

            void setClearValue(const VkClearColorValue clearColorValue = DEFAULT_CLEAR_COLOR, const VkClearDepthStencilValue clearDepthValue = DEFAULT_CLEAR_DEPTH) { m_clearValues[0].color = clearColorValue; m_clearValues[1].depthStencil = clearDepthValue; }
            VkCommandBuffer beginFrame();
            void endFrame();
            void beginSwapChainRenderPass(VkCommandBuffer commandBuffer) const;
            void endSwapChainRenderPass(VkCommandBuffer commandBuffer) const;

        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{DEFAULT_CLEAR_COLOR, 1.0F, 0.F};

            uint32_t m_currentImageIndex{0};
            unsigned long m_currentFrameIndex{0};
            bool m_isFrameStarted{false};

    }; // class Renderer

} // namespace ven
```

## 8.39 /home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/ARenderSystemBase.hpp File Reference

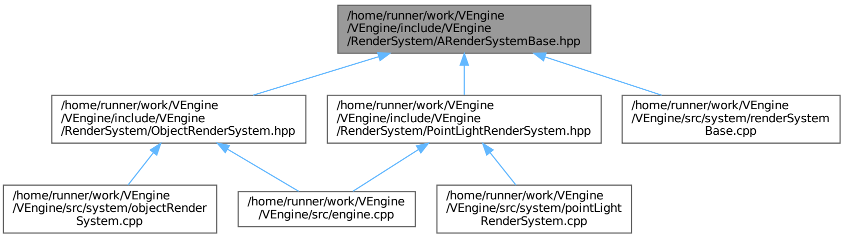
This file contains the ARenderSystemBase class.

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

Include dependency graph for ARenderSystemBase.hpp:



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




### Classes

- class ven::ARenderSystemBase

  *Abstract class for render system base.*

### Namespaces

- namespace ven

### 8.39.1 Detailed Description

This file contains the ARenderSystemBase class.

Definition in file ARenderSystemBase.hpp.

## 8.40 ARenderSystemBase.hpp

Go to the documentation of this file.

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

#pragma once

#include <memory>

#include "VEngine/Device.hpp"
#include "VEngine/Shaders.hpp"
#include "VEngine/FrameInfo.hpp"

namespace ven {

    ///
    /// @class ARenderSystemBase
    /// @brief Abstract class for render system base
    /// @namespace ven
    ///
    class ARenderSystemBase {

        public:

            explicit ARenderSystemBase(Device& device) : m_device{device} {}
            virtual ~ARenderSystemBase() { vkDestroyPipelineLayout(m_device.device(),
    m_pipelineLayout, nullptr); }

```

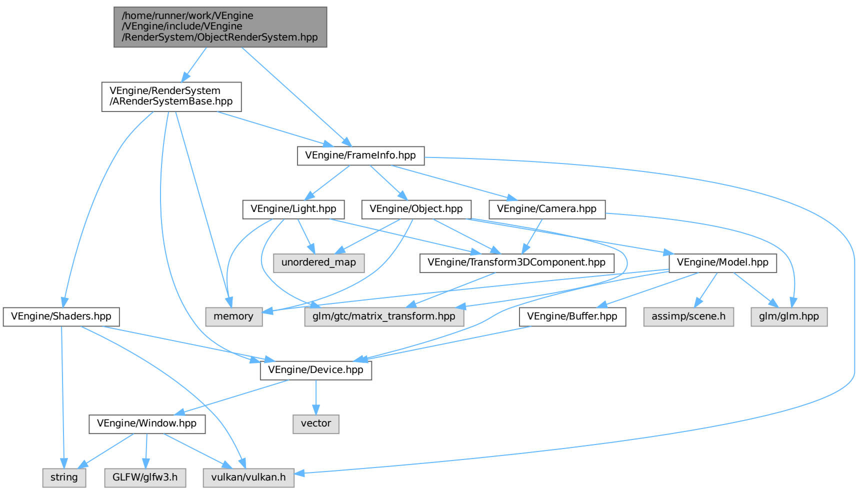
```
        virtual void render(const FrameInfo &frameInfo) const = 0;

    protected:

        void createPipelineLayout(VkDescriptorSetLayout globalSetLayout, uint32_t
 pushConstantSize);
        void createPipeline(VkRenderPass renderPass, const std::string &shadersVertPath, const
 std::string &shadersFragPath, bool isLight);

        [[nodiscard]] Device& getDevice() const { return m_device; }
        [[nodiscard]] VkPipelineLayout getPipelineLayout() const { return m_pipelineLayout; }
        [[nodiscard]] const std::unique_ptr<Shaders>& getShaders() const { return m_shaders; }

    private:

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

    }; // class ARenderSystemBase

} // namespace ven
```

## 8.41 /home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/ObjectRenderSystem.hpp File Reference

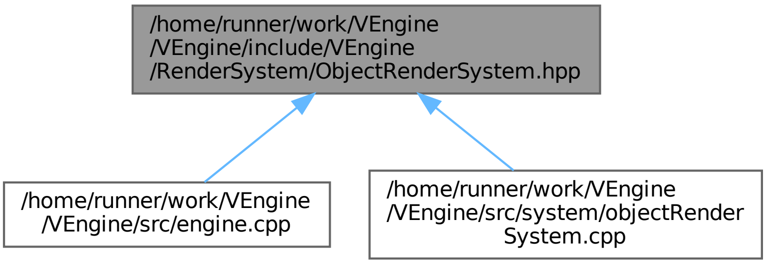
This file contains the ObjectRenderSystem class.

```
#include "VEngine/FrameInfo.hpp"
#include "VEngine/RenderSystem/ARenderSystemBase.hpp"
```

Include dependency graph for ObjectRenderSystem.hpp:



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




### Classes

- struct ven::ObjectPushConstantData
- class ven::ObjectRenderSystem

  *Class for object render system.*

### Namespaces

- namespace ven

### 8.41.1 Detailed Description

This file contains the ObjectRenderSystem class.

Definition in file ObjectRenderSystem.hpp.

## 8.42 ObjectRenderSystem.hpp

Go to the documentation of this file.

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

#pragma once

#include "VEngine/FrameInfo.hpp"
#include "VEngine/RenderSystem/ARenderSystemBase.hpp"

namespace ven {

    struct ObjectPushConstantData {
        glm::mat4 modelMatrix{};
        glm::mat4 normalMatrix{};
    };

    ///
    /// @class ObjectRenderSystem
    /// @brief Class for object render system
    /// @namespace ven
```

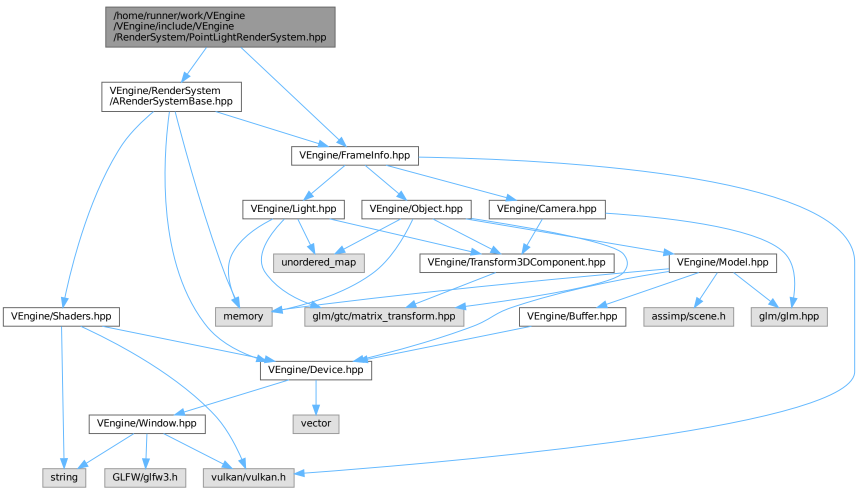
```
    ///
    class ObjectRenderSystem final : public ARenderSystemBase {

        public:

            explicit ObjectRenderSystem(Device& device, const VkRenderPass renderPass, const VkDescriptorSetLayout globalSetLayout) : ARenderSystemBase(device) {
                createPipelineLayout(globalSetLayout, sizeof(ObjectPushConstantData));
                createPipeline(renderPass, std::string(SHADERS_BIN_PATH) + "vertex_shader.spv", std::string(SHADERS_BIN_PATH) + "fragment_shader.spv", false);
            }

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

            void render(const FrameInfo &frameInfo) const override;

    }; // class ObjectRenderSystem

} // namespace ven
```

## 8.43 /home/runner/work/VEngine/VEngine/include/VEngine/RenderSystem/PointLightRenderSystem.hpp File Reference

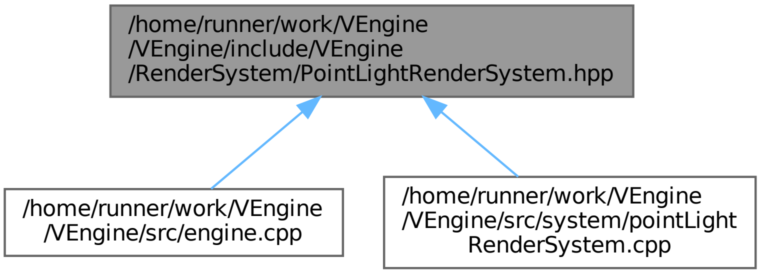
This file contains the PointLightRenderSystem class.

```
#include "VEngine/RenderSystem/ARenderSystemBase.hpp"
#include "VEngine/FrameInfo.hpp"
```

Include dependency graph for PointLightRenderSystem.hpp:



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




### Classes

- struct ven::LightPushConstantData
- class ven::PointLightRenderSystem

  *Class for point light system.*

### Namespaces

- namespace ven

### 8.43.1 Detailed Description

This file contains the PointLightRenderSystem class.

Definition in file PointLightRenderSystem.hpp.

## 8.44 PointLightRenderSystem.hpp

Go to the documentation of this file.

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

#pragma once

#include "VEngine/RenderSystem/ARenderSystemBase.hpp"
#include "VEngine/FrameInfo.hpp"

namespace ven {

    struct LightPushConstantData {
        glm::vec4 position{};
        glm::vec4 color{};
        float radius;
    };

    ///
    /// @class PointLightRenderSystem
    /// @brief Class for point light system
```

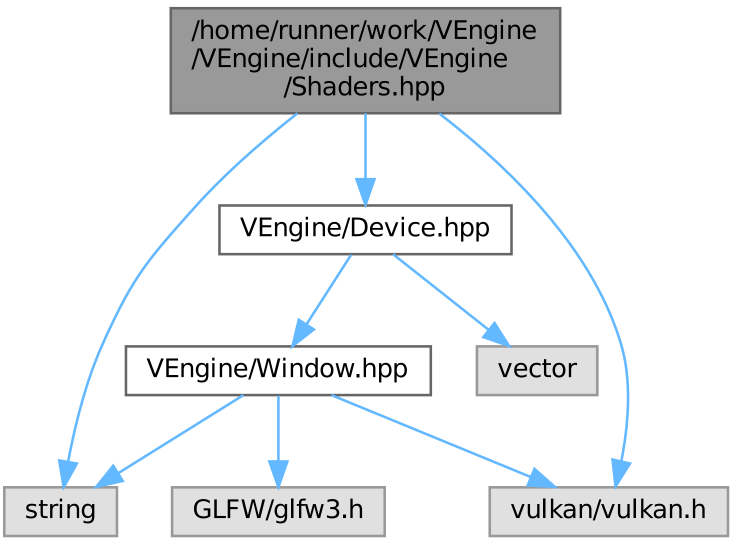
```
    /// @namespace ven
    ///
    class PointLightRenderSystem final : public ARenderSystemBase {

        public:

            explicit PointLightRenderSystem(Device& device, const VkRenderPass renderPass, const
VkDescriptorSetLayout globalSetLayout) : ARenderSystemBase(device) {
                createPipelineLayout(globalSetLayout, sizeof(LightPushConstantData));
                createPipeline(renderPass, std::string(SHADERS_BIN_PATH) + "vertex_point_light.spv",
std::string(SHADERS_BIN_PATH) + "fragment_point_light.spv", true);
            }

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

            void render(const FrameInfo &frameInfo) const override;

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

    }; // class PointLightRenderSystem

} // namespace ven
```

## 8.45 /home/runner/work/VEngine/VEngine/include/VEngine/Shaders.hpp File Reference

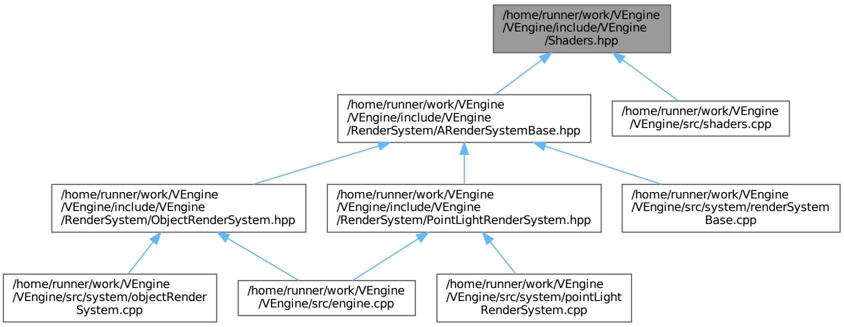
This file contains the Shader class.

```
#include <string>
#include <vulkan/vulkan.h>
#include "VEngine/Device.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 = "build/shaders/"

### 8.45.1 Detailed Description

This file contains the Shader class.

Definition in file Shaders.hpp.

## 8.46 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 "VEngine/Device.hpp"

namespace ven {
```

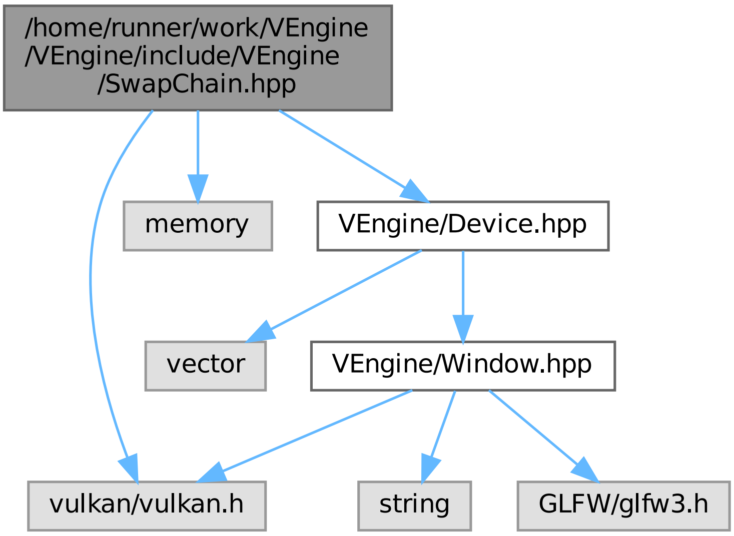
```
    static constexpr std::string_view SHADERS_BIN_PATH = "build/shaders/";

    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
```

