LLGL

1.00 Alpha

Generated by Doxygen 1.8.11

Contents

Chapter 1

LLGL 1.00 Alpha Documentation

LLGL (Low Level Graphics Library)

Overview

- **Version**: 1.00 Alpha
- **License**: 3-Clause BSD License

Progress

- **OpenGL Renderer**: ~70% done
- **Direct3D 12 Renderer**: ~5% done
- **Direct3D 11 Renderer**: not started yet
- **Vulkan Renderer**: not started yet

Getting Started

```
#include <LLGL/LLGL.h>

int main()
{
    // Create a window to render into
    LLGL::WindowDescriptor windowDesc;

    windowDesc.title    = L"LLGL Example";
    windowDesc.visible  = true;
    windowDesc.centered = true;
    windowDesc.width    = 640;
    windowDesc.height   = 480;

    auto window = LLGL::Window::Create(windowDesc);

    // Add keyboard/mouse event listener
    auto input = std::make_shared<LLGL::Input>();
    window->AddEventListener(input);

    //TO BE CONTINUED ...

    // Main loop
    while (window->ProcessEvents() && !input->KeyPressed(LLGL::Key::Escape))
    {

        // Draw with OpenGL, or Direct3D, or Vulkan, or whatever ...

    }

    return 0;
}
```

Thin Abstraction Layer

```
// Interface:
RenderContext::DrawIndexed(unsigned int numVertices, unsigned int firstIndex);

// OpenGL Implementation:
void GLRenderContext::DrawIndexed(unsigned int numVertices, unsigned int firstIndex)
{
    glDrawElements(
        renderState_.drawMode,
        static_cast<GLsizei>(numVertices),
        renderState_.indexBufferDataType,
        (reinterpret_cast<const GLvoid*>(firstIndex * renderState_.indexBufferStride))
    );
}

// Direct3D 11 Implementation
void D3D11RenderContext::DrawIndexed(unsigned int numVertices, unsigned int firstIndex)
{
    context_->DrawIndexed(numVertices, 0, firstIndex);
}

// Direct3D 12 Implementation
void D3D12RenderContext::DrawIndexed(unsigned int numVertices, unsigned int firstIndex)
{
    commandList_->DrawIndexedInstanced(numVertices, 1, firstIndex, 0, 0);
}

// Vulkan Implementation
void VKRenderContext::DrawIndexed(unsigned int numVertices, unsigned int firstIndex)
{
    vkCmdDrawIndexed(commandBuffer_, numVertices, 1, firstIndex, 0, 0);
}
```

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

Class Documentation

4.1 LLGL::AntiAliasingDescriptor Struct Reference

Public Attributes

- bool enabled = false

  Specifies whether multi-sampling is enabled or disabled. By default disabled.
- unsigned int samples = 1

  Number of samples used for multi-sampling. By default 1.

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

- RenderContextDescriptor.h

4.2 LLGL::BlendDescriptor Struct Reference

Blending state descriptor structure.

`#include <GraphicsPipelineFlags.h>`

Public Attributes

- bool blendEnabled = false

  Specifies whether blending is enabled or disabled. This applies to all blending targets.
- std::vector< BlendTargetDescriptor > targets

  Render-target blend states. A maximum of 8 targets is supported. Further targets will be ignored.

4.2.1 Detailed Description

Blending state descriptor structure.

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

- GraphicsPipelineFlags.h

4.3 LLGL::BlendTargetDescriptor Struct Reference

Blend target state descriptor structure.

`#include <GraphicsPipelineFlags.h>`

Public Attributes

- BlendOp srcColor = BlendOp::SrcAlpha
  Source color blending operation.
- BlendOp destColor = BlendOp::InvSrcAlpha
  Destination color blending operation.
- BlendArithmetic colorArithmetic = BlendArithmetic::Add
  Color blending arithmetic.
- BlendOp srcAlpha = BlendOp::SrcAlpha
  Source alpha blending operation.
- BlendOp destAlpha = BlendOp::InvSrcAlpha
  Destination alpha blending operation.
- BlendArithmetic alphaArithmetic = BlendArithmetic::Add
  Alpha blending arithmetic.
- ColorRGBAb colorMask
  Specifies which color components are enabled for writing. By default (true, true, true, true).

4.3.1 Detailed Description

Blend target state descriptor structure.

4.3.2 Member Data Documentation

4.3.2.1 BlendArithmetic LLGL::BlendTargetDescriptor::alphaArithmetic = BlendArithmetic::Add

Alpha blending arithmetic.

Note

Only supported with: Direct3D 11, Direct3D 12.

4.3.2.2 BlendArithmetic LLGL::BlendTargetDescriptor::colorArithmetic = BlendArithmetic::Add

Color blending arithmetic.

Note

Only supported with: Direct3D 11, Direct3D 12.

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

- GraphicsPipelineFlags.h

4.4 LLGL::ClearBuffersFlags Struct Reference

Render context clear buffer flags.

`#include <RenderContextFlags.h>`

Public Types

- enum { **Color** = (1 << 0), **Depth** = (1 << 1), **Stencil** = (1 << 2) }

4.4.1 Detailed Description

Render context clear buffer flags.

See also

RenderContext::ClearBuffers

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

- RenderContextFlags.h

4.5 LLGL::Color< T, N > Class Template Reference

Base color class with N components.

`#include <Color.h>`

Public Member Functions

- **Color** (const Color< T, N > &rhs)
- **Color** (Gs::UninitializeTag)
- Color< T, N > & **operator+=** (const Color< T, N > &rhs)
- Color< T, N > & **operator-=** (const Color< T, N > &rhs)
- Color< T, N > & **operator*=** (const Color< T, N > &rhs)
- Color< T, N > & **operator/=** (const Color< T, N > &rhs)
- Color< T, N > & **operator*=** (const T &rhs)
- Color< T, N > & **operator/=** (const T &rhs)
- T & operator[ ] (std::size_t component)

  Returns the specified vector component.
- const T & operator[ ] (std::size_t component) const

  Returns the specified vector component.
- Color< T, N > **operator-** () const
- template<typename C >
  Color< C, N > Cast () const
- T * Ptr ()

  Returns a pointer to the first element of this vector.
- const T * Ptr () const

  Returns a constant pointer to the first element of this vector.

Static Public Attributes

- static const std::size_t components = N

  Specifies the number of vector components.

4.5.1 Detailed Description

template<typename T, std::size_t N>
class LLGL::Color< T, N >

Base color class with N components.

Template Parameters

T	Specifies the data type of the vector components. This should be a primitive data type such as float, double, int etc.
N	Specifies the number of components. There are specialized templates for N = 3, and 4.

4.5.2 Member Function Documentation

4.5.2.1 template<typename T, std::size_t N> template<typename C > Color<C, N> LLGL::Color< T, N >::Cast ( ) const [inline]

Returns a type casted instance of this vector.

Template Parameters

C	Specifies the static cast type.

4.5.2.2 template<typename T, std::size_t N> T& LLGL::Color< T, N >::operator[ ] (std::size_t *component*) [inline]

Returns the specified vector component.

Parameters

in	*component*	Specifies the vector component index. This must be in the range [0, N).

4.5.2.3 template<typename T, std::size_t N> const T& LLGL::Color< T, N >::operator[ ] (std::size_t *component*) const [inline]

Returns the specified vector component.

Parameters

in	*component*	Specifies the vector component index. This must be in the range [0, N).

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

- Color.h

4.6 LLGL::Color< T, 3u > Class Template Reference

RGB color class with components: r, g, and b.

`#include <ColorRGB.h>`

Public Member Functions

- **Color** (const Color< T, 3 > &rhs)
- **Color** (const T &scalar)
- **Color** (const T &r, const T &g, const T &b)
- **Color** (Gs::UninitializeTag)
- Color< T, 3 > & **operator+=** (const Color< T, 3 > &rhs)
- Color< T, 3 > & **operator-=** (const Color< T, 3 > &rhs)
- Color< T, 3 > & **operator*=** (const Color< T, 3 > &rhs)
- Color< T, 3 > & **operator/=** (const Color< T, 3 > &rhs)
- Color< T, 3 > & **operator*=** (const T &rhs)
- Color< T, 3 > & **operator/=** (const T &rhs)
- Color< T, 3 > **operator-** () const
- T & operator[ ] (std::size_t component)

  Returns the specified color component.
- const T & operator[ ] (std::size_t component) const

  Returns the specified color component.
- template<typename C >
  Color< C, 3 > Cast () const

  Returns a type casted instance of this color.
- T * Ptr ()

  Returns a pointer to the first element of this color.
- const T * Ptr () const

  Returns a constant pointer to the first element of this color.

Public Attributes

- T **r**
- T **g**
- T **b**

Static Public Attributes

- static const std::size_t components = 3

  Specifies the number of color components.

4.6.1 Detailed Description

template<typename T>
class LLGL::Color< T, 3u >

RGB color class with components: r, g, and b.

Remarks

Color components are default initialized with their maximal value, i.e. for floating-points, the initial value is 1.0, because this its maximal color value, but for unsigned-bytes, the initial value is 255.

4.6.2 Member Function Documentation

4.6.2.1 template<typename T > template<typename C > Color<C, 3> LLGL::Color< T, 3u >::Cast ( ) const [inline]

Returns a type casted instance of this color.

Remarks

All color components will be scaled to the range of the new color type.

Template Parameters

C	Specifies the static cast type.

4.6.2.2 template<typename T > T& LLGL::Color< T, 3u >::operator[ ] (std::size_t *component*) [inline]

Returns the specified color component.

Parameters

in	*component*	Specifies the color component index. This must be 0, 1, or 2.

4.6.2.3 template<typename T > const T& LLGL::Color< T, 3u >::operator[ ] (std::size_t *component*) const [inline]

Returns the specified color component.

Parameters

in	*component*	Specifies the color component index. This must be 0, 1, or 2.

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

- ColorRGB.h

4.7 LLGL::Color< T, 4u > Class Template Reference

RGBA color class with components: r, g, b, and a.

`#include <ColorRGBA.h>`

Public Member Functions

- **Color** (const Color< T, 4 > &rhs)
- **Color** (const T &brightness)
- **Color** (const T &r, const T &g, const T &b)
- **Color** (const T &r, const T &g, const T &b, const T &a)
- **Color** (Gs::UninitializeTag)
- Color< T, 4 > & **operator+=** (const Color< T, 4 > &rhs)
- Color< T, 4 > & **operator-=** (const Color< T, 4 > &rhs)
- Color< T, 4 > & **operator*=** (const Color< T, 4 > &rhs)
- Color< T, 4 > & **operator/=** (const Color< T, 4 > &rhs)
- Color< T, 4 > & **operator*=** (const T &rhs)
- Color< T, 4 > & **operator/=** (const T &rhs)
- Color< T, 4 > **operator-** () const
- T & operator[ ] (std::size_t component)

  Returns the specified color component.
- const T & operator[ ] (std::size_t component) const

  Returns the specified color component.
- template<typename C >
  Color< C, 4 > Cast () const

  Returns a type casted instance of this color.
- T * Ptr ()

  Returns a pointer to the first element of this color.
- const T * Ptr () const

  Returns a constant pointer to the first element of this color.