## 8.47 /home/runner/work/VEngine/VEngine/include/VEngine/SwapChain.hpp File Reference

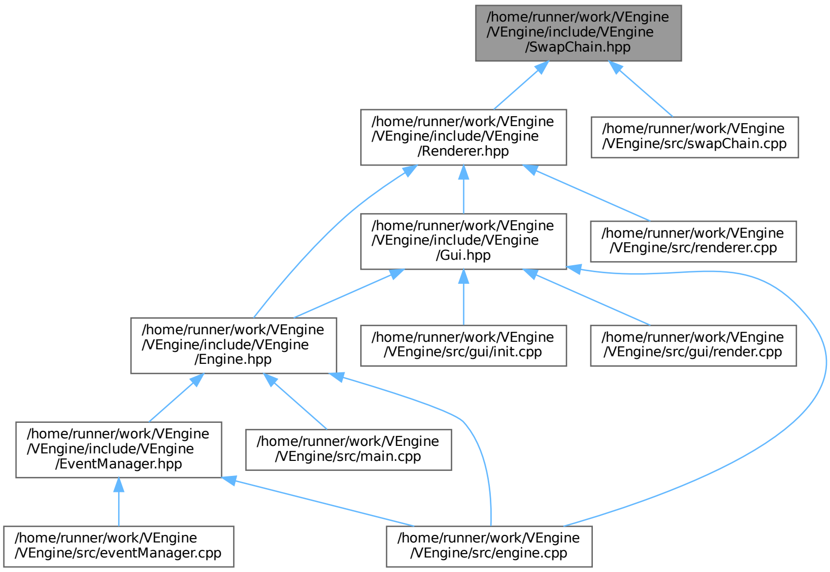
This file contains the Shader class.

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

Include dependency graph for SwapChain.hpp:




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




## Classes

- class ven::SwapChain

    *Class for swap chain.*

### Namespaces

- namespace ven

### Variables

- static constexpr int ven::MAX_FRAMES_IN_FLIGHT = 2

## 8.47.1 Detailed Description

This file contains the Shader class.

Definition in file SwapChain.hpp.

# 8.48 SwapChain.hpp

Go to the documentation of this file.

```
///
/// @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 {

    static constexpr int MAX_FRAMES_IN_FLIGHT = 2;

    ///
    /// @class SwapChain
    /// @brief Class for swap chain
    /// @namespace ven
    ///
    class SwapChain {

        public:

            SwapChain(Device &deviceRef, const VkExtent2D windowExtentRef) : m_device{deviceRef}, m_windowExtent{windowExtentRef} { init(); }
            SwapChain(Device &deviceRef, const VkExtent2D windowExtentRef, std::shared_ptr<SwapChain> previous) : m_device{deviceRef}, m_windowExtent{windowExtentRef}, m_oldSwapChain{std::move(previous)} { init(); m_oldSwapChain = nullptr; }
            ~SwapChain();

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

            [[nodiscard]] VkFramebuffer getFrameBuffer(const unsigned long index) const { return m_swapChainFrameBuffers[index]; }
            [[nodiscard]] VkRenderPass getRenderPass() const { return m_renderPass; }
            [[nodiscard]] VkImageView getImageView(const int index) const { return m_swapChainImageViews[static_cast<unsigned long>(index)]; }
            [[nodiscard]] size_t imageCount() const { return m_swapChainImages.size(); }
            [[nodiscard]] VkFormat getSwapChainImageFormat() const { return m_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);

```

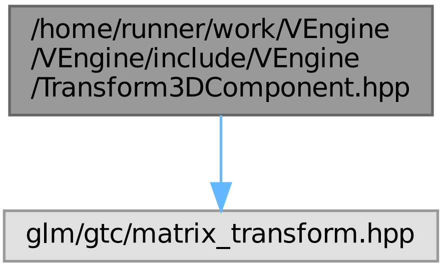
```cpp
        [[nodiscard]] bool compareSwapFormats(const SwapChain &swapChain) const { return m_swapChainImageFormat == swapChain.m_swapChainImageFormat && m_swapChainDepthFormat == swapChain.m_swapChainDepthFormat; }

    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 m_swapChainImageFormat{};
        VkFormat m_swapChainDepthFormat{};
        VkExtent2D m_swapChainExtent{};

        std::vector<VkFramebuffer> m_swapChainFrameBuffers;
        VkRenderPass m_renderPass{};

        std::vector<VkImage> m_depthImages;
        std::vector<VkDeviceMemory> m_depthImageMemory;
        std::vector<VkImageView> m_depthImageViews;
        std::vector<VkImage> m_swapChainImages;
        std::vector<VkImageView> m_swapChainImageViews;

        Device &m_device;
        VkExtent2D m_windowExtent;

        VkSwapchainKHR m_swapChain{};
        std::shared_ptr<SwapChain> m_oldSwapChain;

        std::vector<VkSemaphore> m_imageAvailableSemaphores;
        std::vector<VkSemaphore> m_renderFinishedSemaphores;
        std::vector<VkFence> m_inFlightFences;
        std::vector<VkFence> m_imagesInFlight;
        size_t m_currentFrame{0};

    }; // class SwapChain

} // namespace ven
```

## 8.49 /home/runner/work/VEngine/VEngine/include/VEngine/Transform3DComponent.hpp File Reference

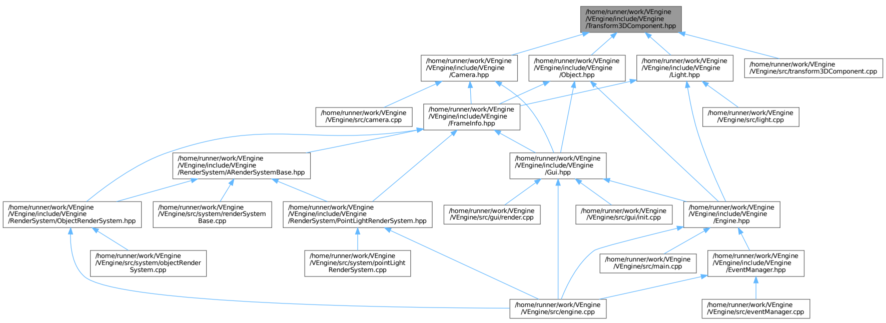
This file contains the Transform3DComponent class.

```
#include <glm/gtc/matrix_transform.hpp>
```

Include dependency graph for Transform3DComponent.hpp:




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



### Classes

- class ven::Transform3DComponent

    *Class for 3D transformation.*

### Namespaces

- namespace ven

## 8.49.1 Detailed Description

This file contains the Transform3DComponent class.

Definition in file Transform3DComponent.hpp.

## 8.50 Transform3DComponent.hpp

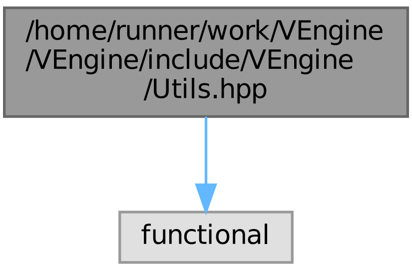
Go to the documentation of this file.

```
///
/// @file Transform3DComponent.hpp
/// @brief This file contains the Transform3DComponent class
/// @namespace ven
///

#pragma once

#include <glm/gtc/matrix_transform.hpp>

namespace ven {

    ///
    /// @class Transform3DComponent
    /// @brief Class for 3D transformation
    /// @namespace ven
    ///
    class Transform3DComponent {

        public:

            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;

    }; // class Transform3DComponent

} // namespace ven
```

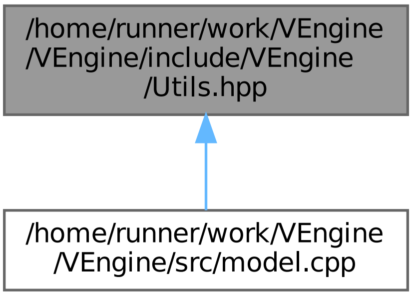
## 8.51 /home/runner/work/VEngine/VEngine/include/VEngine/Utils.hpp File Reference

#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)

## 8.52 Utils.hpp

Go to the documentation of this file.

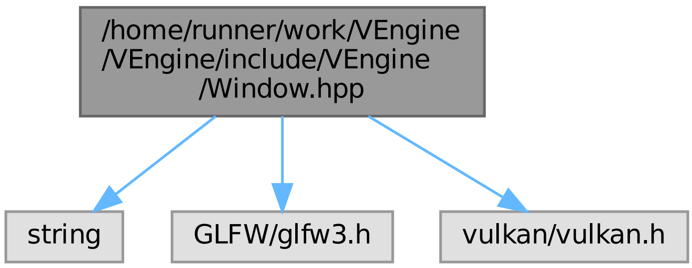
```
///
/// @file Utils.hpp
/// @brief
/// @namespace ven
///

#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
```

## 8.53 /home/runner/work/VEngine/VEngine/include/VEngine/Window.hpp File Reference

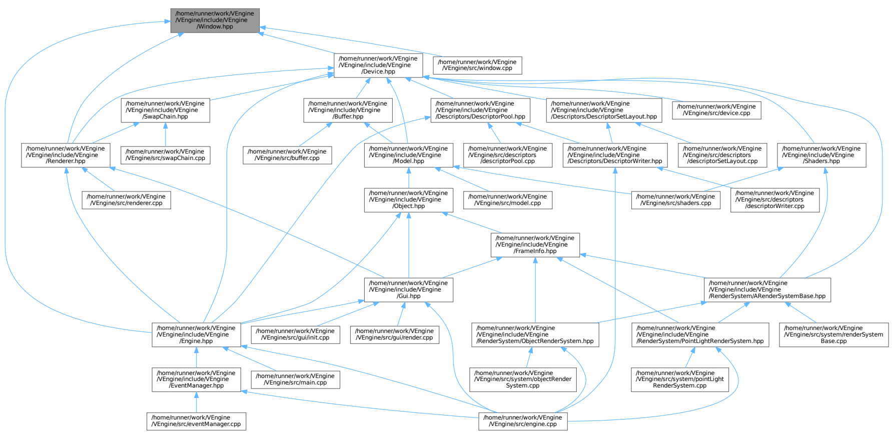
This file contains the Window class.

```
#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"

## 8.53.1 Detailed Description

This file contains the Window class.

Definition in file Window.hpp.

## 8.53.2 Macro Definition Documentation

### 8.53.2.1 GLFW_INCLUDE_VULKAN

```
#define GLFW_INCLUDE_VULKAN
```

Definition at line 11 of file Window.hpp.

# 8.54 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             explicit Window(const uint32_t width = DEFAULT_WIDTH, const uint32_t height =
      DEFAULT_HEIGHT, const std::string &title = DEFAULT_TITLE.data()) : m_window(createWindow(width,
      height, title)), m_width(width), m_height(height) {}
00031             ~Window() { glfwDestroyWindow(m_window); glfwTerminate(); m_window = nullptr;};
00032
00033             Window(const Window&) = delete;
00034             Window& operator=(const Window&) = delete;
00035
00036             [[nodiscard]] GLFWwindow* createWindow(uint32_t width, uint32_t height, const std::string
      &title);
00037             void createWindowSurface(VkInstance instance, VkSurfaceKHR* surface) const;
00038
00039             [[nodiscard]] GLFWwindow* getGLFWindow() const { return m_window; }
00040
```

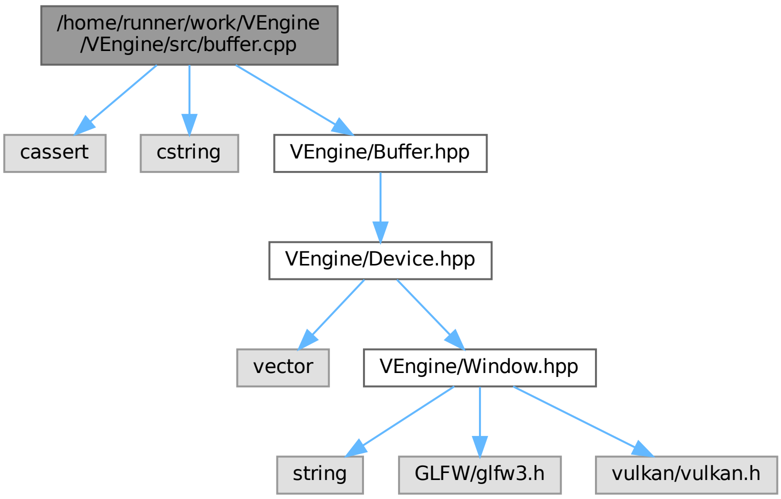
```
            [[nodiscard]] VkExtent2D getExtent() const { return {m_width, m_height}; }
            [[nodiscard]] bool wasWindowResized() const { return m_framebufferResized; }
            void resetWindowResizedFlag() { m_framebufferResized = false; }

            void setFullscreen(bool fullscreen, uint32_t width, uint32_t height);

        private:

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

            GLFWwindow* m_window{nullptr};
            uint32_t m_width{DEFAULT_WIDTH};
            uint32_t m_height{DEFAULT_HEIGHT};

            bool m_framebufferResized = false;

    }; // class Window

} // namespace ven
```

## 8.55 /home/runner/work/VEngine/VEngine/README.md File Reference

## 8.56 /home/runner/work/VEngine/VEngine/src/buffer.cpp File Reference

```
#include <cassert>
#include <cstring>
#include "VEngine/Buffer.hpp"
```

Include dependency graph for buffer.cpp:




## 8.57 buffer.cpp

Go to the documentation of this file.

```
#include <cassert>
#include <cstring>

#include "VEngine/Buffer.hpp"

```

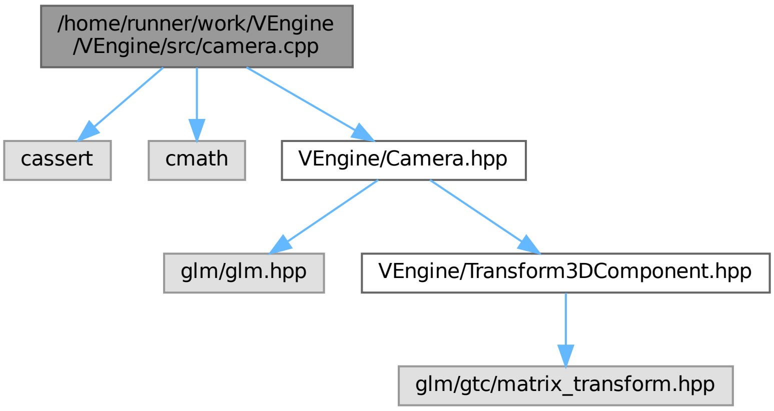
```
VkDeviceSize ven::Buffer::getAlignment(const VkDeviceSize instanceSize, const VkDeviceSize
minOffsetAlignment) {
    if (minOffsetAlignment > 0) {
        return (instanceSize + minOffsetAlignment - 1) & ~(minOffsetAlignment - 1);
    }
    return instanceSize;
}

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);
}
```

## 8.58 /home/runner/work/VEngine/VEngine/src/camera.cpp File Reference

```
#include <cassert>
#include <cmath>
```

`#include "VEngine/Camera.hpp"`

Include dependency graph for camera.cpp:




## 8.59 camera.cpp

Go to the documentation of this file.

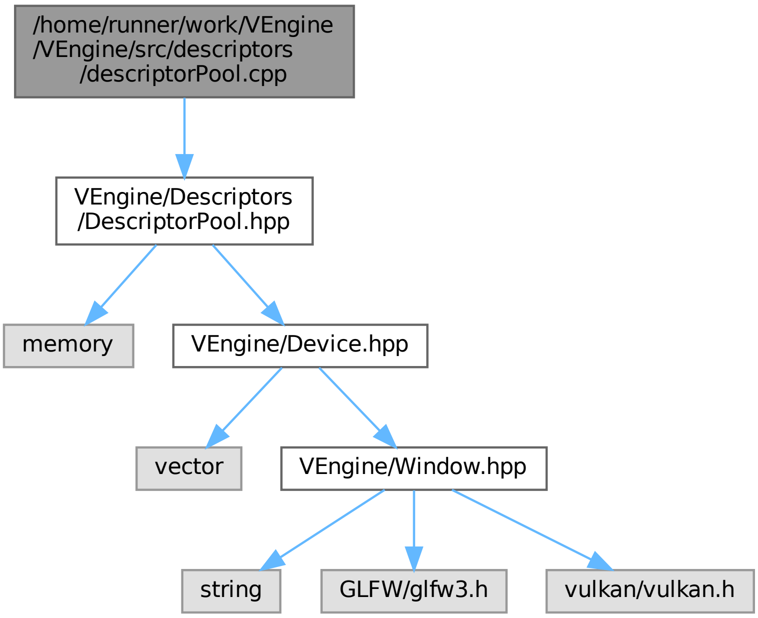
```
#include <cassert>
#include <cmath>

#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::setViewXYZ(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;
}
```

## 8.60 /home/runner/work/VEngine/VEngine/src/descriptors/descriptor↩ Pool.cpp File Reference

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

Include dependency graph for descriptorPool.cpp:




## 8.61 descriptorPool.cpp

Go to the documentation of this file.

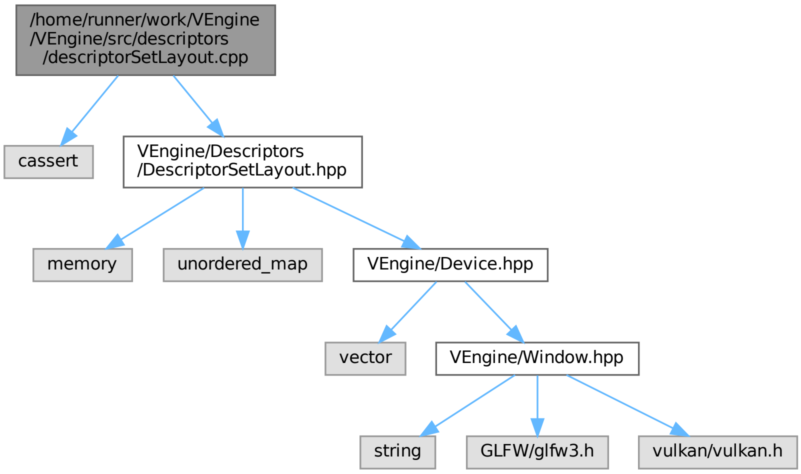
```
#include "VEngine/Descriptors/DescriptorPool.hpp"

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;
}
```

## 8.62 /home/runner/work/VEngine/VEngine/src/descriptors/descriptor←SetLayout.cpp File Reference

```
#include <cassert>
#include "VEngine/Descriptors/DescriptorSetLayout.hpp"
```

Include dependency graph for descriptorSetLayout.cpp:




## 8.63 descriptorSetLayout.cpp

Go to the documentation of this file.

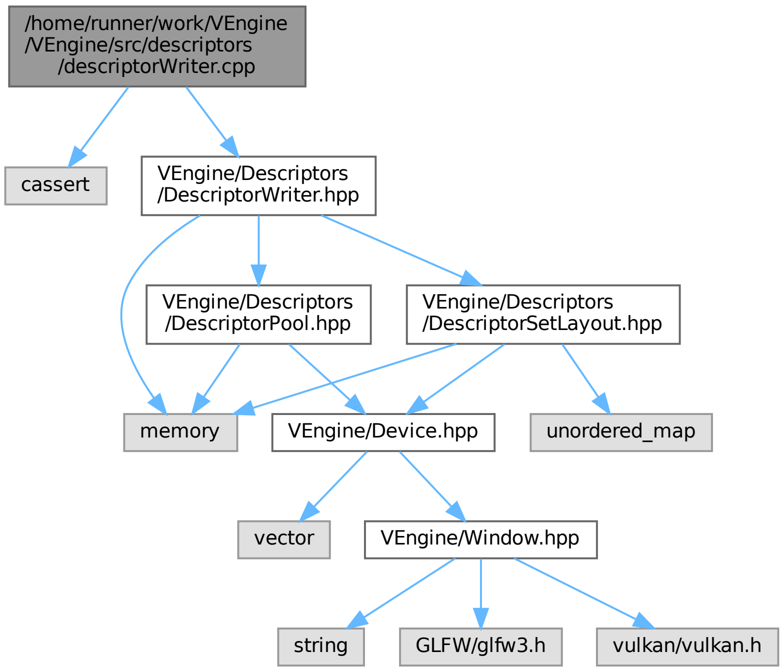
```
#include <cassert>

#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 [fst, snd] : bindings) {
        setLayoutBindings.push_back(snd);
    }

```

```
    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!");
    }
}
```

## 8.64 /home/runner/work/VEngine/VEngine/src/descriptors/descriptor↩ Writer.cpp File Reference

```
#include <cassert>
#include "VEngine/Descriptors/DescriptorWriter.hpp"
```

Include dependency graph for descriptorWriter.cpp:




## 8.65 descriptorWriter.cpp

Go to the documentation of this file.

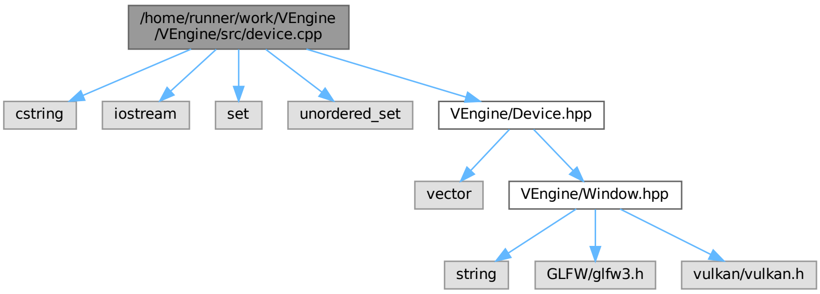
```
#include <cassert>

#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.at(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.at(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 &[sType, pNext, dstSet, dstBinding, dstArrayElement, descriptorCount, descriptorType, pImageInfo, pBufferInfo, pTexelBufferView] : m_writes) {
        dstSet = set;
    }
    vkUpdateDescriptorSets(m_pool.m_device.device(), static_cast<unsigned int>(m_writes.size()), m_writes.data(), 0, nullptr);
}
```

## 8.66 /home/runner/work/VEngine/VEngine/src/device.cpp File Reference

```
#include <cstring>
#include <iostream>
#include <set>
#include <unordered_set>
#include "VEngine/Device.hpp"
```

Include dependency graph for device.cpp:




## Functions

- static VKAPI_ATTR VkBool32 VKAPI_CALL debugCallback (const VkDebugUtilsMessageSeverityFlagBits↵ EXT messageSeverity, const VkDebugUtilsMessageTypeFlagsEXT messageType, const VkDebugUtils↵ MessengerCallbackDataEXT ∗pCallbackData, void ∗pUserData)
- VkResult CreateDebugUtilsMessengerEXT (const VkInstance instance, const VkDebugUtilsMessenger↵ CreateInfoEXT ∗pCreateInfo, const VkAllocationCallbacks ∗pAllocator, VkDebugUtilsMessengerEXT ∗p↵ DebugMessenger)
- void DestroyDebugUtilsMessengerEXT (const VkInstance instance, const VkDebugUtilsMessengerEXT debugMessenger, const VkAllocationCallbacks ∗pAllocator)

## 8.66.1 Function Documentation

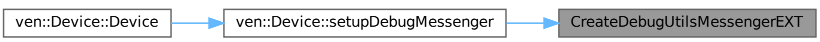
### 8.66.1.1 CreateDebugUtilsMessengerEXT()

```
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:



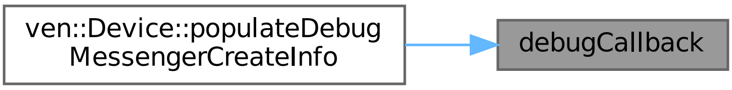
#### 8.66.1.2 debugCallback()

```
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:




#### 8.66.1.3 DestroyDebugUtilsMessengerEXT()

```
void DestroyDebugUtilsMessengerEXT (
             const VkInstance instance,
             const VkDebugUtilsMessengerEXT debugMessenger,
             const VkAllocationCallbacks * pAllocator)
```

Definition at line 25 of file device.cpp.

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

Here is the caller graph for this function:

## 8.67 device.cpp

Go to the documentation of this file.

```
#include <cstring>
#include <iostream>
#include <set>
#include <unordered_set>

#include "VEngine/Device.hpp"

static VKAPI_ATTR VkBool32 VKAPI_CALL debugCallback(const VkDebugUtilsMessageSeverityFlagBitsEXT messageSeverity, const VkDebugUtilsMessageTypeFlagsEXT messageType, const VkDebugUtilsMessengerCallbackDataEXT *pCallbackData, void *pUserData)
{
    (void) pUserData; (void) messageSeverity; (void) messageType;

    std::cerr « "validation layer: " « pCallbackData->pMessage « '\n';
    return VK_FALSE;
}

VkResult CreateDebugUtilsMessengerEXT(const VkInstance instance, const VkDebugUtilsMessengerCreateInfoEXT *pCreateInfo, const VkAllocationCallbacks *pAllocator, VkDebugUtilsMessengerEXT *pDebugMessenger)
{
    if (const auto func = reinterpret_cast<PFN_vkCreateDebugUtilsMessengerEXT>(vkGetInstanceProcAddr(instance, "vkCreateDebugUtilsMessengerEXT")); func != nullptr) {
        return func(instance, pCreateInfo, pAllocator, pDebugMessenger);
    }

    return VK_ERROR_EXTENSION_NOT_PRESENT;
}

void DestroyDebugUtilsMessengerEXT(const VkInstance instance, const VkDebugUtilsMessengerEXT debugMessenger, const VkAllocationCallbacks *pAllocator)
{
    if (const auto func = reinterpret_cast<PFN_vkDestroyDebugUtilsMessengerEXT>(vkGetInstanceProcAddr(instance, "vkDestroyDebugUtilsMessengerEXT")); func != nullptr) {
        func(instance, debugMessenger, pAllocator);
    }
}

ven::Device::Device(Window &window) : m_window{window}
{
    createInstance();
    setupDebugMessenger();
    createSurface();
    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;

    const std::vector<const char *> extensions = getRequiredExtensions();
```

```cpp
    createInfo.enabledExtensionCount = static_cast<uint32_t>(extensions.size());
    createInfo.ppEnabledExtensionNames = extensions.data();

    VkDebugUtilsMessengerCreateInfoEXT debugCreateInfo;
    if (enableValidationLayers) {
        createInfo.enabledLayerCount = static_cast<uint32_t>(m_validationLayers.size());
        createInfo.ppEnabledLayerNames = m_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, &m_properties);
    std::cout « "physical device: " « m_properties.deviceName « '\n';
}

void ven::Device::createLogicalDevice()
{
    const auto [graphicsFamily, presentFamily, graphicsFamilyHasValue, presentFamilyHasValue] = findQueueFamilies(m_physicalDevice);

    std::vector<VkDeviceQueueCreateInfo> queueCreateInfos;
    const std::set<uint32_t> uniqueQueueFamilies = {graphicsFamily, 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>(m_deviceExtensions.size());
    createInfo.ppEnabledExtensionNames = m_deviceExtensions.data();

        // might not really be necessary anymore because device specific validation layers
        // have been deprecated
    if (enableValidationLayers) {
        createInfo.enabledLayerCount = static_cast<uint32_t>(m_validationLayers.size());
        createInfo.ppEnabledLayerNames = m_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, graphicsFamily, 0, &m_graphicsQueue);
    vkGetDeviceQueue(m_device, 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) {
        auto [capabilities, formats, presentModes] = querySwapChainSupport(device);
        swapChainAdequate = !formats.empty() && !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 *validationLayer : m_validationLayers) {
        bool layerFound = false;

        for (const auto &[layerName, specVersion, implementationVersion, description] : availableLayers) {
            if (strcmp(layerName, validationLayer) == 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 &[extensionName, specVersion] : extensions) {
        std::cout « '\t' « extensionName « '\n';
        available.insert(extensionName);
    }

    std::cout « "required extensions:\n";
    for (const std::vector<const char *> requiredExtensions = getRequiredExtensions(); const auto
&required : requiredExtensions) {
        std::cout « "\t" « required « '\n';
        if (!available.contains(required)) {
            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());

    std::set<std::string> requiredExtensions(m_deviceExtensions.begin(), m_deviceExtensions.end());
     for (const auto &[extensionName, specVersion] : availableExtensions) {
         requiredExtensions.erase(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 &[queueFlags, queueCount, timestampValidBits, minImageTransferGranularity] :
queueFamilies) {
        if (queueCount > 0 && ((queueFlags & VK_QUEUE_GRAPHICS_BIT) != 0U)) {
            indices.graphicsFamily = index;
            indices.graphicsFamilyHasValue = true;
        }
        VkBool32 presentSupport = 0U;
        vkGetPhysicalDeviceSurfaceSupportKHR(device, index, m_surface, &presentSupport);
        if (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 properties) 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 & properties) == properties) {
            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 properties, 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, properties);

    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
```