Public Attributes

- T **r**
- T **g**
- T **b**
- T **a**

Static Public Attributes

- static const std::size_t components = 4

  Specifies the number of color components.

4.7.1 Detailed Description

template<typename T>
class LLGL::Color< T, 4u >

RGBA color class with components: r, g, b, and a.

Remarks

Color components are default initialized with their maximal value, i.e. for floating-points, the initial value is 1.0, because this its maximal color value, but for unsigned-bytes, the initial value is 255.

4.7.2 Member Function Documentation

4.7.2.1 template<typename T > template<typename C > Color<C, 4> LLGL::Color< T, 4u >::Cast ( ) const [inline]

Returns a type casted instance of this color.

Remarks

All color components will be scaled to the range of the new color type.

Template Parameters

C	Specifies the static cast type.

4.7.2.2 template<typename T > T& LLGL::Color< T, 4u >::operator[ ] (std::size_t *component*) [inline]

Returns the specified color component.

Parameters

in	*component*	Specifies the color component index. This must be 0, 1, 2, or 3.

4.7.2.3 template<typename T > const T& LLGL::Color< T, 4u >::operator[ ] (std::size_t *component*) const [inline]

Returns the specified color component.

Parameters

in	*component*	Specifies the color component index. This must be 0, 1, 2, or 3.

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

- ColorRGBA.h

4.8 LLGL::ComputePipeline Class Reference

Compute pipeline interface.

`#include <ComputePipeline.h>`

4.8.1 Detailed Description

Compute pipeline interface.

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

- ComputePipeline.h

4.9 LLGL::ComputePipelineDescriptor Struct Reference

Compute pipeline descriptor structure.

`#include <ComputePipeline.h>`

Public Member Functions

- **ComputePipelineDescriptor** (ShaderProgram *shaderProgram)

Public Attributes

- ShaderProgram * shaderProgram = nullptr

  Pointer to the shader program for the compute pipeline.

4.9.1 Detailed Description

Compute pipeline descriptor structure.

4.9.2 Member Data Documentation

4.9.2.1 ShaderProgram* LLGL::ComputePipelineDescriptor::shaderProgram = nullptr

Pointer to the shader program for the compute pipeline.

Remarks

This must never be null when "RenderSystem::CreateComputePipeline" is called with this structure.

See also

RenderSystem::CreateComputePipeline
RenderSystem::CreateShaderProgram

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

- ComputePipeline.h

4.10 LLGL::RenderSystem::Configuration Struct Reference

Render system configuration structure.

`#include <RenderSystem.h>`

Public Attributes

- ColorRGBAub defaultTextureImageColor
  Default color for an uninitialized texture. The default value is white (255, 255, 255, 255).

4.10.1 Detailed Description

Render system configuration structure.

4.10.2 Member Data Documentation

4.10.2.1 ColorRGBAub LLGL::RenderSystem::Configuration::defaultTextureImageColor

Default color for an uninitialized texture. The default value is white (255, 255, 255, 255).

Remarks

This will be used for each "SetupTexture..." function (not the "WriteTexture..." functions), when no initial image data is specified.

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

- RenderSystem.h

4.11 LLGL::ConstantBuffer Class Reference

Constant buffer (also "Uniform Buffer Object") interface.

`#include <ConstantBuffer.h>`

4.11.1 Detailed Description

Constant buffer (also "Uniform Buffer Object") interface.

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

- ConstantBuffer.h

4.12 LLGL::ConstantBufferDescriptor Struct Reference

Constant buffer descriptor structure.

`#include <ConstantBuffer.h>`

Public Attributes

- std::string name

  Constant buffer name.
- unsigned int index = 0

  Index of the constant buffer within the respective shader.
- unsigned int size = 0

  Buffer size (in bytes).

4.12.1 Detailed Description

Constant buffer descriptor structure.

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

- ConstantBuffer.h

4.13 LLGL::RenderingProfiler::Counter Class Reference

Public Types

- using **ValueType** = unsigned int

Public Member Functions

- void **Inc** ()
- void **Inc** (ValueType value)
- void **Reset** ()
- ValueType **Count** () const
- **operator unsigned int** () const

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

- RenderingProfiler.h

4.14 LLGL::DepthDescriptor Struct Reference

Depth state descriptor structure.

`#include <GraphicsPipelineFlags.h>`

Public Attributes

- bool testEnabled = false

  Specifies whether the depth test is enabled or disabled. By default disabled.
- bool writeEnabled = false

  Specifies whether writing to the depth buffer is enabled or disabled. By default disabled.
- CompareOp compareOp = CompareOp::Less

  Specifies the depth test comparison function. By default CompareOp::Less.

4.14.1 Detailed Description

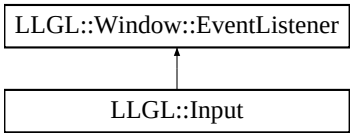
Depth state descriptor structure.

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

- GraphicsPipelineFlags.h

4.15 LLGL::Window::EventListener Class Reference

Inheritance diagram for LLGL::Window::EventListener:



Protected Member Functions

- virtual void **OnProcessEvents** (Window &sender)
- virtual void **OnKeyDown** (Window &sender, Key keyCode)
- virtual void **OnKeyUp** (Window &sender, Key keyCode)
- virtual void **OnDoubleClick** (Window &sender, Key keyCode)
- virtual void **OnChar** (Window &sender, wchar_t chr)
- virtual void **OnWheelMotion** (Window &sender, int motion)
- virtual void **OnLocalMotion** (Window &sender, const Point &position)
- virtual void **OnGlobalMotion** (Window &sender, const Point &motion)
- virtual void **OnResize** (Window &sender, const Size &clientAreaSize)
- virtual bool OnQuit (Window &sender)

  Returns true if the specified window can quit, i.e. "ProcessEvents" returns false from now on.

Friends

- class **Window**

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

- Window.h

4.16 LLGL::ShaderSource::GLSL Struct Reference

Shader source descriptor for GLSL.

```
#include <Shader.h>
```

Public Attributes

- const std::string & sourceCode

  Shader source code string.

4.16.1 Detailed Description

Shader source descriptor for GLSL.

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

- Shader.h

4.17 LLGL::GraphicsAPIDependentStateDescriptor Union Reference

Low-level graphics API dependent state descriptor union.

```
#include <RenderContextFlags.h>
```

Classes

- struct StateOpenGLDescriptor

Public Attributes

- struct LLGL::GraphicsAPIDependentStateDescriptor::StateOpenGLDescriptor **stateOpenGL**

4.17.1 Detailed Description

Low-level graphics API dependent state descriptor union.

See also

RenderContext::SetGraphicsAPIDependentState

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

- RenderContextFlags.h

4.18 LLGL::GraphicsPipeline Class Reference

Graphics pipeline interface.

```
#include <GraphicsPipeline.h>
```

4.18.1 Detailed Description

Graphics pipeline interface.

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

- GraphicsPipeline.h

4.19 LLGL::GraphicsPipelineDescriptor Struct Reference

Graphics pipeline descriptor structure.

```
#include <GraphicsPipelineFlags.h>
```

Public Attributes

- DepthDescriptor depth

  Specifies the depth state descriptor.
- StencilDescriptor stencil

  Specifies the stencil state descriptor.
- RasterizerDescriptor rasterizer

  Specifies the rasterizer state descriptor.
- BlendDescriptor blend

  Specifies the blending state descriptor.
- ShaderProgram * shaderProgram = nullptr

  Pointer to the shader program for the graphics pipeline.

4.19.1 Detailed Description

Graphics pipeline descriptor structure.

Remarks

This structure describes the entire graphics pipeline: viewports, depth-/ stencil-/ rasterizer-/ blend states, shader stages etc.

4.19.2 Member Data Documentation

4.19.2.1 ShaderProgram* LLGL::GraphicsPipelineDescriptor::shaderProgram = nullptr

Pointer to the shader program for the graphics pipeline.

Remarks

This must never be null when "RenderSystem::CreateGraphicsPipeline" is called with this structure.

See also

RenderSystem::CreateGraphicsPipeline
RenderSystem::CreateShaderProgram

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

- GraphicsPipelineFlags.h

4.20 LLGL::ShaderSource::HLSL Struct Reference

Shader source descriptor for HLSL.

```
#include <Shader.h>
```

Public Attributes

- const std::string & sourceCode

  Shader source code string.
- std::string entryPoint

  Shader entry point (this is the name of the shader main function).
- std::string target

  Shdaer version target (see https://msdn.microsoft.com/en-us/library/windows/desktop/jj215820(v=vs.↩85).aspx*).*
- int flags

  Optional compilation flags. This can be a bitwise OR combination of the 'ShaderCompileFlags' enumeration entries.

4.20.1 Detailed Description

Shader source descriptor for HLSL.

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

- Shader.h

4.21 LLGL::ImageDataDescriptor Struct Reference

Texture data descriptor structure.

`#include <TextureFlags.h>`

Public Member Functions

- **ImageDataDescriptor** (ColorFormat dataFormat, DataType dataType, const void *data)
- ImageDataDescriptor (ColorFormat dataFormat, const void *data, unsigned int compressedSize)

  Constructor for compressed image data.

Public Attributes

- ColorFormat dataFormat = ColorFormat::Gray

  Specifies the color format.
- DataType dataType = DataType::UInt8

  Speciifes the image data type. This must be DataType::UInt8 for compressed images.
- const void * data = nullptr

  Pointer to the image data source.
- unsigned int compressedSize = 0

  Specifies the size (in bytes) of the compressed image. This must be 0 for uncompressed images.

4.21.1 Detailed Description

Texture data descriptor structure.

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

- TextureFlags.h

4.22 LLGL::IndexBuffer Class Reference

Index buffer interface.

`#include <IndexBuffer.h>`

Public Member Functions

- const IndexFormat & **GetIndexFormat** () const

Protected Member Functions

- void **SetIndexFormat** (const IndexFormat &indexFormat)

4.22.1 Detailed Description

Index buffer interface.

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

- IndexBuffer.h

4.23 LLGL::IndexFormat Class Reference

Public Member Functions

- **IndexFormat** (const DataType dataType)
- DataType GetDataType () const

  Returns the data type of this index format.
- unsigned int GetFormatSize () const

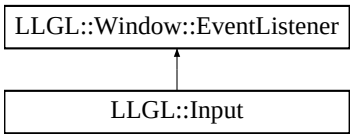
  Returns the size of this vertex format (in bytes).