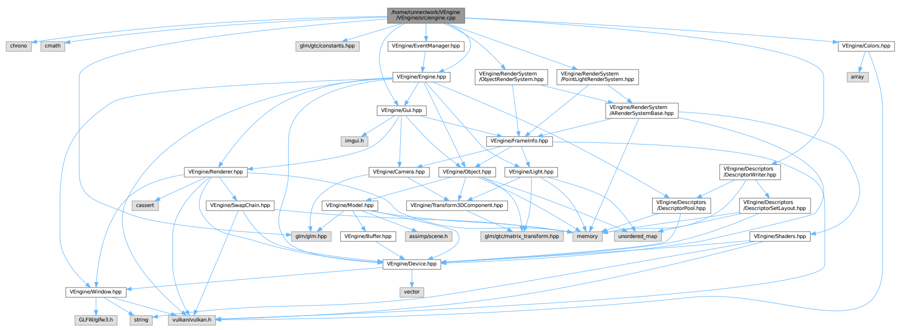
```cpp
{
    VkCommandBufferAllocateInfo allocInfo{};
    allocInfo.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_ALLOCATE_INFO;
    allocInfo.level = VK_COMMAND_BUFFER_LEVEL_PRIMARY;
    allocInfo.commandPool = m_commandPool;
    allocInfo.commandBufferCount = 1;

    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 = {.x=0, .y=0, .z=0};
    region.imageExtent = {.width=width, .height=height, .depth=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 memory!");
```

```
    }

    if (vkBindImageMemory(m_device, image, imageMemory, 0) != VK_SUCCESS) {
        throw std::runtime_error("failed to bind image memory!");
    }
}
```

## 8.68 /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/EventManager.hpp>
#include "VEngine/Engine.hpp"
#include "VEngine/RenderSystem/ObjectRenderSystem.hpp"
#include "VEngine/RenderSystem/PointLightRenderSystem.hpp"
#include "VEngine/Descriptors/DescriptorWriter.hpp"
#include "VEngine/Gui.hpp"
#include "VEngine/Colors.hpp"
```

Include dependency graph for engine.cpp:



## 8.69 engine.cpp

Go to the documentation of this file.

```
#include <chrono>
#include <cmath>

#include <glm/glm.hpp>
#include <glm/gtc/constants.hpp>
#include <VEngine/EventManager.hpp>

#include "VEngine/Engine.hpp"
#include "VEngine/RenderSystem/ObjectRenderSystem.hpp"
#include "VEngine/RenderSystem/PointLightRenderSystem.hpp"
#include "VEngine/Descriptors/DescriptorWriter.hpp"
#include "VEngine/Gui.hpp"
#include "VEngine/Colors.hpp"

ven::Engine::Engine(const uint32_t width, const uint32_t height, const std::string &title) :
m_state(EDITOR), m_window(width, height, title) {
    createInstance();
    createSurface();
    Gui::init(m_window.getGLFWindow(), m_instance, &m_device, m_renderer.getSwapChainRenderPass());
    m_globalPool =
DescriptorPool::Builder(m_device).setMaxSets(MAX_FRAMES_IN_FLIGHT).addPoolSize(VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER,
MAX_FRAMES_IN_FLIGHT).build();
    loadObjects();
}

```

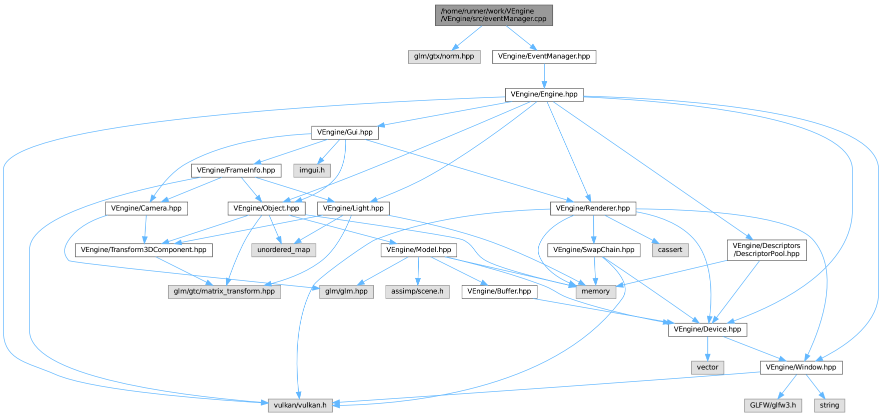
```
void ven::Engine::createInstance()
{
    uint32_t glfwExtensionCount = 0;
    const char** glfwExtensions = nullptr;
    VkInstanceCreateInfo createInfo{};
    constexpr VkApplicationInfo appInfo{ .sType = VK_STRUCTURE_TYPE_APPLICATION_INFO, .pNext = nullptr, .pApplicationName = "VEngine App", .applicationVersion = 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()
{
    constexpr std::array lightColors{
        Colors::RED_4,
        Colors::GREEN_4,
        Colors::BLUE_4,
        Colors::YELLOW_4,
        Colors::CYAN_4,
        Colors::MAGENTA_4
    };
    std::shared_ptr model = Model::createModelFromFile(m_device, "assets/models/quad.obj");

    Object quad = Object::createObject();
    quad.setName("quad");
    quad.setModel(model);
    quad.transform3D.translation = {0.F, .5F, 0.F};
    quad.transform3D.scale = {3.F, 1.F, 3.F};
    m_objects.emplace(quad.getId(), std::move(quad));

    model = Model::createModelFromFile(m_device, "assets/models/flat_vase.obj");
    Object flatVase = Object::createObject();
    flatVase.setName("flat vase");
    flatVase.setModel(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, "assets/models/smooth_vase.obj");
    Object smoothVase = Object::createObject();
    smoothVase.setName("smooth vase");
    smoothVase.setModel(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));

    for (std::size_t i = 0; i < lightColors.size(); i++)
    {
        Light pointLight = Light::createLight();
        pointLight.color = lightColors.at(i);
        glm::mat4 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_lights.emplace(pointLight.getId(), std::move(pointLight));
    }
}

void ven::Engine::mainLoop()
{
    GlobalUbo ubo{};
    Camera camera{};
    EventManager eventManager{};
    std::chrono::duration<float> deltaTime{};
    VkCommandBuffer_T *commandBuffer = nullptr;
    float frameTime = NAN;
    unsigned long frameIndex = 0;
    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(MAX_FRAMES_IN_FLIGHT);
    std::vector<VkDescriptorSet> globalDescriptorSets(MAX_FRAMES_IN_FLIGHT);
    ObjectRenderSystem objectRenderSystem(m_device, m_renderer.getSwapChainRenderPass(),
```

```
globalSetLayout->getDescriptorSetLayout());
    PointLightRenderSystem pointLightRenderSystem(m_device, m_renderer.getSwapChainRenderPass(), globalSetLayout->getDescriptorSetLayout());
    VkDescriptorBufferInfo bufferInfo{};

    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]);
    }

    while (m_state != EXIT)
    {
        glfwPollEvents();
        eventManager.handleEvents(m_window.getGLFWindow(), &m_state, camera, m_gui, frameTime);
        newTime = std::chrono::high_resolution_clock::now();
        deltaTime = newTime - currentTime;
        currentTime = newTime;
        frameTime = deltaTime.count();
        commandBuffer = m_renderer.beginFrame();

        camera.setViewXYZ(camera.transform3D.translation, camera.transform3D.rotation);
        camera.setPerspectiveProjection(m_renderer.getAspectRatio());

        if (commandBuffer != nullptr) {
            frameIndex = m_renderer.getFrameIndex();
            FrameInfo frameInfo{.frameIndex=frameIndex, .frameTime=frameTime, .commandBuffer=commandBuffer, .camera=camera, .globalDescriptorSet=globalDescriptorSets[frameIndex], .objects=m_objects, .lights=m_lights};
            ubo.projection = camera.getProjection();
            ubo.view = camera.getView();
            ubo.inverseView = camera.getInverseView();
            PointLightRenderSystem::update(frameInfo, ubo);
            uboBuffers[frameIndex]->writeToBuffer(&ubo);
            uboBuffers[frameIndex]->flush();

            m_renderer.beginSwapChainRenderPass(frameInfo.commandBuffer);
            objectRenderSystem.render(frameInfo);
            pointLightRenderSystem.render(frameInfo);

            if (m_gui.getState() == VISIBLE) { Gui::render(&m_renderer, m_objects, m_lights, camera, m_device.getPhysicalDevice(), ubo); }

            m_renderer.endSwapChainRenderPass(commandBuffer);
            m_renderer.endFrame();
            commandBuffer = nullptr;
        }
    }
    Gui::cleanup();
    vkDeviceWaitIdle(m_device.device());
}
```

## 8.70 /home/runner/work/VEngine/VEngine/src/eventManager.cpp File Reference

```
#include <glm/gtx/norm.hpp>
#include "VEngine/EventManager.hpp"
```

Include dependency graph for eventManager.cpp:




# 8.71 eventManager.cpp

Go to the documentation of this file.

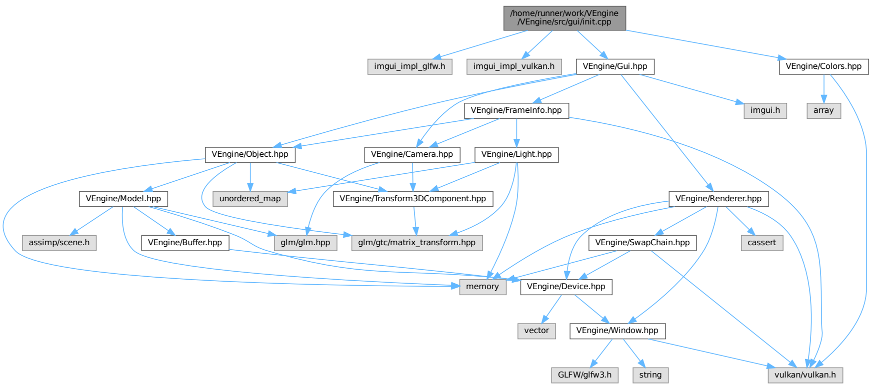
```
#include <glm/gtx/norm.hpp>
#include "VEngine/EventManager.hpp"

void ven::EventManager::handleEvents(GLFWwindow *window, ENGINE_STATE *engineState, Camera& camera, Gui& gui, const float dt) const
{
    if (glfwWindowShouldClose(window) == GLFW_TRUE) {
        updateEngineState(engineState, EXIT);
    }
    moveCamera(window, camera, gui, dt);
}

bool ven::EventManager::isKeyJustPressed(GLFWwindow* window, const int key, std::unordered_map<int, bool>& keyStates)
{
    const bool isPressed = glfwGetKey(window, key) == GLFW_PRESS;
    const bool wasPressed = keyStates[key];

    keyStates[key] = isPressed;

    return isPressed && !wasPressed;
}

void ven::EventManager::moveCamera(GLFWwindow* window, Camera& camera, Gui& gui, const float dt) const
{
    glm::vec3 rotate{0};
    glm::vec3 moveDir{0.F};
    const float yaw = camera.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};
    const std::array<KeyAction, 4> rotationActions = {{
        {.key=m_keys.lookLeft, .dir=&rotate, .value={0.F, -1.F, 0.F}},
        {.key=m_keys.lookRight, .dir=&rotate, .value={0.F, 1.F, 0.F}},
        {.key=m_keys.lookUp, .dir=&rotate, .value={1.F, 0.F, 0.F}},
        {.key=m_keys.lookDown, .dir=&rotate, .value={-1.F, 0.F, 0.F}}
    }};
    const std::array<KeyAction, 6> moveActions = {{
        {.key=m_keys.moveForward, .dir=&moveDir, .value=forwardDir},
        {.key=m_keys.moveBackward, .dir=&moveDir, .value=-forwardDir},
        {.key=m_keys.moveRight, .dir=&moveDir, .value=rightDir},
        {.key=m_keys.moveLeft, .dir=&moveDir, .value=-rightDir},
        {.key=m_keys.moveUp, .dir=&moveDir, .value=upDir},
        {.key=m_keys.moveDown, .dir=&moveDir, .value=-upDir}
    }};

    for (const auto&[key, dir, value] : rotationActions) {
        if (glfwGetKey(window, key) == GLFW_PRESS) {
            *dir += value;
        }
    }
```

```
    for (const auto&[key, dir, value] : moveActions) {
        if (glfwGetKey(window, key) == GLFW_PRESS) {
            *dir += value;
        }
    }

    if (glm::length2(rotate) > std::numeric_limits<float>::epsilon()) {
        camera.transform3D.rotation += camera.getLookSpeed() * dt * glm::normalize(rotate);
    }
    if (glm::length2(moveDir) > std::numeric_limits<float>::epsilon()) {
        camera.transform3D.translation += camera.getMoveSpeed() * dt * glm::normalize(moveDir);
    }

    camera.transform3D.rotation.x = glm::clamp(camera.transform3D.rotation.x, -1.5F, 1.5F);
    camera.transform3D.rotation.y = glm::mod(camera.transform3D.rotation.y, glm::two_pi<float>());

    if (isKeyJustPressed(window, m_keys.toggleGui, m_keyState)) {
        gui.getState() == HIDDEN ? gui.setState(VISIBLE) : gui.setState(HIDDEN);
    }
}
```

## 8.72 /home/runner/work/VEngine/VEngine/src/gui/init.cpp File Reference