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

- IndexFormat.h

4.24 LLGL::Input Class Reference

Inheritance diagram for LLGL::Input:



Public Member Functions

- bool KeyPressed (Key keyCode) const

  Returns true if the specified key is currently being pressed down.
- bool KeyDown (Key keyCode) const

  Returns true if the specified key was pressed down in the previous event processing.
- bool KeyUp (Key keyCode) const

  Returns true if the specified key was released in the previous event processing.
- bool KeyDoubleClick (Key keyCode) const

  Returns true if the specified key was double clicked.
- const Point & GetMousePosition () const

  Returns the local mouse position.
- const Point & GetMouseMotion () const

  Returns the global mouse motion.
- int GetWheelMotion () const

  Returns the mouse wheel motion.
- const std::wstring & GetEnteredChars () const

  Returns the entered characters.

Additional Inherited Members

4.24.1 Member Function Documentation

4.24.1.1 bool LLGL::Input::KeyDoubleClick (Key *keyCode*) const

Returns true if the specified key was double clicked.

Remarks

This can only be true for the key codes: Key::LButton, Key::RButton, and Key::MButton.

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

- Input.h

4.25 LLGL::NativeContextHandle Struct Reference

Linux native context handle structure.

`#include <LinuxNativeHandle.h>`

Public Attributes

- ::Display * **display**
- ::Window **parentWindow**
- ::XVisualInfo * **visual**
- ::Colormap **colorMap**
- int **screen**
- NSWindow * **parentWindow**
- HWND **parentWindow**

4.25.1 Detailed Description

Linux native context handle structure.

Win32 native context handle structure.

MacOS native context handle structure.

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

- LinuxNativeHandle.h
- MacOSNativeHandle.h
- Win32NativeHandle.h

4.26 LLGL::NativeHandle Struct Reference

Linux native handle structure.

`#include <LinuxNativeHandle.h>`

Public Attributes

- ::Display * **display**
- ::Window **window**
- ::XVisualInfo * **visual**
- NSWindow * **window**
- HWND **window**

4.26.1 Detailed Description

Linux native handle structure.

Win32 native handle structure.

MacOS native handle structure.

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

- LinuxNativeHandle.h
- MacOSNativeHandle.h
- Win32NativeHandle.h

4.27 LLGL::ProfileOpenGLDescriptor Struct Reference

Public Attributes

- bool extProfile = false

  Specifies whether an extended renderer profile is to be used. By default false.
- bool coreProfile = false

  Specifies whether to use 'OpenGL Core Profile', instead of 'OpenGL Compatibility Profile'. By default disbaled.
- bool debugDump = false

  Specifies whether the hardware renderer will produce debug dump. By default disabled.
- OpenGLVersion version = OpenGLVersion::OpenGL_Latest

  OpenGL version to create the render context with.

4.27.1 Member Data Documentation

4.27.1.1 bool LLGL::ProfileOpenGLDescriptor::coreProfile = false

Specifies whether to use 'OpenGL Core Profile', instead of 'OpenGL Compatibility Profile'. By default disbaled.

Remarks

This requires 'extProfile' to be enabled.

4.27.1.2 OpenGLVersion LLGL::ProfileOpenGLDescriptor::version = OpenGLVersion::OpenGL_Latest

OpenGL version to create the render context with.

Remarks

This required 'coreProfile' to be enabled.

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

- RenderContextDescriptor.h

4.28 LLGL::Query Class Reference

Query interface.

`#include <Query.h>`

4.28.1 Detailed Description

Query interface.

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

- Query.h

4.29 LLGL::RasterizerDescriptor Struct Reference

Rasterizer state descriptor structure.

`#include <GraphicsPipelineFlags.h>`

Public Attributes

- PolygonMode polygonMode = PolygonMode::Fill
  Polygon render mode. By default PolygonMode::Fill.
- CullMode **cullMode** = CullMode::Disabled
- int **depthBias** = 0
- float **depthBiasClamp** = 0.0f
- float **slopeScaledDepthBias** = 0.0f
- unsigned int samples = 1
  Number of samples for multi-sample anti-aliasing (MSAA).
- bool frontCCW = false
  If true, front facing polygons are in counter-clock-wise winding, otherwise in clock-wise winding.
- bool **depthClampEnabled** = false
- bool **scissorTestEnabled** = false
- bool **multiSampleEnabled** = false
- bool **antiAliasedLineEnabled** = false
- bool conservativeRasterization = false
  If ture, conservative rasterization is enabled.

4.29.1 Detailed Description

Rasterizer state descriptor structure.

4.29.2 Member Data Documentation

4.29.2.1 bool LLGL::RasterizerDescriptor::conservativeRasterization = false

If ture, conservative rasterization is enabled.

Note

Only supported with: Direct3D 12 (or OpenGL if the extension "GL_NV_conservative_raster" or "GL_INTE↩ L_conservative_rasterization" is supported).

See also

https://www.opengl.org/registry/specs/NV/conservative_raster.txt
https://www.opengl.org/registry/specs/INTEL/conservative_rasterization.↩ txt

4.29.2.2 unsigned int LLGL::RasterizerDescriptor::samples = 1

Number of samples for multi-sample anti-aliasing (MSAA).

See also

multiSampleEnabled

Note

Only supported with: Direct3D 11, Direct3D 12.

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

- GraphicsPipelineFlags.h

4.30 LLGL::RenderContext Class Reference

Render context interface.

`#include <RenderContext.h>`

Public Member Functions

- **RenderContext** (const RenderContext &)=delete
- RenderContext & **operator=** (const RenderContext &)=delete
- virtual void Present ()=0

  Presents the current frame on the screen.
- Window & GetWindow () const

  Returns the window which is used to draw all content.
- virtual void SetGraphicsAPIDependentState (const GraphicsAPIDependentStateDescriptor &state)=0
- virtual void SetVideoMode (const VideoModeDescriptor &videoModeDesc)

  Sets the new video mode for this render context.
- virtual void SetVsync (const VsyncDescriptor &vsyncDesc)=0

  Sets the new vertical-sychronization (Vsync) configuration for this render context.
- const VideoModeDescriptor & GetVideoMode () const

  Returns the video mode for this render context.
- virtual void SetViewports (const std::vector< Viewport > &viewports)=0

  Sets the specified viewports.
- virtual void SetScissors (const std::vector< Scissor > &scissors)=0

  Sets the specified scissor rectangles.
- virtual void SetClearColor (const ColorRGBAf &color)=0

  Sets the new value to clear the color buffer. By default black (0, 0, 0, 0).
- virtual void SetClearDepth (float depth)=0

  Sets the new value to clear the depth buffer with. By default 0.0.
- virtual void SetClearStencil (int stencil)=0

  Sets the new value to clear the stencil buffer. By default 0.
- virtual void ClearBuffers (long flags)=0

  Clears the specified frame buffers.
- virtual void SetVertexBuffer (VertexBuffer &vertexBuffer)=0

  Sets the active vertex buffer for subsequent drawing operations.
- virtual void SetIndexBuffer (IndexBuffer &indexBuffer)=0

  Sets the active index buffer for subsequent drawing operations.
- virtual void SetConstantBuffer (ConstantBuffer &constantBuffer, unsigned int slot)=0

  Sets the active constant buffer of the specified slot index for subsequent drawing and compute operations.
- virtual void SetStorageBuffer (StorageBuffer &storageBuffer, unsigned int slot)=0

  Sets the active storage buffer of the specified slot index for subsequent drawing and compute operations.
- virtual void * MapStorageBuffer (StorageBuffer &storageBuffer, const BufferCPUAccess access)=0

  Maps the specified storage buffer from GPU to CPU memory space.
- virtual void UnmapStorageBuffer ()=0

  Unmaps the previously mapped storage buffer.
- virtual void SetTexture (Texture &texture, unsigned int slot)=0

  Sets the active texture of the specified slot index for subsequent drawing and compute operations.
- virtual void GenerateMips (Texture &texture)=0

  Generates the MIP ("Multum in Parvo") maps for the specified texture.
- virtual void SetSampler (Sampler &sampler, unsigned int slot)=0

  Sets the active sampler of the specified slot index for subsequent drawing and compute operations.
- virtual void SetRenderTarget (RenderTarget &renderTarget)=0

  Sets the active render target.
- virtual void UnsetRenderTarget ()=0

  Unsets the previously set render target.
- virtual void SetGraphicsPipeline (GraphicsPipeline &graphicsPipeline)=0
- virtual void SetComputePipeline (ComputePipeline &computePipeline)=0

Sets the active compute pipeline state.

- virtual void BeginQuery (Query &query)=0

  Begins the specified query.
- virtual void EndQuery (Query &query)=0

  Ends the specified query.
- virtual bool QueryResult (Query &query, std::uint64_t &result)=0

  Queries the result of the specified Query object.
- virtual void SetPrimitiveTopology (const PrimitiveTopology topology)=0

  Sets the primitive topology for subsequent draw operations.
- virtual void Draw (unsigned int numVertices, unsigned int firstVertex)=0

  Draws the specified amount of primitives from the currently set vertex buffer.
- virtual void DrawIndexed (unsigned int numVertices, unsigned int firstIndex)=0
- virtual void DrawIndexed (unsigned int numVertices, unsigned int firstIndex, int vertexOffset)=0

  Draws the specified amount of primitives from the currently set vertex- and index buffers.
- virtual void DrawInstanced (unsigned int numVertices, unsigned int firstVertex, unsigned int numInstances)=0
- virtual void DrawInstanced (unsigned int numVertices, unsigned int firstVertex, unsigned int numInstances, unsigned int instanceOffset)=0

  Draws the specified amount of instances of primitives from the currently set vertex buffer.
- virtual void DrawIndexedInstanced (unsigned int numVertices, unsigned int numInstances, unsigned int first↩ Index)=0
- virtual void DrawIndexedInstanced (unsigned int numVertices, unsigned int numInstances, unsigned int first↩ Index, int vertexOffset)=0
- virtual void DrawIndexedInstanced (unsigned int numVertices, unsigned int numInstances, unsigned int first↩ Index, int vertexOffset, unsigned int instanceOffset)=0

  Draws the specified amount of instances of primitives from the currently set vertex- and index buffers.
- virtual void DispatchCompute (const Gs::Vector3ui &threadGroupSize)=0

  Dispachtes a compute command.
- virtual void SyncGPU ()=0

  Synchronizes the GPU, i.e. waits until the GPU has completed all pending commands.

Protected Member Functions

- void **SetWindow** (const std::shared_ptr< Window > &window, VideoModeDescriptor &videoModeDesc, const void *windowContext)

4.30.1 Detailed Description

Render context interface.

Remarks

The render context is the main interface for drawing and compute operations.

4.30.2 Member Function Documentation

4.30.2.1 virtual void LLGL::RenderContext::BeginQuery (Query & *query*) [pure virtual]

Begins the specified query.

Parameters

in	*query*	Specifies the query to begin with. This must be same query object as in the subsequent "EndQuery" function call, to end the query operation.