```
#include <imgui_impl_glfw.h>
#include <imgui_impl_vulkan.h>
#include "VEngine/Gui.hpp"
#include "VEngine/Colors.hpp"
```

Include dependency graph for init.cpp:




### Variables

- static constexpr uint32_t DESCRIPTOR_COUNT = 1000

### 8.72.1 Variable Documentation

#### 8.72.1.1 DESCRIPTOR_COUNT

```
uint32_t DESCRIPTOR_COUNT = 1000 [static], [constexpr]
```

Definition at line 7 of file init.cpp.

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

## 8.73 init.cpp

Go to the documentation of this file.

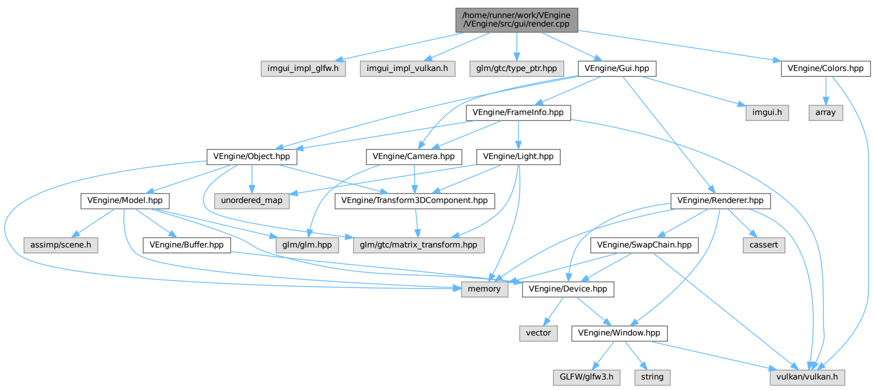
```
#include <imgui_impl_glfw.h>
#include <imgui_impl_vulkan.h>

#include "VEngine/Gui.hpp"
#include "VEngine/Colors.hpp"

static constexpr uint32_t DESCRIPTOR_COUNT = 1000;

ImGuiIO *ven::Gui::m_io = nullptr;

void ven::Gui::init(GLFWwindow* window, const VkInstance instance, const Device* device, const
VkRenderPass renderPass)
{
    VkDescriptorPool pool = nullptr;

    ImGui::CreateContext();
    m_io = &ImGui::GetIO();
    m_io->IniFilename = "assets/imgui-config.txt";

    // ImGui::StyleColorsDark();

    std::array<VkDescriptorPoolSize, 11> pool_sizes = {{
            { .type=VK_DESCRIPTOR_TYPE_SAMPLER, .descriptorCount=DESCRIPTOR_COUNT },
            { .type=VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, .descriptorCount=DESCRIPTOR_COUNT },
            { .type=VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, .descriptorCount=DESCRIPTOR_COUNT },
            { .type=VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, .descriptorCount=DESCRIPTOR_COUNT },
            { .type=VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER, .descriptorCount=DESCRIPTOR_COUNT },
            { .type=VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER, .descriptorCount=DESCRIPTOR_COUNT },
            { .type=VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER, .descriptorCount=DESCRIPTOR_COUNT },
            { .type=VK_DESCRIPTOR_TYPE_STORAGE_BUFFER, .descriptorCount=DESCRIPTOR_COUNT },
            { .type=VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC, .descriptorCount=DESCRIPTOR_COUNT },
            { .type=VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC, .descriptorCount=DESCRIPTOR_COUNT },
            { .type=VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT, .descriptorCount=DESCRIPTOR_COUNT }
    }};
    const VkDescriptorPoolCreateInfo pool_info = {
            VK_STRUCTURE_TYPE_DESCRIPTOR_POOL_CREATE_INFO,
            nullptr,
            VK_DESCRIPTOR_POOL_CREATE_FREE_DESCRIPTOR_SET_BIT,
            DESCRIPTOR_COUNT,
            std::size(pool_sizes),
            pool_sizes.data()
    };

    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);
    initStyle();
}

void ven::Gui::initStyle()
{
    ImGuiStyle& style = ImGui::GetStyle();
    style.Alpha = 1.0;
    style.WindowRounding = 3;
    style.GrabRounding = 1;
    style.GrabMinSize = 20;
    style.FrameRounding = 3;

    style.Colors[ImGuiCol_Text] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
    style.Colors[ImGuiCol_TextDisabled] = ImVec4(0.00F, 0.40F, 0.41F, 1.00F);
    style.Colors[ImGuiCol_WindowBg] = ImVec4(0.1F, 0.1F, 0.1F, 0.70F);
    style.Colors[ImGuiCol_Border] = ImVec4(0.00F, 1.00F, 1.00F, 0.35F);
    style.Colors[ImGuiCol_BorderShadow] = ImVec4(0.00F, 0.00F, 0.00F, 0.00F);
    style.Colors[ImGuiCol_FrameBg] = ImVec4(0.44F, 0.80F, 0.80F, 0.18F);
    style.Colors[ImGuiCol_FrameBgHovered] = ImVec4(0.44F, 0.80F, 0.80F, 0.27F);
    style.Colors[ImGuiCol_FrameBgActive] = ImVec4(0.44F, 0.81F, 0.86F, 0.66F);
    style.Colors[ImGuiCol_TitleBg] = ImVec4(0.14F, 0.18F, 0.21F, 0.73F);
    style.Colors[ImGuiCol_TitleBgCollapsed] = ImVec4(0.00F, 0.00F, 0.00F, 0.54F);
    style.Colors[ImGuiCol_TitleBgActive] = ImVec4(0.00F, 1.00F, 1.00F, 0.27F);
```

```
    style.Colors[ImGuiCol_MenuBarBg] = ImVec4(0.00F, 0.00F, 0.00F, 0.20F);
    style.Colors[ImGuiCol_ScrollbarBg] = ImVec4(0.22F, 0.29F, 0.30F, 0.71F);
    style.Colors[ImGuiCol_ScrollbarGrab] = ImVec4(0.00F, 1.00F, 1.00F, 0.44F);
    style.Colors[ImGuiCol_ScrollbarGrabHovered] = ImVec4(0.00F, 1.00F, 1.00F, 0.74F);
    style.Colors[ImGuiCol_ScrollbarGrabActive] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
    style.Colors[ImGuiCol_CheckMark] = ImVec4(0.00F, 1.00F, 1.00F, 0.68F);
    style.Colors[ImGuiCol_SliderGrab] = ImVec4(0.00F, 1.00F, 1.00F, 0.36F);
    style.Colors[ImGuiCol_SliderGrabActive] = ImVec4(0.00F, 1.00F, 1.00F, 0.76F);
    style.Colors[ImGuiCol_Button] = ImVec4(0.00F, 0.65F, 0.65F, 0.46F);
    style.Colors[ImGuiCol_ButtonHovered] = ImVec4(0.01F, 1.00F, 1.00F, 0.43F);
    style.Colors[ImGuiCol_ButtonActive] = ImVec4(0.00F, 1.00F, 1.00F, 0.62F);
    style.Colors[ImGuiCol_Header] = ImVec4(0.00F, 1.00F, 1.00F, 0.33F);
    style.Colors[ImGuiCol_HeaderHovered] = ImVec4(0.00F, 1.00F, 1.00F, 0.42F);
    style.Colors[ImGuiCol_HeaderActive] = ImVec4(0.00F, 1.00F, 1.00F, 0.54F);
    style.Colors[ImGuiCol_ResizeGrip] = ImVec4(0.00F, 1.00F, 1.00F, 0.54F);
    style.Colors[ImGuiCol_ResizeGripHovered] = ImVec4(0.00F, 1.00F, 1.00F, 0.74F);
    style.Colors[ImGuiCol_ResizeGripActive] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
    style.Colors[ImGuiCol_PlotLines] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
    style.Colors[ImGuiCol_PlotLinesHovered] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
    style.Colors[ImGuiCol_PlotHistogram] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
    style.Colors[ImGuiCol_PlotHistogramHovered] = ImVec4(0.00F, 1.00F, 1.00F, 1.00F);
    style.Colors[ImGuiCol_TextSelectedBg] = ImVec4(0.00F, 1.00F, 1.00F, 0.22F);
}
```

## 8.74 /home/runner/work/VEngine/VEngine/src/gui/render.cpp File Reference

```
#include <imgui_impl_glfw.h>
#include <imgui_impl_vulkan.h>
#include <glm/gtc/type_ptr.hpp>
#include "VEngine/Gui.hpp"
#include "VEngine/Colors.hpp"
```

Include dependency graph for render.cpp:




## 8.75 render.cpp

Go to the documentation of this file.

```
#include <imgui_impl_glfw.h>
#include <imgui_impl_vulkan.h>

#include <glm/gtc/type_ptr.hpp>

#include "VEngine/Gui.hpp"
#include "VEngine/Colors.hpp"

void ven::Gui::cleanup()
{
    ImGui_ImplVulkan_Shutdown();
```

```
    ImGui_ImplGlfw_Shutdown();
    ImGui::DestroyContext();
}

void ven::Gui::render(Renderer* renderer, std::unordered_map<unsigned int, Object>& objects, std::unordered_map<unsigned int, Light>& lights, Camera& camera, const VkPhysicalDevice physicalDevice, GlobalUbo& ubo)
{
    VkPhysicalDeviceProperties deviceProperties;
    vkGetPhysicalDeviceProperties(physicalDevice, &deviceProperties);

    ImGui_ImplVulkan_NewFrame();
    ImGui_ImplGlfw_NewFrame();
    ImGui::NewFrame();

    renderFrameWindow();

    ImGui::Begin("Debug Window");
    rendererSection(renderer, ubo);
    cameraSection(camera);
    objectsSection(objects);
    lightsSection(lights);
    inputsSection(m_io);
    devicePropertiesSection(deviceProperties);

    ImGui::End();
    ImGui::Render();
    ImGui_ImplVulkan_RenderDrawData(ImGui::GetDrawData(), renderer->getCurrentCommandBuffer());
}

void ven::Gui::renderFrameWindow()
{
    const float framerate = m_io->Framerate;

    ImGui::SetNextWindowPos(ImVec2(0.0F, 0.0F), ImGuiCond_Always, ImVec2(0.0F, 0.0F));
    ImGui::Begin("Application Info", nullptr, ImGuiWindowFlags_NoDecoration | ImGuiWindowFlags_NoMove | ImGuiWindowFlags_NoResize | ImGuiWindowFlags_NoSavedSettings | ImGuiWindowFlags_NoFocusOnAppearing | ImGuiWindowFlags_NoNav);
    ImGui::Text("FPS: %.1f", framerate);
    ImGui::Text("Frame time: %.3fms", 1000.0F / framerate);
    ImGui::End();
}

void ven::Gui::rendererSection(Renderer *renderer, GlobalUbo& ubo)
{
    if (ImGui::CollapsingHeader("Renderer")) {
        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())) {
                const 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##clearColor",
                             &item_current,
                             [](void*, const int idx, const char** out_text) -> bool {
                                 if (idx < 0 || idx >= static_cast<int>(std::size(Colors::COLOR_PRESETS_VK))) { return false; }
                                 *out_text = Colors::COLOR_PRESETS_VK.at(static_cast<unsigned long>(idx)).first;
                                 return true;
                             },
                             nullptr,
                             std::size(Colors::COLOR_PRESETS_VK))) {
                renderer->setClearValue(Colors::COLOR_PRESETS_VK.at(static_cast<unsigned long>(item_current)).second);
                             }

            ImGui::TableNextColumn();
            ImGui::ColorEdit4("Ambient Light Color", glm::value_ptr(ubo.ambientLightColor));
            ImGui::TableNextColumn();
            if (ImGui::Combo("Color Presets##ambientColor",
                             &item_current,
                             [](void*, const int idx, const char** out_text) -> bool {
                                 if (idx < 0 || idx >= static_cast<int>(std::size(Colors::COLOR_PRESETS_4))) { return false; }
                                 *out_text = Colors::COLOR_PRESETS_4.at(static_cast<unsigned long>(idx)).first;
                                 return true;
```

```
                            },
                            nullptr,
                            std::size(Colors::COLOR_PRESETS_4))) {
                ubo.ambientLightColor = Colors::COLOR_PRESETS_4.at(static_cast<unsigned long>(item_current)).second;
                            }

            ImGui::TableNextColumn();
            ImGui::SliderFloat(("Intensity##" + std::to_string(0)).c_str(), &ubo.ambientLightColor.a, 0.0F, 1.0F);
            ImGui::TableNextColumn();
            if (ImGui::Button("Reset##ambientIntensity")) { ubo.ambientLightColor.a = DEFAULT_AMBIENT_LIGHT_INTENSITY; }

            ImGui::EndTable();
        }

        static bool fullscreen = false;
        if (ImGui::Checkbox("Fullscreen", &fullscreen)) {
            renderer->getWindow().setFullscreen(fullscreen, renderer->getWindow().getExtent().width, renderer->getWindow().getExtent().height);
        }
    }
}

void ven::Gui::cameraSection(Camera &camera)
{
    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(camera.transform3D.translation), 0.1F);
            ImGui::TableNextColumn();
            if (ImGui::Button("Reset##position")) { camera.transform3D.translation = DEFAULT_POSITION; }

            ImGui::TableNextColumn();
            ImGui::DragFloat3("Rotation", glm::value_ptr(camera.transform3D.rotation), 0.1F);
            ImGui::TableNextColumn();
            if (ImGui::Button("Reset##rotation")) { camera.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();
            float moveSpeed = camera.getMoveSpeed();
            if (ImGui::SliderFloat("Move speed", &moveSpeed, 0.1F, 10.0F)) { camera.setMoveSpeed(moveSpeed); }
            ImGui::TableNextColumn();
            if (ImGui::Button("Reset##moveSpeed")) { camera.setMoveSpeed(DEFAULT_MOVE_SPEED); }

            ImGui::TableNextColumn();
            float lookSpeed = camera.getLookSpeed();
            if (ImGui::SliderFloat("Look speed", &lookSpeed, 0.1F, 10.0F)) { camera.setLookSpeed(lookSpeed); }
            ImGui::TableNextColumn();
            if (ImGui::Button("Reset##lookSpeed")) { camera.setLookSpeed(DEFAULT_LOOK_SPEED); }