Remarks

The "BeginQuery" and "EndQuery" functions can be wrapped around any drawing and/or compute operation. This can an occlusion query for instance, which determines how many fragments have passed the depth test.

See also

RenderSystem::CreateQuery
EndQuery
QueryResult

4.30.2.2 virtual void LLGL::RenderContext::ClearBuffers (long *flags*) [pure virtual]

Clears the specified frame buffers.

Parameters

in	*flags*	Specifies the clear buffer flags. This can be a bitwise OR combination of the "ClearBuffersFlags" enumeration entries.

Remarks

To specify the clear values for each buffer use the respective "SetClear..." function

See also

ClearBuffersFlags
SetClearColor
SetClearDepth
SetClearStencil

4.30.2.3 virtual void LLGL::RenderContext::DispatchCompute (const Gs::Vector3ui & *threadGroupSize*) [pure virtual]

Dispachtes a compute command.

Parameters

in	*threadGroupSize*	Specifies the number of thread groups, where the number of threads per group is specified statically within the compute shader.

See also

SetComputePipeline

4.30.2.4 virtual void LLGL::RenderContext::Draw (unsigned int *numVertices*, unsigned int *firstVertex*) [pure virtual]

Draws the specified amount of primitives from the currently set vertex buffer.

Parameters

in	*numVertices*	Specifies the number of vertices to generate.
in	*firstVertex*	Specifies the zero-based offset of the first vertex from the vertex buffer.

4.30.2.5 virtual void LLGL::RenderContext::DrawIndexed (unsigned int *numVertices*, unsigned int *firstIndex*) [pure virtual]

See also

DrawIndexed(unsigned int, unsigned int, int)

4.30.2.6 virtual void LLGL::RenderContext::DrawIndexed (unsigned int *numVertices*, unsigned int *firstIndex*, int *vertexOffset*) [pure virtual]

Draws the specified amount of primitives from the currently set vertex- and index buffers.

Parameters

in	*numVertices*	Specifies the number of vertices to generate.
in	*firstIndex*	Specifies the zero-based offset of the first index from the index buffer.
in	*vertexOffset*	Specifies the base vertex offset (positive or negative) which is added to each index from the index buffer.

4.30.2.7 virtual void LLGL::RenderContext::DrawIndexedInstanced (unsigned int *numVertices*, unsigned int *numInstances*, unsigned int *firstIndex*) [pure virtual]

See also

DrawIndexedInstanced(unsigned int, unsigned int, unsigned int, int, unsigned int)

4.30.2.8 virtual void LLGL::RenderContext::DrawIndexedInstanced (unsigned int *numVertices*, unsigned int *numInstances*, unsigned int *firstIndex*, int *vertexOffset*) [pure virtual]

See also

DrawIndexedInstanced(unsigned int, unsigned int, unsigned int, int, unsigned int)

4.30.2.9 virtual void LLGL::RenderContext::DrawIndexedInstanced (unsigned int *numVertices*, unsigned int *numInstances*, unsigned int *firstIndex*, int *vertexOffset*, unsigned int *instanceOffset*) [pure virtual]

Draws the specified amount of instances of primitives from the currently set vertex- and index buffers.

Parameters

in	*numVertices*	Specifies the number of vertices to generate.
in	*numInstances*	Specifies the number of instances to generate.
in	*firstIndex*	Specifies the zero-based offset of the first index from the index buffer.
in	*vertexOffset*	Specifies the base vertex offset (positive or negative) which is added to each index from the index buffer.
in	*instanceOffset*	Specifies the zero-based instance offset which is added to each instance ID.

4.30.2.10 virtual void LLGL::RenderContext::DrawInstanced (unsigned int *numVertices*, unsigned int *firstVertex*, unsigned int *numInstances*) [pure virtual]

See also

DrawInstanced(unsigned int, unsigned int, unsigned int, unsigned int)

4.30.2.11 virtual void LLGL::RenderContext::DrawInstanced (unsigned int *numVertices*, unsigned int *firstVertex*, unsigned int *numInstances*, unsigned int *instanceOffset*) [pure virtual]

Draws the specified amount of instances of primitives from the currently set vertex buffer.

Parameters

in	*numVertices*	Specifies the number of vertices to generate.
in	*firstVertex*	Specifies the zero-based offset of the first vertex from the vertex buffer.
in	*numInstances*	Specifies the number of instances to generate.
in	*instanceOffset*	Specifies the zero-based instance offset which is added to each instance ID.

4.30.2.12 virtual void LLGL::RenderContext::EndQuery (Query & *query*) [pure virtual]

Ends the specified query.

See also

RenderSystem::CreateQuery
BeginQuery
QueryResult

4.30.2.13 virtual void LLGL::RenderContext::GenerateMips (Texture & *texture*) [pure virtual]

Generates the MIP ("Multum in Parvo") maps for the specified texture.

See also

https://developer.valvesoftware.com/wiki/MIP_Mapping

4.30.2.14 virtual void* LLGL::RenderContext::MapStorageBuffer (StorageBuffer & *storageBuffer,* const BufferCPUAccess *access*) [pure virtual]

Maps the specified storage buffer from GPU to CPU memory space.

Parameters

in	*storageBuffer*	Specifies the storage buffer which is to be mapped.
in	*access*	Specifies the CPU buffer access requirement, i.e. if the CPU can read and/or write the mapped memory.

Returns

Raw pointer to the mapped memory block. You should be aware of the storage buffer size, to not cause memory violations.

Exceptions

std::runtime_error	If a storage buffer is already being mapped.

See also

UnmapStorageBuffer

4.30.2.15 virtual bool LLGL::RenderContext::QueryResult (Query & *query,* std::uint64_t & *result*) [pure virtual]

Queries the result of the specified Query object.

Parameters

in,out	*query*	Specifies the Query object whose result is to be queried.
out	*result*	Specifies the output result.

Returns

True if the result is available, otherwise false in which case 'result' is not modified.

4.30.2.16 virtual void LLGL::RenderContext::SetComputePipeline (ComputePipeline & *computePipeline*) [pure virtual]

Sets the active compute pipeline state.

Parameters

in	*computePipeline*	Specifies the compuite pipeline state to set.

Remarks

This will set the compute shader states. A valid compute pipeline must always be set before any compute operation can be performed.

See also

RenderSystem::CreateComputePipeline

4.30.2.17 virtual void LLGL::RenderContext::SetConstantBuffer (ConstantBuffer & *constantBuffer,* unsigned int *slot*) [pure virtual]

Sets the active constant buffer of the specified slot index for subsequent drawing and compute operations.

Parameters

in	*constantBuffer*	Specifies the constant buffer to set. This must not be an unspecified constant buffer, i.e. it must be initialized with the "RenderSystem::SetupConstantBuffer" function.
in	*slot*	Specifies the slot index where to put the constant buffer.

See also

RenderSystem::SetupConstantBuffer

4.30.2.18 virtual void LLGL::RenderContext::SetGraphicsAPIDependentState (const GraphicsAPIDependentStateDescriptor & *state*) [pure virtual]

Sets a few low-level graphics API dependent states.

Remarks

This is mainly used to work around uniform render target behavior between different low-level graphics APIs such as OpenGL and Direct3D.

4.30.2.19 virtual void LLGL::RenderContext::SetGraphicsPipeline (GraphicsPipeline & *graphicsPipeline*) [pure virtual]

Sets the active graphics pipeline state.

Parameters

in	*graphicsPipeline*	Specifies the graphics pipeline state to set.

Remarks

This will set all blending-, rasterizer-, depth-, stencil-, and shader states. A valid graphics pipeline must always be set before any drawing operation can be performed.

See also

RenderSystem::CreateGraphicsPipeline

4.30.2.20 virtual void LLGL::RenderContext::SetIndexBuffer (IndexBuffer & *indexBuffer*) [pure virtual]

Sets the active index buffer for subsequent drawing operations.

Parameters

in	*indexBuffer*	Specifies the index buffer to set. This must not be an unspecified index buffer, i.e. it must be initialized with the "RenderSystem::SetupIndexBuffer" function.

Remarks

An active index buffer is only required for any "DrawIndexed" or "DrawIndexedInstanced" draw call.

See also

RenderSystem::SetupIndexBuffer

4.30.2.21 virtual void LLGL::RenderContext::SetPrimitiveTopology (const PrimitiveTopology *topology*) [pure virtual]

Sets the primitive topology for subsequent draw operations.

See also

PrimitiveTopology

4.30.2.22 virtual void LLGL::RenderContext::SetRenderTarget (RenderTarget & *renderTarget*) [pure virtual]

Sets the active render target.

Parameters

in	*renderTarget*	Specifies the render target to set.

Remarks

Subsequent drawing operations will be rendered into the textures that are attached to the specified render target.

See also

UnsetRenderTarget

4.30.2.23 virtual void LLGL::RenderContext::SetSampler (Sampler & *sampler,* unsigned int *slot*) [pure virtual]

Sets the active sampler of the specified slot index for subsequent drawing and compute operations.

Parameters

in	*sampler*	Specifies the sampler to set.
in	*slot*	Specifies the slot index where to put the sampler.

See also

RenderSystem::CreateSampler

4.30.2.24 virtual void LLGL::RenderContext::SetScissors (const std::vector< Scissor > & *scissors*) [pure virtual]

Sets the specified scissor rectangles.

Parameters

in	*scissors*	Specifies the list of scissor rectangles.

Remarks

This function behaves differently on the OpenGL render system, depending on the state configured with the "SetGraphicsAPIDependentState" function. If 'stateOpenGL.flipViewportVertical' is false, the origin of each scissor rectangle is on the upper-left (like for all other render systems). If 'stateOpenGL.flipViewportVertical' is true, the origin of each scissor rectangle is on the lower-left (this is useful when a render target is set).

See also

SetGraphicsAPIDependentState

4.30.2.25 virtual void LLGL::RenderContext::SetStorageBuffer (StorageBuffer & *storageBuffer,* unsigned int *slot*) [pure virtual]

Sets the active storage buffer of the specified slot index for subsequent drawing and compute operations.

Parameters

in	*storageBuffer*	Specifies the storage buffer to set. This must not be an unspecified storage buffer, i.e. it must be initialized with the "RenderSystem::SetupStorageBuffer" function.
in	*slot*	Specifies the slot index where to put the storage buffer.

See also

RenderSystem::SetupStorageBuffer

4.30.2.26 virtual void LLGL::RenderContext::SetTexture (Texture & *texture*, unsigned int *slot*) [pure virtual]

Sets the active texture of the specified slot index for subsequent drawing and compute operations.

Parameters

in	*texture*	Specifies the texture to set. This must not be an unspecified texture, i.e. it must be initialized with any of the "RenderSystem::SetupTexture..." functions.
in	*slot*	Specifies the slot index where to put the texture.

4.30.2.27 virtual void LLGL::RenderContext::SetVertexBuffer (VertexBuffer & *vertexBuffer*) [pure virtual]

Sets the active vertex buffer for subsequent drawing operations.