            ImGui::EndTable();
        }
    }
}

void ven::Gui::objectsSection(std::unordered_map<unsigned int, Object>& objects)
{
    if (ImGui::CollapsingHeader("Objects")) {
        bool open = false;
        for (auto& [id, object] : objects) {
            ImGui::PushStyleColor(ImGuiCol_Text, { Colors::GRAY_4.r, Colors::GRAY_4.g, Colors::GRAY_4.b, 1.0F });
            open = ImGui::TreeNode(std::string(object.getName() + " [" + std::to_string(object.getId()) + "]").c_str());
```

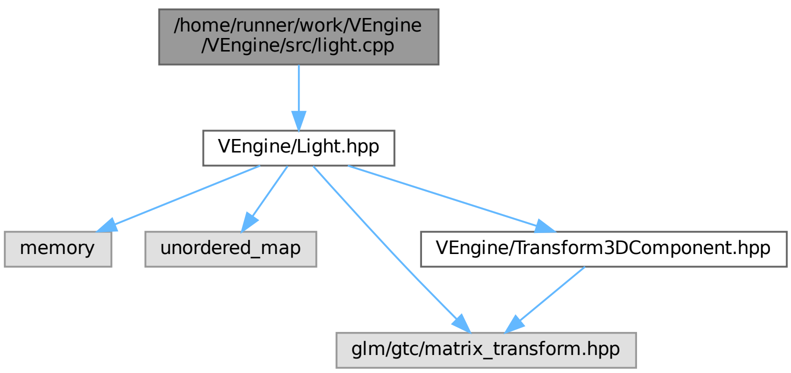
```cpp
            ImGui::PopStyleColor(1);
            if (open) {
                ImGui::DragFloat3(("Position##" + object.getName()).c_str(), glm::value_ptr(object.transform3D.translation), 0.1F);
                ImGui::DragFloat3(("Rotation##" + object.getName()).c_str(), glm::value_ptr(object.transform3D.rotation), 0.1F);
                ImGui::DragFloat3(("Scale##" + object.getName()).c_str(), glm::value_ptr(object.transform3D.scale), 0.1F);
                ImGui::Text("Address: %p", &object);
                ImGui::TreePop();
            }
        }
    }
}

void ven::Gui::lightsSection(std::unordered_map<unsigned int, Light> &lights)
{
    if (ImGui::CollapsingHeader("Lights")) {
        bool open = false;

        for (auto& [id, light] : lights) {
            ImGui::PushStyleColor(ImGuiCol_Text, {light.color.r, light.color.g, light.color.b, 1.0F});
            open = ImGui::TreeNode(std::string(light.getName() + " [" + std::to_string(light.getId()) + "]").c_str());
            ImGui::PopStyleColor(1);
            if (open) {
                ImGui::Text("Address: %p", &light);
                ImGui::DragFloat3(("Position##" + std::to_string(light.getId())).c_str(), glm::value_ptr(light.transform3D.translation), 0.1F);
                ImGui::DragFloat3(("Rotation##" + std::to_string(light.getId())).c_str(), glm::value_ptr(light.transform3D.rotation), 0.1F);
                ImGui::DragFloat3(("Scale##" + std::to_string(light.getId())).c_str(), glm::value_ptr(light.transform3D.scale), 0.1F);
                if (ImGui::BeginTable("ColorTable", 2)) {
                    ImGui::TableNextColumn(); ImGui::ColorEdit4(("Color##" + std::to_string(light.getId())).c_str(), glm::value_ptr(light.color));

                    ImGui::TableNextColumn();
                    static int item_current = 0;
                    if (ImGui::Combo("Color Presets",
                                     &item_current,
                                     [](void*, const int idx, const char** out_text) -> bool {
                                         if (idx < 0 || idx >= static_cast<int>(std::size(Colors::COLOR_PRESETS_3))) { return false; }
                                         *out_text = Colors::COLOR_PRESETS_3.at(static_cast<unsigned long>(idx)).first;
                                         return true;
                                     },
                                     nullptr,
                                     std::size(Colors::COLOR_PRESETS_3))) {
                        light.color = {Colors::COLOR_PRESETS_3.at(static_cast<unsigned long>(item_current)).second, light.color.a};
                    }

                    ImGui::TableNextColumn();
                    ImGui::SliderFloat(("Intensity##" + std::to_string(light.getId())).c_str(), &light.color.a, 0.0F, 5.F);
                    ImGui::TableNextColumn();
                    if (ImGui::Button(("Reset##" + std::to_string(light.getId())).c_str())) { light.color.a = DEFAULT_LIGHT_INTENSITY; }

                    ImGui::EndTable();
                }
                ImGui::TreePop();
            }
        }
    }
}

void ven::Gui::inputsSection(const ImGuiIO* io)
{
    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 (auto 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);
        }
    }
}

void ven::Gui::devicePropertiesSection(VkPhysicalDeviceProperties deviceProperties)
{
    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();
        }
    }
}
```

## 8.76 /home/runner/work/VEngine/VEngine/src/light.cpp File Reference

```
#include "VEngine/Light.hpp"
```

Include dependency graph for light.cpp:



## 8.77 light.cpp

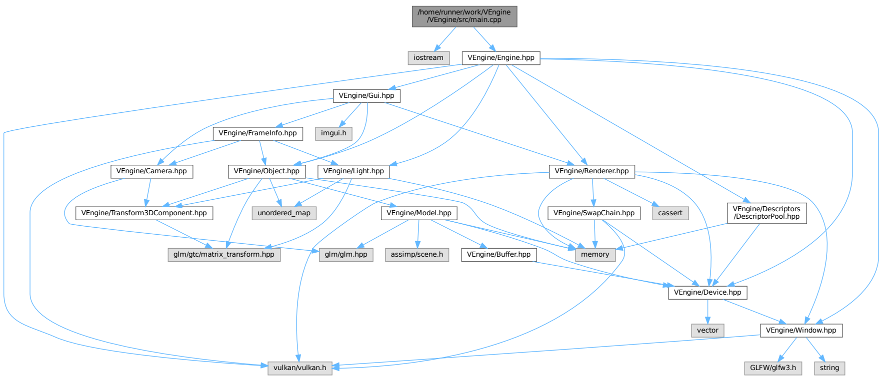
Go to the documentation of this file.

```cpp
#include "VEngine/Light.hpp"

ven::Light ven::Light::createLight(const float radius, const glm::vec4 color)
{
    static unsigned int objId = 0;
    Light light(objId++);

    light.color = color;
    light.transform3D.scale.x = radius;

    return light;
}
```

## 8.78 /home/runner/work/VEngine/VEngine/src/main.cpp File Reference

```cpp
#include <iostream>
#include "VEngine/Engine.hpp"
```

Include dependency graph for main.cpp:




### Functions

- int main ()

### 8.78.1 Function Documentation

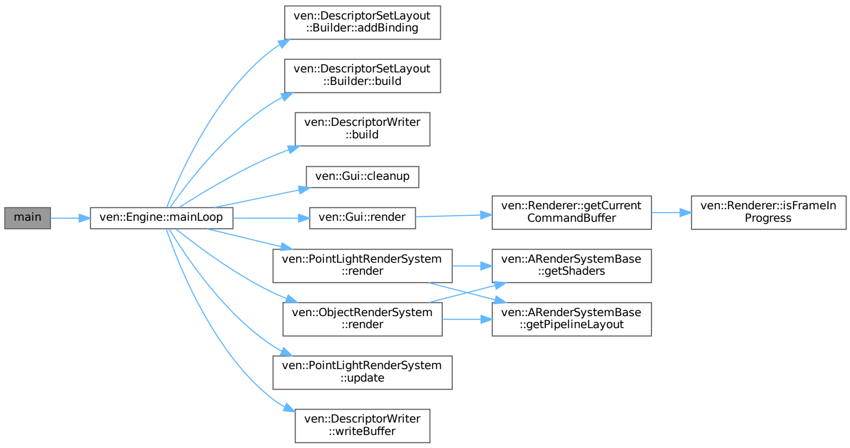
#### 8.78.1.1 main()

```cpp
int main ()
```

Definition at line 7 of file main.cpp.

References ven::Engine::mainLoop().

Here is the call graph for this function:




## 8.79 main.cpp

Go to the documentation of this file.

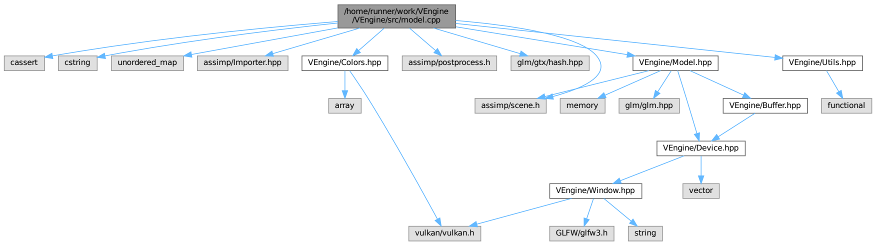
```
#include <iostream>

#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;
}
```

## 8.80 /home/runner/work/VEngine/VEngine/src/model.cpp File Reference

```
#include <cassert>
#include <cstring>
#include <unordered_map>
#include <assimp/Importer.hpp>
#include <assimp/scene.h>
#include <assimp/postprocess.h>
#include <glm/gtx/hash.hpp>
#include "VEngine/Colors.hpp"
#include "VEngine/Model.hpp"
```

```
#include "VEngine/Utils.hpp"
```

Include dependency graph for model.cpp:




### Classes

- struct std::hash< ven::Model::Vertex >

### Macros

- #define GLM_ENABLE_EXPERIMENTAL

## 8.80.1 Macro Definition Documentation

### 8.80.1.1 GLM_ENABLE_EXPERIMENTAL

```
#define GLM_ENABLE_EXPERIMENTAL
```

Definition at line 9 of file model.cpp.

# 8.81 model.cpp

Go to the documentation of this file.

```
00001 #include <cassert>
00002 #include <cstring>
00003 #include <unordered_map>
00004
00005 #include <assimp/Importer.hpp>
00006 #include <assimp/scene.h>
00007 #include <assimp/postprocess.h>
00008
00009 #define GLM_ENABLE_EXPERIMENTAL
00010 #include <glm/gtx/hash.hpp>
00011
00012 #include "VEngine/Colors.hpp"
00013 #include "VEngine/Model.hpp"
00014 #include "VEngine/Utils.hpp"
00015
00016 template<>
00017 struct std::hash<ven::Model::Vertex> {
00018     size_t operator()(ven::Model::Vertex const &vertex) const noexcept {
00019         size_t seed = 0;
00020         ven::hashCombine(seed, vertex.position, vertex.color, vertex.normal, vertex.uv);
00021         return seed;
00022     }
00023 };
00024
00025 ven::Model::Model(Device &device, const Builder &builder) : m_device{device}, m_vertexCount(0),
      m_indexCount(0)
```

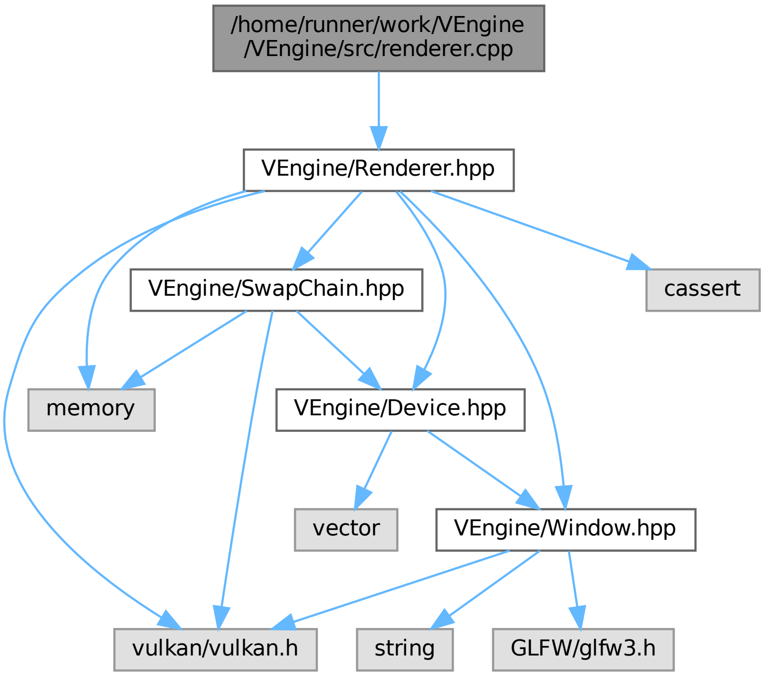
```
{
    createVertexBuffer(builder.vertices);
    createIndexBuffer(builder.indices);
}

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;
    }

    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(), sizeof(indices[0]) * m_indexCount);
}

void ven::Model::draw(const VkCommandBuffer commandBuffer) const
{
    if (m_hasIndexBuffer) {
        vkCmdDrawIndexed(commandBuffer, m_indexCount, 1, 0, 0, 0);
    } else {
        vkCmdDraw(commandBuffer, m_vertexCount, 1, 0, 0);
    }
}

void ven::Model::bind(const VkCommandBuffer commandBuffer) const
{
    const std::array buffers{m_vertexBuffer->getBuffer()};
    constexpr std::array<VkDeviceSize, 1> offsets{0};
    vkCmdBindVertexBuffers(commandBuffer, 0, 1, buffers.data(), offsets.data());

    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) {
    Assimp::Importer importer;

    const aiScene* scene = importer.ReadFile(filename, aiProcess_Triangulate | aiProcess_FlipUVs | aiProcess_CalcTangentSpace | aiProcess_GenNormals);

    if ((scene == nullptr) || ((scene->mFlags & AI_SCENE_FLAGS_INCOMPLETE) != 0U) || !scene->mRootNode) {
        throw std::runtime_error("Failed to load model with Assimp: " + std::string(importer.GetErrorString()));
    }

    vertices.clear();
    indices.clear();

    processNode(scene->mRootNode, scene);
}

void ven::Model::Builder::processNode(const aiNode* node, const aiScene* scene) {
    for (unsigned int i = 0; i < node->mNumMeshes; i++) {
        const aiMesh* mesh = scene->mMeshes[node->mMeshes[i]];
        processMesh(mesh, scene);
    }

    for (unsigned int i = 0; i < node->mNumChildren; i++) {
        processNode(node->mChildren[i], scene);
    }
}

void ven::Model::Builder::processMesh(const aiMesh* mesh, const aiScene* scene) {
    std::unordered_map<Vertex, uint32_t> uniqueVertices;

    for (unsigned int i = 0; i < mesh->mNumVertices; i++) {
        Vertex vertex{};

        vertex.position = glm::vec3(
            mesh->mVertices[i].x,
            mesh->mVertices[i].y,
            mesh->mVertices[i].z
        );

        if (mesh->HasNormals()) {
            vertex.normal = glm::vec3(
                mesh->mNormals[i].x,
                mesh->mNormals[i].y,
                mesh->mNormals[i].z
            );
        }

        if (mesh->mTextureCoords[0] != nullptr) {
            vertex.uv = glm::vec2(
                mesh->mTextureCoords[0][i].x,
                mesh->mTextureCoords[0][i].y
            );
        } else {
            vertex.uv = glm::vec2(0.0F, 0.0F);
        }

        if (vertex.color == Colors::BLACK_3) {
            vertex.color = Colors::WHITE_3;
        }

        if (!uniqueVertices.contains(vertex)) {
            uniqueVertices[vertex] = static_cast<uint32_t>(vertices.size());
            vertices.push_back(vertex);
        }

        indices.push_back(uniqueVertices[vertex]);
    }
}

```

## 8.82 /home/runner/work/VEngine/VEngine/src/renderer.cpp File Reference

`#include "VEngine/Renderer.hpp"`

Include dependency graph for renderer.cpp:




## 8.83 renderer.cpp

Go to the documentation of this file.