Parameters

in	*vertexBuffer*	Specifies the vertex buffer to set. This must not be an unspecified vertex buffer, i.e. it must be initialized with the "RenderSystem::SetupVertexBuffer" function.

See also

RenderSystem::SetupVertexBuffer

4.30.2.28 virtual void LLGL::RenderContext::SetViewports (const std::vector< Viewport > & *viewports*) [pure virtual]

Sets the specified viewports.

Parameters

in	*viewports*	Specifies the list of viewports.

Remarks

This function behaves differently on the OpenGL render system, depending on the state configured with the "SetGraphicsAPIDependentState" function. If 'stateOpenGL.flipViewportVertical' is false, the origin of each viewport is on the upper-left (like for all other render systems). If 'stateOpenGL.flipViewportVertical' is true, the origin of each viewport is on the lower-left (this is useful when a render target is set).

See also

SetGraphicsAPIDependentState

4.30.2.29 virtual void LLGL::RenderContext::UnmapStorageBuffer ( ) [pure virtual]

Unmaps the previously mapped storage buffer.

See also

MapStorageBuffer

4.30.2.30 virtual void LLGL::RenderContext::UnsetRenderTarget ( ) [pure virtual]

Unsets the previously set render target.

Remarks

Subsequent drawing operations will be rendered into the main framebuffer, which can then be presented onto the screen.

See also

SetRenderTarget

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

- RenderContext.h

4.31 LLGL::RenderContextDescriptor Struct Reference

Public Attributes

- VsyncDescriptor vsync
  Vertical-synchronization (Vsync) descriptor.
- AntiAliasingDescriptor antiAliasing
  Multi-sample anti-aliasing descriptor.
- VideoModeDescriptor videoMode
  Video mode descriptor.
- ProfileOpenGLDescriptor profileOpenGL
  OpenGL profile descriptor (to switch between compatability or core profile).
- DebugCallback debugCallback
  Debuging callback descriptor.

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

- RenderContextDescriptor.h

4.32 LLGL::RenderingCaps Struct Reference

Rendering capabilities structure.

`#include <RenderSystemFlags.h>`

Public Attributes

- ScreenOrigin screenOrigin = ScreenOrigin::UpperLeft

  Screen coordinate system origin.
- ClippingRange clippingRange = ClippingRange::ZeroToOne

  Clipping depth range.
- bool hasRenderTargets = false

  Specifies whether render targets (also "frame buffer objects") are supported.
- bool has3DTextures = false

  Specifies whether 3D textures are supported.
- bool hasCubeTextures = false

  Specifies whether cube textures are supported.
- bool hasTextureArrays = false

  Specifies whether 1D- and 2D array textures are supported.
- bool hasCubeTextureArrays = false

  Specifies whether cube array textures are supported.
- bool hasSamplers = false

  Specifies whether samplers are supported.
- bool hasConstantBuffers = false

  Specifies whether constant buffers (also "uniform buffer objects") are supported.
- bool hasStorageBuffers = false

  Specifies whether storage buffers (also "read/write buffers") are supported.
- bool hasUniforms = false

  Specifies whether individual shader uniforms are supported (typically only for OpenGL 2.0+).
- bool hasGeometryShaders = false

  Specifies whether geometry shaders are supported.
- bool hasTessellationShaders = false

  Specifies whether tessellation shaders are supported.
- bool hasComputeShaders = false

  Speciifes whether compute shaders are supported.
- bool hasInstancing = false

  Specifies whether hardware instancing is supported.
- bool hasOffsetInstancing = false

  Specifies whether hardware instancing with instance offsets is supported.
- bool hasViewportArrays = false

  Specifies whether multiple viewports, depth-ranges, and scissors are supported at once.
- bool hasConservativeRasterization = false

  Specifies whether conservative rasterization is supported.
- unsigned int maxNumTextureArrayLayers = 0

  Specifies maximum number of texture array layers (for 1D-, 2D-, and cube textures).
- unsigned int maxNumRenderTargetAttachments = 0

  Specifies maximum number of attachment points for each render target.
- unsigned int maxConstantBufferSize = 0

  Specifies maximum size (in bytes) of each constant buffer.

- int maxPatchVertices = 0

  Specifies maximum number of patch control points.
- int max1DTextureSize = 0

  Specifies maximum size of each 1D texture.
- int max2DTextureSize = 0

  Specifies maximum size of each 2D texture (for width and height).
- int max3DTextureSize = 0

  Specifies maximum size of each 3D texture (for width, height, and depth).
- int maxCubeTextureSize = 0

  Specifies maximum size of each cube texture (for width and height).
- int maxAnisotropy = 0

  Specifies maximum anisotropy texture filter.
- Gs::Vector3ui maxNumComputeShaderWorkGroups

  Specifies maximum number of work groups in a compute shader.
- Gs::Vector3ui maxComputeShaderWorkGroupSize

  Specifies maximum work group size in a compute shader.

4.32.1 Detailed Description

Rendering capabilities structure.

4.32.2 Member Data Documentation

4.32.2.1 ScreenOrigin LLGL::RenderingCaps::screenOrigin = ScreenOrigin::UpperLeft

Screen coordinate system origin.

Remarks

This determines the coordinate space of viewports, scissors, and framebuffers.

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

- RenderSystemFlags.h

4.33 LLGL::RenderingProfiler Class Reference

Rendering profiler model class.

```
#include <RenderingProfiler.h>
```

Classes

- class Counter

Public Member Functions

- void ResetCounters ()

  Resets all counters.
- void **RecordDrawCall** (const PrimitiveTopology topology, Counter::ValueType numVertices)
- void **RecordDrawCall** (const PrimitiveTopology topology, Counter::ValueType numVertices, Counter::ValueType numInstances)

Public Attributes

- Counter **writeVertexBuffer**
- Counter **writeIndexBuffer**
- Counter **writeConstantBuffer**
- Counter **writeStorageBuffer**
- Counter **mapConstantBuffer**
- Counter **mapStorageBuffer**
- Counter **setVertexBuffer**
- Counter **setIndexBuffer**
- Counter **setConstantBuffer**
- Counter **setStorageBuffer**
- Counter **setGraphicsPipeline**
- Counter **setComputePipeline**
- Counter **setTexture**
- Counter **setSampler**
- Counter **setRenderTarget**
- Counter **drawCalls**
- Counter **dispatchComputeCalls**
- Counter **renderedPoints**
- Counter **renderedLines**
- Counter **renderedTriangles**
- Counter **renderedPatches**

4.33.1 Detailed Description

Rendering profiler model class.

Remarks

This can be used to profile the renderer draw calls and buffer updates.

4.33.2 Member Function Documentation

4.33.2.1 void LLGL::RenderingProfiler::ResetCounters ( )

Resets all counters.

See also

Counter::Reset

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

- RenderingProfiler.h

4.34 LLGL::RenderSystem Class Reference

Render system interface.

`#include <RenderSystem.h>`

Classes

- struct Configuration

  Render system configuration structure.

Public Member Functions

- **RenderSystem** (const RenderSystem &)=delete
- RenderSystem & **operator=** (const RenderSystem &)=delete
- const std::string & GetName () const

  Returns the name of this render system.
- virtual std::map< RendererInfo, std::string > QueryRendererInfo () const =0

  Returns all available renderer information.
- virtual RenderingCaps QueryRenderingCaps () const =0

  Returns the rendering capabilities.
- virtual ShadingLanguage QueryShadingLanguage () const =0

  Returns the highest version of the supported shading language.
- virtual RenderContext * CreateRenderContext (const RenderContextDescriptor &desc, const std::shared_ptr< Window > &window=nullptr)=0

  Creates a new render context and returns the raw pointer.
- virtual void Release (RenderContext &renderContext)=0

  Releases the specified render context. This will all release all resources, that are associated with this render context.
- bool MakeCurrent (RenderContext *renderContext)

  Makes the specified render context to the current one.
- RenderContext * GetCurrentContext () const

  Returns the current render context. This may also be null.
- virtual VertexBuffer * CreateVertexBuffer ()=0

  Creates a new, empty, and unspecified vertex buffer.
- virtual IndexBuffer * CreateIndexBuffer ()=0

  Creates a new, empty, and unspecified index buffer.
- virtual ConstantBuffer * CreateConstantBuffer ()=0

  Creates a new, empty, and unspecified constant buffer (also called "Uniform Buffer Object").
- virtual StorageBuffer * CreateStorageBuffer ()=0

  Creates a new, empty, and unspecified storage buffer (also called "Read/Write Buffer").
- virtual void **Release** (VertexBuffer &vertexBuffer)=0
- virtual void **Release** (IndexBuffer &indexBuffer)=0
- virtual void **Release** (ConstantBuffer &constantBuffer)=0
- virtual void **Release** (StorageBuffer &storageBuffer)=0
- virtual void SetupVertexBuffer (VertexBuffer &vertexBuffer, const void *data, std::size_t dataSize, const BufferUsage usage, const VertexFormat &vertexFormat)=0

  Initializes the specified vertex buffer.
- virtual void SetupIndexBuffer (IndexBuffer &indexBuffer, const void *data, std::size_t dataSize, const BufferUsage usage, const IndexFormat &indexFormat)=0

  Initializes the specified index buffer.

- virtual void SetupConstantBuffer (ConstantBuffer &constantBuffer, const void *data, std::size_t dataSize, const BufferUsage usage)=0

  Initializes the specified constant buffer.
- virtual void SetupStorageBuffer (StorageBuffer &storageBuffer, const void *data, std::size_t dataSize, const BufferUsage usage)=0

  Initializes the specified storage buffer.
- virtual void WriteVertexBuffer (VertexBuffer &vertexBuffer, const void *data, std::size_t dataSize, std::size_t offset)=0

  Updates the data of the specified vertex buffer.
- virtual void WriteIndexBuffer (IndexBuffer &indexBuffer, const void *data, std::size_t dataSize, std::size_t offset)=0
- virtual void WriteConstantBuffer (ConstantBuffer &constantBuffer, const void *data, std::size_t dataSize, std::size_t offset)=0
- virtual void WriteStorageBuffer (StorageBuffer &storageBuffer, const void *data, std::size_t dataSize, std::size_t offset)=0
- virtual Texture * CreateTexture ()=0

  Creates a new, empty, and unspecified texture.
- virtual void **Release** (Texture &texture)=0
- virtual TextureDescriptor QueryTextureDescriptor (const Texture &texture)=0

  Queries a descriptor of the specified texture.
- virtual void SetupTexture1D (Texture &texture, const TextureFormat format, int size, const ImageDataDescriptor *imageDesc=nullptr)=0

  Initializes the specified texture as a 1-dimensional texture.
- virtual void SetupTexture2D (Texture &texture, const TextureFormat format, const Gs::Vector2i &size, const ImageDataDescriptor *imageDesc=nullptr)=0