```
#include "VEngine/Renderer.hpp"

void ven::Renderer::createCommandBuffers()
{
    m_commandBuffers.resize(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()
{
```

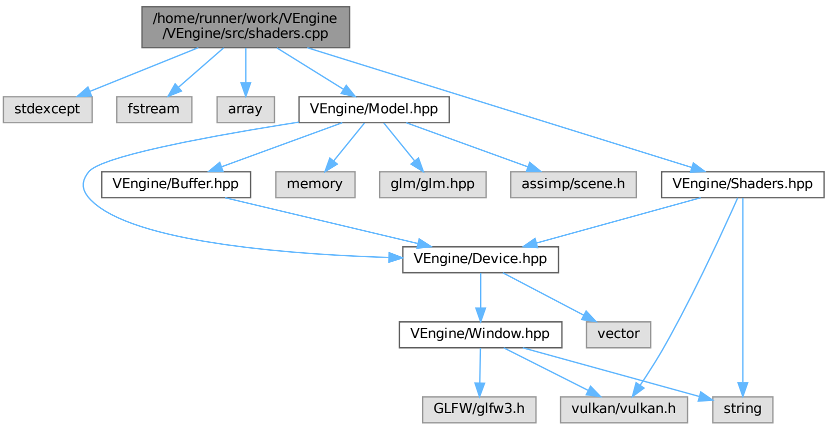
```
    VkExtent2D extent = m_window.getExtent();
    while (extent.width == 0 || extent.height == 0) {
        extent = m_window.getExtent();
        glfwWaitEvents();
    }
    vkDeviceWaitIdle(m_device.device());
    if (m_swapChain == nullptr) {
        m_swapChain = std::make_unique<SwapChain>(m_device, 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");
    }
    if (const VkResult result = m_swapChain->submitCommandBuffers(&commandBuffer, &m_currentImageIndex); 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) % MAX_FRAMES_IN_FLIGHT;
}

void ven::Renderer::beginSwapChainRenderPass(const VkCommandBuffer commandBuffer) const
{
    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 = {.x=0, .y=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) const
{
    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);
}
```

## 8.84 /home/runner/work/VEngine/VEngine/src/shaders.cpp File Reference

```
#include <stdexcept>
#include <fstream>
#include <array>
#include "VEngine/Model.hpp"
#include "VEngine/Shaders.hpp"
```

Include dependency graph for shaders.cpp:




## 8.85 shaders.cpp

Go to the documentation of this file.

```
#include <stdexcept>
#include <fstream>
#include <array>

#include "VEngine/Model.hpp"
#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);
```

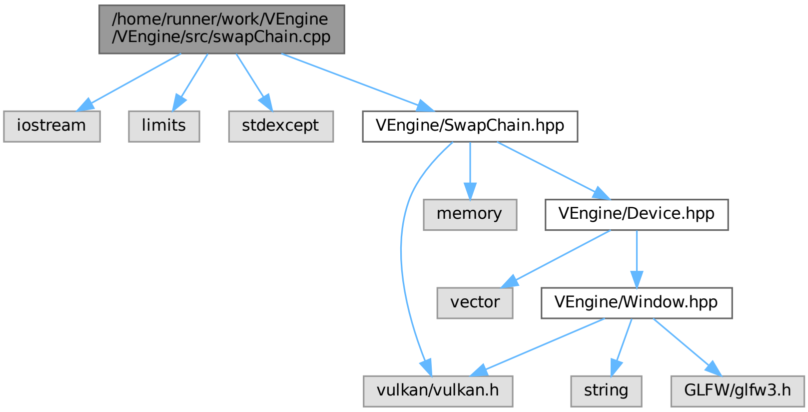
```cpp
}

std::vector<char> ven::Shaders::readFile(const std::string &filename) {
    std::ifstream file(filename, std::ios::binary | std::ios::ate);
    if (!file.is_open()) {
        throw std::runtime_error("failed to open file!");
    }

    const long int fileSize = file.tellg();
    std::vector<char> buffer(static_cast<long unsigned int>(fileSize));
    file.seekg(0);
    file.read(buffer.data(), fileSize);
    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);

    std::array<VkPipelineShaderStageCreateInfo, 2> shaderStages{};
    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.data();
    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");
    }
}

```

```cpp
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; // to enable later (VK_CULL_MODE_BACK_BIT) back-face culling
    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();
}
```

## 8.86 /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:




## 8.87 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 : m_swapChainImageViews) {
        vkDestroyImageView(m_device.device(), imageView, nullptr);
    }
    m_swapChainImageViews.clear();

    if (m_swapChain != nullptr) {
        vkDestroySwapchainKHR(m_device.device(), m_swapChain, nullptr);
        m_swapChain = nullptr;
    }

    for (size_t i = 0; i < m_depthImages.size(); i++) {
        vkDestroyImageView(m_device.device(), m_depthImageViews[i], nullptr);
        vkDestroyImage(m_device.device(), m_depthImages[i], nullptr);
        vkFreeMemory(m_device.device(), m_depthImageMemory[i], nullptr);
    }

    for (VkFramebuffer_T *framebuffer : m_swapChainFrameBuffers) {
        vkDestroyFramebuffer(m_device.device(), framebuffer, nullptr);
    }

    vkDestroyRenderPass(m_device.device(), m_renderPass, nullptr);

    // cleanup synchronization objects
    for (size_t i = 0; i < MAX_FRAMES_IN_FLIGHT; i++) {
        vkDestroySemaphore(m_device.device(), m_renderFinishedSemaphores[i], nullptr);
        vkDestroySemaphore(m_device.device(), m_imageAvailableSemaphores[i], nullptr);
        vkDestroyFence(m_device.device(), m_inFlightFences[i], nullptr);
    }
```

```cpp
}

void ven::SwapChain::init()
{
    createSwapChain();
    createImageViews();
    createRenderPass();
    createDepthResources();
    createFrameBuffers();
    createSyncObjects();
}

VkResult ven::SwapChain::acquireNextImage(uint32_t *imageIndex) const
{
    vkWaitForFences(m_device.device(), 1, &m_inFlightFences[m_currentFrame], VK_TRUE, std::numeric_limits<uint64_t>::max());

    return vkAcquireNextImageKHR(m_device.device(), m_swapChain, std::numeric_limits<uint64_t>::max(), m_imageAvailableSemaphores[m_currentFrame], VK_NULL_HANDLE, imageIndex);;
}

VkResult ven::SwapChain::submitCommandBuffers(const VkCommandBuffer *buffers, const uint32_t *imageIndex)
{
    if (m_imagesInFlight[*imageIndex] != VK_NULL_HANDLE) {
        vkWaitForFences(m_device.device(), 1, &m_imagesInFlight[*imageIndex], VK_TRUE, UINT64_MAX);
    }
    m_imagesInFlight[*imageIndex] = m_inFlightFences[m_currentFrame];

    VkSubmitInfo submitInfo = {};
    submitInfo.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;

    const std::array<VkSemaphore, 1> waitSemaphores = {m_imageAvailableSemaphores[m_currentFrame]};
    constexpr std::array<VkPipelineStageFlags, 1> waitStages = {VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT};
    submitInfo.waitSemaphoreCount = 1;
    submitInfo.pWaitSemaphores = waitSemaphores.data();
    submitInfo.pWaitDstStageMask = waitStages.data();

    submitInfo.commandBufferCount = 1;
    submitInfo.pCommandBuffers = buffers;

    const std::array<VkSemaphore, 1> signalSemaphores = {m_renderFinishedSemaphores[m_currentFrame]};
    submitInfo.signalSemaphoreCount = 1;
    submitInfo.pSignalSemaphores = signalSemaphores.data();

    vkResetFences(m_device.device(), 1, &m_inFlightFences[m_currentFrame]);
    if (vkQueueSubmit(m_device.graphicsQueue(), 1, &submitInfo, m_inFlightFences[m_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.data();

    const std::array<VkSwapchainKHR, 1> swapChains = {m_swapChain};
    presentInfo.swapchainCount = 1;
    presentInfo.pSwapchains = swapChains.data();

    presentInfo.pImageIndices = imageIndex;

    const VkResult result = vkQueuePresentKHR(m_device.presentQueue(), &presentInfo);

    m_currentFrame = (m_currentFrame + 1) % MAX_FRAMES_IN_FLIGHT;

    return result;
}

void ven::SwapChain::createSwapChain()
{
    const auto [capabilities, formats, presentModes] = m_device.getSwapChainSupport();

    const auto [format, colorSpace] = chooseSwapSurfaceFormat(formats);
    const VkPresentModeKHR presentMode = chooseSwapPresentMode(presentModes);
    const VkExtent2D extent = chooseSwapExtent(capabilities);

    uint32_t imageCount = capabilities.minImageCount + 1;
    if (capabilities.maxImageCount > 0 && imageCount > capabilities.maxImageCount) {
        imageCount = capabilities.maxImageCount;
    }

    VkSwapchainCreateInfoKHR createInfo = {};
    createInfo.sType = VK_STRUCTURE_TYPE_SWAPCHAIN_CREATE_INFO_KHR;
    createInfo.surface = m_device.surface();
```

```
    createInfo.minImageCount = imageCount;
    createInfo.imageFormat = format;
    createInfo.imageColorSpace = colorSpace;
    createInfo.imageExtent = extent;
    createInfo.imageArrayLayers = 1;
    createInfo.imageUsage = VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT;

    const auto [graphicsFamily, presentFamily, graphicsFamilyHasValue, presentFamilyHasValue] =
m_device.findPhysicalQueueFamilies();
    const std::array<uint32_t, 2> queueFamilyIndices = {graphicsFamily, presentFamily};

    if (graphicsFamily != presentFamily) {
        createInfo.imageSharingMode = VK_SHARING_MODE_CONCURRENT;
        createInfo.queueFamilyIndexCount = 2;
        createInfo.pQueueFamilyIndices = queueFamilyIndices.data();
    } else {
        createInfo.imageSharingMode = VK_SHARING_MODE_EXCLUSIVE;
        createInfo.queueFamilyIndexCount = 0;      // Optional
        createInfo.pQueueFamilyIndices = nullptr;  // Optional
    }

    createInfo.preTransform = capabilities.currentTransform;
    createInfo.compositeAlpha = VK_COMPOSITE_ALPHA_OPAQUE_BIT_KHR;

    createInfo.presentMode = presentMode;
    createInfo.clipped = VK_TRUE;

    createInfo.oldSwapchain = m_oldSwapChain == nullptr ? VK_NULL_HANDLE :
m_oldSwapChain->m_swapChain;

    if (vkCreateSwapchainKHR(m_device.device(), &createInfo, nullptr, &m_swapChain) != VK_SUCCESS) {
        throw std::runtime_error("failed to create swap chain!");
    }

    vkGetSwapchainImagesKHR(m_device.device(), m_swapChain, &imageCount, nullptr);
    m_swapChainImages.resize(imageCount);
    vkGetSwapchainImagesKHR(m_device.device(), m_swapChain, &imageCount, m_swapChainImages.data());

    m_swapChainImageFormat = format;
    m_swapChainExtent = extent;
}

void ven::SwapChain::createImageViews()
{
    m_swapChainImageViews.resize(m_swapChainImages.size());
    for (size_t i = 0; i < m_swapChainImages.size(); i++) {
        VkImageViewCreateInfo viewInfo{};
        viewInfo.sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO;
        viewInfo.image = m_swapChainImages[i];
        viewInfo.viewType = VK_IMAGE_VIEW_TYPE_2D;
        viewInfo.format = m_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(m_device.device(), &viewInfo, nullptr, &m_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(m_device.device(), &renderPassInfo, nullptr, &m_renderPass) != VK_SUCCESS) {
        throw std::runtime_error("failed to create render pass!");
    }
}

void ven::SwapChain::createFrameBuffers()
{
    m_swapChainFrameBuffers.resize(imageCount());
    for (size_t i = 0; i < imageCount(); i++) {
        std::array<VkImageView, 2> attachments = {m_swapChainImageViews[i], m_depthImageViews[i]};

        const auto [width, height] = getSwapChainExtent();
        VkFramebufferCreateInfo framebufferInfo = {};
        framebufferInfo.sType = VK_STRUCTURE_TYPE_FRAMEBUFFER_CREATE_INFO;
        framebufferInfo.renderPass = m_renderPass;
        framebufferInfo.attachmentCount = static_cast<uint32_t>(attachments.size());
        framebufferInfo.pAttachments = attachments.data();
        framebufferInfo.width = width;
        framebufferInfo.height = height;
        framebufferInfo.layers = 1;

        if (vkCreateFramebuffer(m_device.device(), &framebufferInfo, nullptr, &m_swapChainFrameBuffers[i]) != VK_SUCCESS) {
            throw std::runtime_error("failed to create framebuffer!");
        }
    }
}

void ven::SwapChain::createDepthResources()
{
    const VkFormat depthFormat = findDepthFormat();
    const auto [width, height] = getSwapChainExtent();

    m_swapChainDepthFormat = depthFormat;
    m_depthImages.resize(imageCount());
    m_depthImageMemory.resize(imageCount());
    m_depthImageViews.resize(imageCount());

    for (size_t i = 0; i < m_depthImages.size(); i++) {
        VkImageCreateInfo imageInfo{};
        imageInfo.sType = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
        imageInfo.imageType = VK_IMAGE_TYPE_2D;
        imageInfo.extent.width = width;
        imageInfo.extent.height = 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;
```

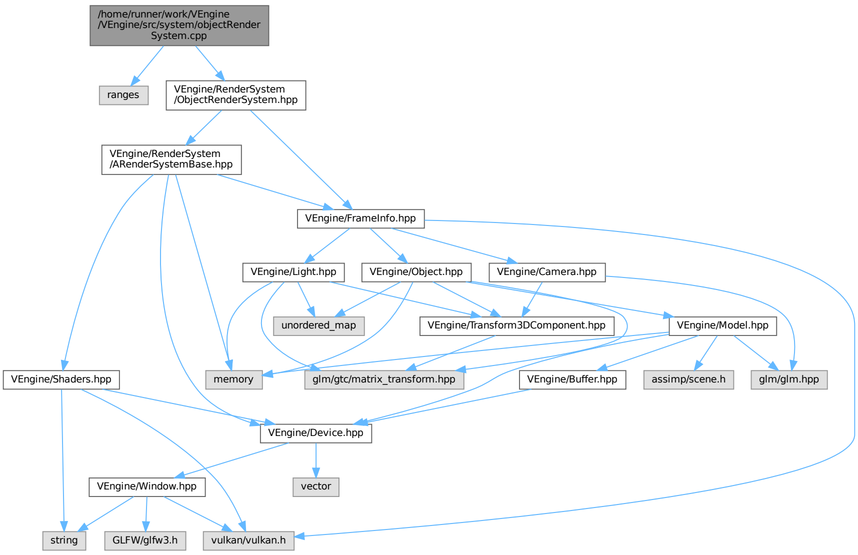
```
        imageInfo.samples = VK_SAMPLE_COUNT_1_BIT;
        imageInfo.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
        imageInfo.flags = 0;

        m_device.createImageWithInfo(imageInfo, VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT, m_depthImages[i], m_depthImageMemory[i]);

        VkImageViewCreateInfo viewInfo{};
        viewInfo.sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO;
        viewInfo.image = m_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(m_device.device(), &viewInfo, nullptr, &m_depthImageViews[i]) != VK_SUCCESS) {
            throw std::runtime_error("failed to create texture image view!");
        }
    }
}

void ven::SwapChain::createSyncObjects()
{
    m_imageAvailableSemaphores.resize(MAX_FRAMES_IN_FLIGHT);
    m_renderFinishedSemaphores.resize(MAX_FRAMES_IN_FLIGHT);
    m_inFlightFences.resize(MAX_FRAMES_IN_FLIGHT);
    m_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(m_device.device(), &semaphoreInfo, nullptr, &m_imageAvailableSemaphores[i]) != VK_SUCCESS ||
            vkCreateSemaphore(m_device.device(), &semaphoreInfo, nullptr, &m_renderFinishedSemaphores[i]) != VK_SUCCESS ||
            vkCreateFence(m_device.device(), &fenceInfo, nullptr, &m_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 = m_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 m_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);
}
```

## 8.88 /home/runner/work/VEngine/VEngine/src/system/objectRender↩ System.cpp File Reference

```
#include <ranges>
#include "VEngine/RenderSystem/ObjectRenderSystem.hpp"
```

Include dependency graph for objectRenderSystem.cpp:




## 8.89 objectRenderSystem.cpp

Go to the documentation of this file.

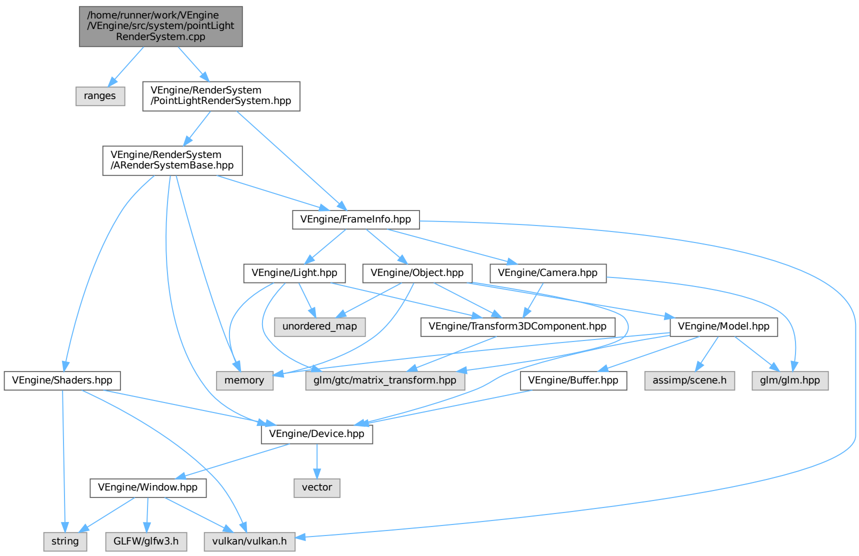
```
#include <ranges>

#include "VEngine/RenderSystem/ObjectRenderSystem.hpp"

void ven::ObjectRenderSystem::render(const FrameInfo &frameInfo) const
{
    getShaders()->bind(frameInfo.commandBuffer);

    vkCmdBindDescriptorSets(frameInfo.commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS,
getPipelineLayout(), 0, 1, &frameInfo.globalDescriptorSet, 0, nullptr);
```

```
    for (const Object& object : frameInfo.objects | std::views::values) {
        if (object.getModel() == nullptr) { continue; }
        const ObjectPushConstantData push{
            .modelMatrix = object.transform3D.mat4(),
            .normalMatrix = object.transform3D.normalMatrix()
        };
        vkCmdPushConstants(frameInfo.commandBuffer, getPipelineLayout(), VK_SHADER_STAGE_VERTEX_BIT |
 VK_SHADER_STAGE_FRAGMENT_BIT, 0, sizeof(ObjectPushConstantData), &push);
        object.getModel()->bind(frameInfo.commandBuffer);
        object.getModel()->draw(frameInfo.commandBuffer);
    }
}
```

## 8.90 /home/runner/work/VEngine/VEngine/src/system/pointLight↩ RenderSystem.cpp File Reference

```
#include <ranges>
#include "VEngine/RenderSystem/PointLightRenderSystem.hpp"
```

Include dependency graph for pointLightRenderSystem.cpp:




## 8.91 pointLightRenderSystem.cpp

Go to the documentation of this file.

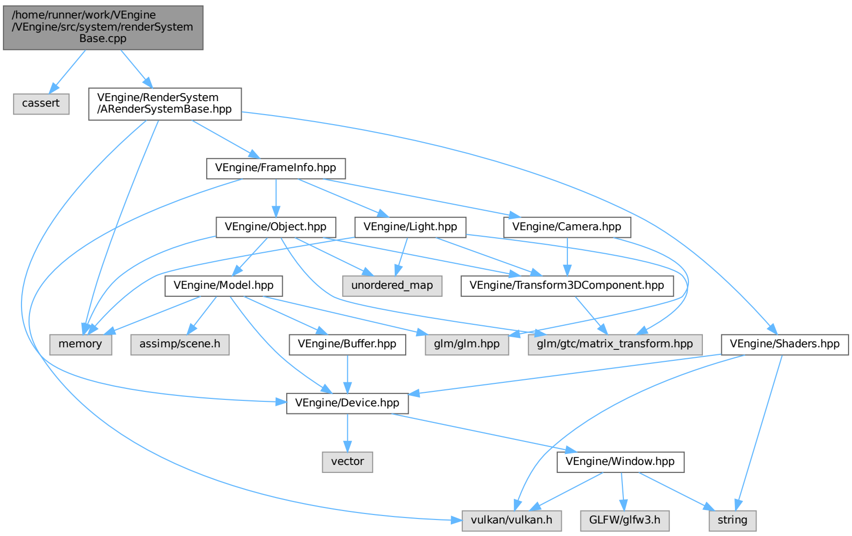
```
#include <ranges>

#include "VEngine/RenderSystem/PointLightRenderSystem.hpp"

void ven::PointLightRenderSystem::render(const FrameInfo &frameInfo) const
{
    getShaders()->bind(frameInfo.commandBuffer);
    vkCmdBindDescriptorSets(frameInfo.commandBuffer, VK_PIPELINE_BIND_POINT_GRAPHICS,
 getPipelineLayout(), 0, 1, &frameInfo.globalDescriptorSet, 0, nullptr);

    for (const Light &light : frameInfo.lights | std::views::values) {
        const LightPushConstantData push{
            .position = glm::vec4(light.transform3D.translation, 1.F),
            .color = light.color,
            .radius = light.transform3D.scale.x
        };
```

```
        vkCmdPushConstants(frameInfo.commandBuffer, getPipelineLayout(), VK_SHADER_STAGE_VERTEX_BIT |
VK_SHADER_STAGE_FRAGMENT_BIT, 0, sizeof(LightPushConstantData), &push);
        vkCmdDraw(frameInfo.commandBuffer, 6, 1, 0, 0);
    }
}

void ven::PointLightRenderSystem::update(const FrameInfo &frameInfo, GlobalUbo &ubo)
{
    const glm::mat4 rotateLight = rotate(glm::mat4(1.F), frameInfo.frameTime, {0.F, -1.F, 0.F});
    uint16_t lightIndex = 0;

    for (Light &light : frameInfo.lights | std::views::values) {
        assert(lightIndex < MAX_LIGHTS && "Too many lights");
        light.transform3D.translation = glm::vec3(rotateLight *
glm::vec4(light.transform3D.translation, 1.F));
        ubo.pointLights.at(lightIndex).position = glm::vec4(light.transform3D.translation, 1.F);
        ubo.pointLights.at(lightIndex).color = light.color;
        lightIndex++;
    }
    ubo.numLights = lightIndex;
}
```

## 8.92 /home/runner/work/VEngine/VEngine/src/system/renderSystemBase.cpp File Reference

```
#include <cassert>
#include "VEngine/RenderSystem/ARenderSystemBase.hpp"
```

Include dependency graph for renderSystemBase.cpp:




## 8.93 renderSystemBase.cpp

[Go to the documentation of this file.](#)

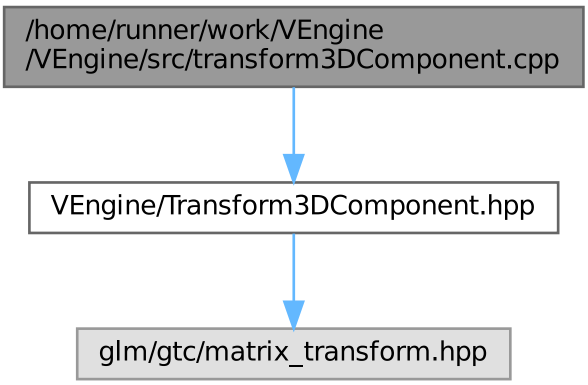
```
#include <cassert>

#include "VEngine/RenderSystem/ARenderSystemBase.hpp"

void ven::ARenderSystemBase::createPipelineLayout(const VkDescriptorSetLayout globalSetLayout, const
uint32_t pushConstantSize)
{
    VkPushConstantRange pushConstantRange{};
    pushConstantRange.stageFlags = VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT;
```

```
    pushConstantRange.offset = 0;
    pushConstantRange.size = pushConstantSize;

    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::ARenderSystemBase::createPipeline(const VkRenderPass renderPass, const std::string &shadersVertPath, const std::string &shadersFragPath, const bool isLight)
{
    assert(m_pipelineLayout && "Cannot create pipeline before pipeline layout");
    PipelineConfigInfo pipelineConfig{};
    Shaders::defaultPipelineConfigInfo(pipelineConfig);
    if (isLight) {
        pipelineConfig.attributeDescriptions.clear();
        pipelineConfig.bindingDescriptions.clear();
    }
    pipelineConfig.renderPass = renderPass;
    pipelineConfig.pipelineLayout = m_pipelineLayout;
    m_shaders = std::make_unique<Shaders>(m_device, shadersVertPath, shadersFragPath, pipelineConfig);
}
```

## 8.94 /home/runner/work/VEngine/VEngine/src/transform3DComponent.cpp File Reference

```
#include "VEngine/Transform3DComponent.hpp"
```

Include dependency graph for transform3DComponent.cpp:




## 8.95 transform3DComponent.cpp

Go to the documentation of this file.

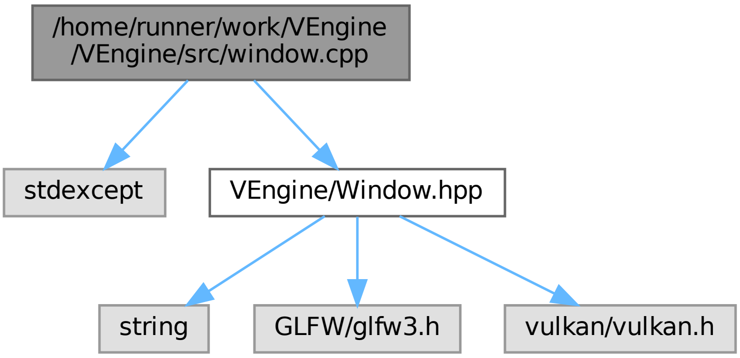
```
#include "VEngine/Transform3DComponent.hpp"
```

```

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)
            }
    };
}
```

## 8.96 /home/runner/work/VEngine/VEngine/src/window.cpp File Reference

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

Include dependency graph for window.cpp:




## 8.97 window.cpp

Go to the documentation of this file.

```
#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);
}

void ven::Window::setFullscreen(const bool fullscreen, const uint32_t width, const uint32_t height)
{
    GLFWmonitor* primaryMonitor = glfwGetPrimaryMonitor();
    const GLFWvidmode* mode = glfwGetVideoMode(primaryMonitor);

    /*
    if (fullscreen) {
```

```
        glfwSetWindowMonitor(m_window, primaryMonitor, 0, 0, mode->width, mode->height,
mode->refreshRate);
    } else {
        // To restore a window that was originally windowed to its original size and position,
        // save these before making it full screen and then pass them in as above
        glfwSetWindowMonitor(m_window, nullptr, 0, 0, static_cast<int>(width),
static_cast<int>(height), mode->refreshRate);

    }

    m_width = width;
    m_height = height;
    */
}
```

# Index