  Initializes the specified texture as a 2-dimensional texture.
- virtual void SetupTexture3D (Texture &texture, const TextureFormat format, const Gs::Vector3i &size, const ImageDataDescriptor *imageDesc=nullptr)=0

  Initializes the specified texture as a 3-dimensional texture.
- virtual void SetupTextureCube (Texture &texture, const TextureFormat format, const Gs::Vector2i &size, const ImageDataDescriptor *imageDesc=nullptr)=0

  Initializes the specified texture as a cube texture with six faces.
- virtual void SetupTexture1DArray (Texture &texture, const TextureFormat format, int size, unsigned int layers, const ImageDataDescriptor *imageDesc=nullptr)=0

  Initializes the specified texture as a 1-dimensional array texture.
- virtual void SetupTexture2DArray (Texture &texture, const TextureFormat format, const Gs::Vector2i &size, unsigned int layers, const ImageDataDescriptor *imageDesc=nullptr)=0

  Initializes the specified texture as a 2-dimensional array texture.
- virtual void SetupTextureCubeArray (Texture &texture, const TextureFormat format, const Gs::Vector2i &size, unsigned int layers, const ImageDataDescriptor *imageDesc=nullptr)=0

  Initializes the specified texture as a cube array texture with six faces for each layer.
- virtual void WriteTexture1D (Texture &texture, int mipLevel, int position, int size, const ImageDataDescriptor &imageDesc)=0

  Updates the data of the specified texture.
- virtual void WriteTexture2D (Texture &texture, int mipLevel, const Gs::Vector2i &position, const Gs::Vector2i &size, const ImageDataDescriptor &imageDesc)=0
- virtual void WriteTexture3D (Texture &texture, int mipLevel, const Gs::Vector3i &position, const Gs::Vector3i &size, const ImageDataDescriptor &imageDesc)=0
- virtual void WriteTextureCube (Texture &texture, int mipLevel, const Gs::Vector2i &position, const AxisDirection cubeFace, const Gs::Vector2i &size, const ImageDataDescriptor &imageDesc)=0
- virtual void WriteTexture1DArray (Texture &texture, int mipLevel, int position, unsigned int layerOffset, int size, unsigned int layers, const ImageDataDescriptor &imageDesc)=0
- virtual void WriteTexture2DArray (Texture &texture, int mipLevel, const Gs::Vector2i &position, unsigned int layerOffset, const Gs::Vector2i &size, unsigned int layers, const ImageDataDescriptor &imageDesc)=0

- virtual void WriteTextureCubeArray (Texture &texture, int mipLevel, const Gs::Vector2i &position, unsigned int layerOffset, const AxisDirection cubeFaceOffset, const Gs::Vector2i &size, unsigned int cubeFaces, const ImageDataDescriptor &imageDesc)=0
- virtual void ReadTexture (const Texture &texture, int mipLevel, ColorFormat dataFormat, DataType dataType, void *data)=0

  Reads the image data from the specified texture.
- virtual Sampler * CreateSampler (const SamplerDescriptor &desc)=0

  Creates a new Sampler object.
- virtual void Release (Sampler &sampler)=0

  Releases the specified Sampler object. After this call, the specified object must no longer be used.
- virtual RenderTarget * CreateRenderTarget (unsigned int multiSamples=0)=0

  Creates a new RenderTarget object with the specified number of samples.
- virtual void Release (RenderTarget &renderTarget)=0

  Releases the specified RenderTarget object. After this call, the specified object must no longer be used.
- virtual Shader * CreateShader (const ShaderType type)=0

  Creates a new and empty shader.
- virtual ShaderProgram * CreateShaderProgram ()=0

  Creates a new and empty shader program.
- virtual void **Release** (Shader &shader)=0
- virtual void **Release** (ShaderProgram &shaderProgram)=0
- virtual GraphicsPipeline * CreateGraphicsPipeline (const GraphicsPipelineDescriptor &desc)=0

  Creates a new and initialized graphics pipeline state object.
- virtual ComputePipeline * CreateComputePipeline (const ComputePipelineDescriptor &desc)=0

  Creates a new and initialized compute pipeline state object.
- virtual void **Release** (GraphicsPipeline &graphicsPipeline)=0
- virtual void **Release** (ComputePipeline &computePipeline)=0
- virtual Query * CreateQuery (const QueryType type)=0

  Creates a new query of the specified type.
- virtual void **Release** (Query &query)=0

Static Public Member Functions

- static std::vector< std::string > FindModules ()

  Returns the list of all available render system modules for the current platform (e.g. on Windows this might be { "OpenGL", "Direct3D12" }, but on MacOS it might be only { "OpenGL" }).
- static std::shared_ptr< RenderSystem > Load (const std::string &moduleName, RenderingProfiler *profiler=nullptr)

  Loads a new render system from the specified module.

Public Attributes

- Configuration config

  Render system basic configuration.

Protected Member Functions

- virtual bool OnMakeCurrent (RenderContext *renderContext)

  Callback when a new render context is about to be made the current one.
- std::vector< ColorRGBAub > GetDefaultTextureImageRGBAub (int numPixels) const

  Creates an RGBA unsigned-byte image buffer for the specified number of pixels.

4.34.1 Detailed Description

Render system interface.

Remarks

This is the main interface for the entire renderer. It manages the ownership of all graphics objects and is used to create, modify, and delete all those objects. The main functions for most graphics objects are "Create...", "Setup...", "Write...", and "Release":

```
// Create an empty and unspecified vertex buffer
auto vertexBuffer = renderSystem->CreateVertexBuffer();

// Initialize object
renderSystem->SetupVertexBuffer(vertexBuffer, initialData, ...);

// Modify data
renderSystem->WriteVertexBuffer(vertexBuffer, modificationData, ...);

// Release object
renderSystem->Release(*vertexBuffer);
```

4.34.2 Member Function Documentation

4.34.2.1 virtual ComputePipeline* LLGL::RenderSystem::CreateComputePipeline (const ComputePipelineDescriptor & *desc*) [pure virtual]

Creates a new and initialized compute pipeline state object.

Parameters

in	*desc*	Specifies the compute pipeline descriptor. This will describe the shader states. The "shaderProgram" member of the descriptor must never be null!

See also

ComputePipelineDescriptor

4.34.2.2 virtual ConstantBuffer* LLGL::RenderSystem::CreateConstantBuffer ( ) [pure virtual]

Creates a new, empty, and unspecified constant buffer (also called "Uniform Buffer Object").

See also

SetupConstantBuffer

4.34.2.3 virtual GraphicsPipeline* LLGL::RenderSystem::CreateGraphicsPipeline (const GraphicsPipelineDescriptor & *desc*) [pure virtual]

Creates a new and initialized graphics pipeline state object.

Parameters

in	*desc*	Specifies the graphics pipeline descriptor. This will describe the entire pipeline state, i.e. the blending-, rasterizer-, depth-, stencil- and shader states. The "shaderProgram" member of the descriptor must never be null!

See also

GraphicsPipelineDescriptor

4.34.2.4 virtual IndexBuffer* LLGL::RenderSystem::CreateIndexBuffer ( ) [pure virtual]

Creates a new, empty, and unspecified index buffer.

See also

SetupIndexBuffer

4.34.2.5 virtual RenderContext* LLGL::RenderSystem::CreateRenderContext (const RenderContextDescriptor & *desc,* const std::shared_ptr< Window > & *window* = nullptr) [pure virtual]

Creates a new render context and returns the raw pointer.

Remarks

The render system takes the ownership of this object. All render contexts are deleted in the destructor of this render system.

4.34.2.6 virtual RenderTarget* LLGL::RenderSystem::CreateRenderTarget (unsigned int *multiSamples* = 0) [pure virtual]

Creates a new RenderTarget object with the specified number of samples.

Exceptions

std::runtime_error	If the renderer does not support RenderTarget objects (e.g. if OpenGL 2.1 or lower is used).

4.34.2.7 virtual Sampler* LLGL::RenderSystem::CreateSampler (const SamplerDescriptor & *desc*) [pure virtual]

Creates a new Sampler object.

Exceptions

std::runtime_error	If the renderer does not support Sampler objects (e.g. if OpenGL 3.1 or lower is used).

See also

RenderContext::QueryRenderingCaps

4.34.2.8 virtual Shader* LLGL::RenderSystem::CreateShader (const ShaderType *type*) [pure virtual]

Creates a new and empty shader.

Parameters

in	*type*	Specifies the type of the shader, i.e. if it is either a vertex or fragment shader or the like.

See also

Shader

4.34.2.9 virtual ShaderProgram* LLGL::RenderSystem::CreateShaderProgram ( ) [pure virtual]

Creates a new and empty shader program.

Remarks

At least one shader must be attached to a shader program to be used for a graphics or compute pipeline.

See also

ShaderProgram

4.34.2.10 virtual StorageBuffer* LLGL::RenderSystem::CreateStorageBuffer ( ) [pure virtual]

Creates a new, empty, and unspecified storage buffer (also called "Read/Write Buffer").

See also

SetupStorageBuffer

4.34.2.11 virtual Texture* LLGL::RenderSystem::CreateTexture ( ) [pure virtual]

Creates a new, empty, and unspecified texture.

Remarks

The type and dimension size of the this texture will be determined by any of the "SetupTexture..." functions.

4.34.2.12 virtual VertexBuffer* LLGL::RenderSystem::CreateVertexBuffer ( ) [pure virtual]

Creates a new, empty, and unspecified vertex buffer.

See also

SetupVertexBuffer

4.34.2.13 static std::shared_ptr<RenderSystem> LLGL::RenderSystem::Load (const std::string & *moduleName*, RenderingProfiler * *profiler* = nullptr) [static]

Loads a new render system from the specified module.

Parameters

in	*moduleName*	Specifies the name from which the new render system is to be loaded. This denotes a dynamic library (∗.dll-files on Windows, ∗.so-files on Unix systems). If compiled in debug mode, the postfix "D" is appended to the module name. Moreover, the platform dependent file extension is always added automatically as well as the prefix "LLGL_", i.e. a module name "OpenGL" will be translated to "LLGL_OpenGLD.dll", if compiled on Windows in Debug mode.
in	*profiler*	Optional pointer to a rendering profiler. If this is used, the counters of the profiler must be reset manually.

Remarks

Usually the return type is a std::unique_ptr, but LLGL needs to keep track of the existance of this render system because only a single instance can be loaded at a time. So a std::weak_ptr is stored internally to check if it has been expired (see http://en.cppreference.com/w/cpp/memory/weak_ptr/expired), and this type can only refer to a std::shared_ptr.

Exceptions

std::runtime_error	If loading the render system from the specified module failed.
std::runtime_error	If there is already a loaded instance of a render system (make sure there are no more shared pointer references to the previous render system!)

4.34.2.14 bool LLGL::RenderSystem::MakeCurrent (RenderContext ∗ *renderContext*)

Makes the specified render context to the current one.

Parameters

in	*renderContext*	Specifies the new current render context. If this is null, no render context is active.

Returns

True on success, otherwise false.

Remarks

Never draw anything, while no render context is active!

4.34.2.15 virtual bool LLGL::RenderSystem::OnMakeCurrent (RenderContext ∗ *renderContext*) [protected], [virtual]

Callback when a new render context is about to be made the current one.

Remarks

At this point, "GetCurrentContext" returns still the previous render context.

4.34.2.16 virtual TextureDescriptor LLGL::RenderSystem::QueryTextureDescriptor (const Texture & *texture*) [pure virtual]

Queries a descriptor of the specified texture.

Remarks

This can be used to query the type and dimension size of the texture.

See also

TextureDescriptor

4.34.2.17 virtual void LLGL::RenderSystem::ReadTexture (const Texture & *texture*, int *mipLevel*, ColorFormat *dataFormat*, DataType *dataType*, void * *data*) [pure virtual]

Reads the image data from the specified texture.

Parameters

in	*texture*	Specifies the texture object to read from.
in	*mipLevel*	Specifies the MIP-level from which to read the image data.
in	*dataFormat*	Specifies the output data format.
in	*dataType*	Specifies the output data type.
out	*data*	Specifies the output image data. This must be a pointer to a memory block, which is large enough to fit all the image data.

Remarks

Depending on the data format, data type, and texture size, the output image container must be allocated with enough memory size. The "QueryTextureDescriptor" function can be used to determine the texture dimensions.

```
std::vector<LLGL::ColorRGBAub> image(textureWidth*textureHeight);
renderSystem->ReadTexture(texture, 0, LLGL::ColorFormat::RGBA, LLGL::DataType::UByte, image.data());
```

See also

QueryTextureDescriptor

4.34.2.18 virtual void LLGL::RenderSystem::SetupConstantBuffer (ConstantBuffer & *constantBuffer*, const void * *data*, std::size_t *dataSize*, const BufferUsage *usage*) [pure virtual]

Initializes the specified constant buffer.

Parameters

in	*constantBuffer*	Specifies the constant buffer which is to be initialized.
in	*data*	Raw pointer to the data with which the constant buffer is to be initialized. This may also be null, to only initialize the size of the buffer. In this case, the buffer must be initialized with the "WriteConstantBuffer" function before it is used for drawing operations.
in	*dataSize*	Specifies the size (in bytes) of the buffer.
in	*usage*	Specifies the buffer usage.

See also

WriteConstantBuffer

4.34.2.19 virtual void LLGL::RenderSystem::SetupIndexBuffer (IndexBuffer & *indexBuffer,* const void * *data,* std::size_t *dataSize,* const BufferUsage *usage,* const IndexFormat & *indexFormat*) [pure virtual]

Initializes the specified index buffer.

Parameters

in	*indexBuffer*	Specifies the index buffer which is to be initialized.
in	*data*	Raw pointer to the data with which the index buffer is to be initialized. This may also be null, to only initialize the size of the buffer. In this case, the buffer must be initialized with the "WriteIndexBuffer" function before it is used for drawing operations.
in	*dataSize*	Specifies the size (in bytes) of the buffer.
in	*usage*	Specifies the buffer usage, which is typically "BufferUsage::Static" for an index buffer, since it is rarely changed.
in	*indexFormat*	Specifies the index format layout, which is basically only the data type of each index. The only valid format types for an index buffer are: DataType::UByte, DataType::UShort, DataType::UInt.

See also

IndexFormat
WriteIndexBuffer

4.34.2.20 virtual void LLGL::RenderSystem::SetupStorageBuffer (StorageBuffer & *storageBuffer,* const void * *data,* std::size_t *dataSize,* const BufferUsage *usage*) [pure virtual]

Initializes the specified storage buffer.

Parameters

in	*storageBuffer*	Specifies the storage buffer which is to be initialized.
in	*data*	Raw pointer to the data with which the storage buffer is to be initialized. This may also be null, to only initialize the size of the buffer. In this case, the buffer must be initialized with the "WriteStorageBuffer" function before it is used for drawing operations.
in	*dataSize*	Specifies the size (in bytes) of the buffer.
in	*usage*	Specifies the buffer usage.

See also

WriteStorageBuffer

4.34.2.21 virtual void LLGL::RenderSystem::SetupTexture1D (Texture & *texture,* const TextureFormat *format,* int *size,* const ImageDataDescriptor * *imageDesc* = nullptr) [pure virtual]

Initializes the specified texture as a 1-dimensional texture.

Parameters

in	*texture*	Specifies the texture which is to be initialized.
in	*format*	Specifies the hardware texture format.
in	*size*	Specifies the size of the texture (in texels, 'texture elements').
in	*imageDesc*	Optional pointer to the image data descriptor. If this is null, the texture will be initialized with the currently configured default image color (see "Configuration::defaultTextureImageColor"). If this is non-null, is is used to initialize the texture data.

See also

WriteTexture1D
Configuration::defaultTextureImageColor

4.34.2.22 virtual void LLGL::RenderSystem::SetupTexture1DArray (Texture & *texture*, const TextureFormat *format*, int *size*, unsigned int *layers*, const ImageDataDescriptor * *imageDesc* = nullptr) [pure virtual]

Initializes the specified texture as a 1-dimensional array texture.

Parameters

in	*layers*	Specifies the number of array layers.

See also

SetupTexture1D
WriteTexture1DArray

4.34.2.23 virtual void LLGL::RenderSystem::SetupTexture2D (Texture & *texture*, const TextureFormat *format*, const Gs::Vector2i & *size*, const ImageDataDescriptor * *imageDesc* = nullptr) [pure virtual]

Initializes the specified texture as a 2-dimensional texture.

See also

SetupTexture1D
WriteTexture2D

4.34.2.24 virtual void LLGL::RenderSystem::SetupTexture2DArray (Texture & *texture*, const TextureFormat *format*, const Gs::Vector2i & *size*, unsigned int *layers*, const ImageDataDescriptor * *imageDesc* = nullptr) [pure virtual]

Initializes the specified texture as a 2-dimensional array texture.

See also

SetupTexture1DArray
WriteTexture2DArray

4.34.2.25 virtual void LLGL::RenderSystem::SetupTexture3D (Texture & *texture*, const TextureFormat *format*, const Gs::Vector3i & *size*, const ImageDataDescriptor * *imageDesc* = nullptr) [pure virtual]

Initializes the specified texture as a 3-dimensional texture.

See also

SetupTexture1D
WriteTexture3D

4.34.2.26 virtual void LLGL::RenderSystem::SetupTextureCube (Texture & *texture*, const TextureFormat *format*, const Gs::Vector2i & *size*, const ImageDataDescriptor * *imageDesc* = nullptr) [pure virtual]

Initializes the specified texture as a cube texture with six faces.

Remarks

If the image data descriptor is used, the image data must be large anough to store the image data of all six cube faces (i.e. width * height * 6 texels). The order of the cube faces is: AxisDirection::XPos, AxisDirection::XNeg, AxisDirection::YPos, AxisDirection::YNeg, AxisDirection::ZPos, AxisDirection::ZNeg.

See also

SetupTexture1D
WriteTextureCube
AxisDirection

4.34.2.27 virtual void LLGL::RenderSystem::SetupTextureCubeArray (Texture & *texture*, const TextureFormat *format*, const Gs::Vector2i & *size*, unsigned int *layers*, const ImageDataDescriptor * *imageDesc* = nullptr) [pure virtual]

Initializes the specified texture as a cube array texture with six faces for each layer.

See also

SetupTexture1DArray
WriteTextureCubeArray

4.34.2.28 virtual void LLGL::RenderSystem::SetupVertexBuffer (VertexBuffer & *vertexBuffer*, const void * *data*, std::size_t *dataSize*, const BufferUsage *usage*, const VertexFormat & *vertexFormat*) [pure virtual]

Initializes the specified vertex buffer.

Parameters

in	*vertexBuffer*	Specifies the vertex buffer which is to be initialized.
in	*data*	Raw pointer to the data with which the vertex buffer is to be initialized. This may also be null, to only initialize the size of the buffer. In this case, the buffer must be initialized with the "WriteVertexBuffer" function before it is used for drawing operations.
in	*dataSize*	Specifies the size (in bytes) of the buffer.
in	*usage*	Specifies the buffer usage, which is typically "BufferUsage::Static" for a vertex buffer, since it is rarely changed.
in	*vertexFormat*	Specifies the vertex format layout, which is required to tell the renderer how the vertex

See also

VertexFormat
WriteVertexBuffer
ShaderProgram

4.34.2.29 virtual void LLGL::RenderSystem::WriteConstantBuffer (ConstantBuffer & *constantBuffer,* const void * *data,* std::size_t *dataSize,* std::size_t *offset*) [pure virtual]

See also

WriteVertexBuffer

4.34.2.30 virtual void LLGL::RenderSystem::WriteIndexBuffer (IndexBuffer & *indexBuffer,* const void * *data,* std::size_t *dataSize,* std::size_t *offset*) [pure virtual]

See also

WriteVertexBuffer

4.34.2.31 virtual void LLGL::RenderSystem::WriteStorageBuffer (StorageBuffer & *storageBuffer,* const void * *data,* std::size_t *dataSize,* std::size_t *offset*) [pure virtual]

See also

WriteVertexBuffer

4.34.2.32 virtual void LLGL::RenderSystem::WriteTexture1D (Texture & *texture,* int *mipLevel,* int *position,* int *size,* const ImageDataDescriptor & *imageDesc*) [pure virtual]

Updates the data of the specified texture.

Parameters

in	*texture*	Specifies the texture whose data is to be updated.
in	*mipLevel*	Specifies the zero-based MIP ("Multum in Parvo") level which is to be updated.
in	*position*	Specifies the position offset of the portion which is to be updated.
in	*size*	Specifies the size of the portion which is to be updated.
in	*imageDesc*	Specifies the image data descriptor. Its "data" member must not be null!

Remarks

This texture must be initialized as a 1-dimensional texture.

4.34.2.33 virtual void LLGL::RenderSystem::WriteTexture1DArray (Texture & *texture,* int *mipLevel,* int *position,* unsigned int *layerOffset,* int *size,* unsigned int *layers,* const ImageDataDescriptor & *imageDesc*) [pure virtual]

Parameters

in	*layerOffset*	Specifies the zero-based layer offset of the portion which is to be updated.
in	*layers*	Specifies the number of layers to update.

See also

WriteTexture1D

Remarks

This texture must be initialized as a 1-dimensional array texture.

4.34.2.34 virtual void LLGL::RenderSystem::WriteTexture2D (Texture & *texture,* int *mipLevel,* const Gs::Vector2i & *position,* const Gs::Vector2i & *size,* const ImageDataDescriptor & *imageDesc*) [pure virtual]

See also

WriteTexture1D

Remarks

This texture must be initialized as a 2-dimensional texture.

4.34.2.35 virtual void LLGL::RenderSystem::WriteTexture2DArray (Texture & *texture,* int *mipLevel,* const Gs::Vector2i & *position,* unsigned int *layerOffset,* const Gs::Vector2i & *size,* unsigned int *layers,* const ImageDataDescriptor & *imageDesc*) [pure virtual]

See also

WriteTexture1DArray

Remarks

This texture must be initialized as a 2-dimensional array texture.

4.34.2.36 virtual void LLGL::RenderSystem::WriteTexture3D (Texture & *texture,* int *mipLevel,* const Gs::Vector3i & *position,* const Gs::Vector3i & *size,* const ImageDataDescriptor & *imageDesc*) [pure virtual]

See also

WriteTexture1D

Remarks

This texture must be initialized as a 3-dimensional texture.

4.34.2.37 virtual void LLGL::RenderSystem::WriteTextureCube (Texture & *texture,* int *mipLevel,* const Gs::Vector2i & *position,* const AxisDirection *cubeFace,* const Gs::Vector2i & *size,* const ImageDataDescriptor & *imageDesc*) [pure virtual]

Parameters

in	*cubeFace*	Specifies the cube face which is to be updated.

See also

WriteTexture1D

Remarks

This texture must be initialized as a cube texture.

4.34.2.38 virtual void LLGL::RenderSystem::WriteTextureCubeArray (Texture & *texture,* int *mipLevel,* const Gs::Vector2i & *position,* unsigned int *layerOffset,* const AxisDirection *cubeFaceOffset,* const Gs::Vector2i & *size,* unsigned int *cubeFaces,* const ImageDataDescriptor & *imageDesc*) [pure virtual]

See also

WriteTexture1DArray

Parameters

in	*cubeFaceOffset*	Specifies the cube face offset of the portion which is to be updated.
in	*cubeFaces*	Specifies the number of cube faces to update. This can be out of bounds of the six cube faces, i.e. it can exceed several layers.

Remarks

This texture must be initialized as a cube array texture.

4.34.2.39 virtual void LLGL::RenderSystem::WriteVertexBuffer (VertexBuffer & *vertexBuffer,* const void * *data,* std::size_t *dataSize,* std::size_t *offset*) [pure virtual]

Updates the data of the specified vertex buffer.

Parameters

in	*vertexBuffer*	Specifies the vertex buffer whose data is to be updated.
in	*data*	Raw pointer to the data with which the vertex buffer is to be updated. This must not be null!
in	*dataSize*	Specifies the size (in bytes) of the data block which is to be updated. This must be less then or equal to the size of the vertex buffer.
in	*offset*	Specifies the offset (in bytes) at which the vertex buffer is to be updated. This offset plus the data block size (i.e. 'offset + dataSize') must be less than or equal to the size of the vertex buffer.

4.34.3 Member Data Documentation

4.34.3.1 Configuration LLGL::RenderSystem::config

Render system basic configuration.

Remarks

This can be used to change the behavior of default initializion of textures for instance.

See also

Configuration

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

- RenderSystem.h

4.35 LLGL::RenderTarget Class Reference

Render target interface.

`#include <RenderTarget.h>`

Public Member Functions

- virtual void AttachDepthBuffer (const Gs::Vector2i &size)=0

  Attaches an internal depth buffer to this render target.
- virtual void AttachStencilBuffer (const Gs::Vector2i &size)=0

  Attaches an internal stencil buffer to this render target.
- virtual void AttachDepthStencilBuffer (const Gs::Vector2i &size)=0

  Attaches an internal depth-stencil buffer to this render target.
- virtual void **AttachTexture1D** (Texture &texture, int mipLevel=0)=0
- virtual void **AttachTexture2D** (Texture &texture, int mipLevel=0)=0
- virtual void **AttachTexture3D** (Texture &texture, int layer, int mipLevel=0)=0
- virtual void **AttachTextureCube** (Texture &texture, const AxisDirection cubeFace, int mipLevel=0)=0
- virtual void **AttachTexture1DArray** (Texture &texture, int layer, int mipLevel=0)=0
- virtual void **AttachTexture2DArray** (Texture &texture, int layer, int mipLevel=0)=0
- virtual void **AttachTextureCubeArray** (Texture &texture, int layer, const AxisDirection cubeFace, int mip↩ Level=0)=0
- virtual void DetachTextures ()=0

  Detaches all textures from this render target.
- const Gs::Vector2i & GetResolution () const

  Returns the frame buffer resolution.

Protected Member Functions

- void **ApplyResolution** (const Gs::Vector2i &resolution)
- void **ResetResolution** ()

4.35.1 Detailed Description

Render target interface.

4.35.2 Member Function Documentation

4.35.2.1 virtual void LLGL::RenderTarget::AttachDepthBuffer (const Gs::Vector2i & *size*) [pure virtual]

Attaches an internal depth buffer to this render target.

Parameters

in	*size*	Specifies the size of the depth buffer. This must be the same as for all other attachemnts.

Remarks

Only a single depth buffer, stencil buffer, or depth-stencil buffer can be attached.

See also

AttachDepthStencilBuffer

4.35.2.2 virtual void LLGL::RenderTarget::AttachDepthStencilBuffer (const Gs::Vector2i & *size*) [pure virtual]

Attaches an internal depth-stencil buffer to this render target.

Remarks

Only a single depth buffer, stencil buffer, or depth-stencil buffer can be attached.

See also

AttachDepthBuffer

4.35.2.3 virtual void LLGL::RenderTarget::AttachStencilBuffer (const Gs::Vector2i & *size*) [pure virtual]

Attaches an internal stencil buffer to this render target.

Remarks

Only a single depth buffer, stencil buffer, or depth-stencil buffer can be attached.

See also

AttachDepthBuffer

4.35.2.4 const Gs::Vector2i& LLGL::RenderTarget::GetResolution ( ) const [inline]

Returns the frame buffer resolution.

Remarks

This will be determined by the first texture attachment. Every further attachment must have the same size.

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

- RenderTarget.h

4.36 LLGL::Sampler Class Reference

Sampler interface.

`#include <Sampler.h>`

4.36.1 Detailed Description

Sampler interface.

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

- Sampler.h

4.37 LLGL::SamplerDescriptor Struct Reference

Texture sampler descriptor structure.

`#include <SamplerFlags.h>`

Public Attributes

- TextureWrap textureWrapU = TextureWrap::Repeat

  Texture coordinate wrap mode in U direction. By default TextureWrap::Repeat.
- TextureWrap textureWrapV = TextureWrap::Repeat

  Texture coordinate wrap mode in V direction. By default TextureWrap::Repeat.
- TextureWrap textureWrapW = TextureWrap::Repeat

  Texture coordinate wrap mode in W direction. By default TextureWrap::Repeat.
- TextureFilter minFilter = TextureFilter::Linear

  Minification filter. By default TextureFilter::Linear.
- TextureFilter magFilter = TextureFilter::Linear

  Magnification filter. By default TextureFilter::Linear.
- TextureFilter mipMapFilter = TextureFilter::Linear

  MIP-mapping filter. By default TextureFilter::Linear.
- bool mipMapping = true

  Specifies whether MIP-maps are used or not. By default true.
- float mipMapLODBias = 0.0f

  MIP-mapping level-of-detail (LOD) bias (or rather offset). By default 0.
- float minLOD = 0.0f

  Lower end of the MIP-map range. By default 0.
- float maxLOD = 1000.0f

  Upper end of the MIP-map range. Must be greater than or equal to "minLOD". By default 1000.
- unsigned int maxAnisotropy = 1

  Maximal anisotropy in the range [1, 16].
- bool depthCompare = false

  Specifies whether the compare operation for depth textures is to be used or not. By default false.
- CompareOp compareOp = CompareOp::Less

  Compare operation for depth textures. By default CompareOp::Less.
- ColorRGBAf borderColor = { 0.0f, 0.0f, 0.0f, 0.0f }

  Border color. By default black (0, 0, 0, 0).

4.37.1 Detailed Description

Texture sampler descriptor structure.

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

- SamplerFlags.h

4.38 LLGL::Scissor Struct Reference

Scissor dimensions.

`#include <RenderContextFlags.h>`

Public Member Functions

- **Scissor** (const Scissor &)=default
- **Scissor** (int x, int y, int width, int height)

Public Attributes

- int **x** = 0
- int **y** = 0
- int **width** = 0
- int **height** = 0

4.38.1 Detailed Description

Scissor dimensions.

Remarks

A scissor is in screen coordinates where the origin is in the left-top corner.

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

- RenderContextFlags.h

4.39 LLGL::Shader Class Reference

Shader interface.

`#include <Shader.h>`

Public Member Functions

- virtual bool Compile (const ShaderSource &shaderSource)=0

  Compiles the specified shader source.
- virtual std::string Disassemble (int flags=0)

  Disassembles the previously compiled shader byte code.
- virtual std::string QueryInfoLog ()=0

  Returns the information log after the shader compilation.
- ShaderType GetType () const

  Returns the type of this shader.

Protected Member Functions

- **Shader** (const ShaderType type)

4.39.1 Detailed Description

Shader interface.

4.39.2 Member Function Documentation

4.39.2.1 virtual bool LLGL::Shader::Compile (const ShaderSource & *shaderSource*) [pure virtual]

Compiles the specified shader source.

Parameters

in	*shaderSource*	Specifies the shader source code.

Returns

True on success, otherwise "QueryInfoLog" can be used to query the reason for failure.

See also

QueryInfoLog

4.39.2.2 virtual std::string LLGL::Shader::Disassemble (int *flags* = 0) [virtual]

Disassembles the previously compiled shader byte code.

Parameters

in	*flags*	Specifies optional disassemble flags. This can be a bitwise OR combination of the 'ShaderDisassembleFlags' enumeration entries. By default 0.

Returns

Disassembled assembler code or an empty string if disassembling was not possible.

Note

Only supported with: Direct3D 11, Direct3D 12 (for HLSL).

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

- Shader.h

4.40 LLGL::ShaderCompileFlags Struct Reference

Shader compilation flags enumeration.

`#include <Shader.h>`

Public Types

- enum {
  Debug = (1 << 0), O1 = (1 << 1), O2 = (1 << 2), O3 = (1 << 3),
  WarnError = (1 << 4) }

4.40.1 Detailed Description

Shader compilation flags enumeration.

4.40.2 Member Enumeration Documentation

4.40.2.1 anonymous enum

Enumerator

Debug Insert debug information.

O1 Optimization level 1.

O2 Optimization level 2.

O3 Optimization level 3.

WarnError Warnings are treated as errors.

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

- Shader.h

4.41 LLGL::ShaderDisassembleFlags Struct Reference

Shader disassemble flags enumeration.

`#include <Shader.h>`

Public Types

- enum { InstructionOnly = (1 << 0) }

4.41.1 Detailed Description

Shader disassemble flags enumeration.

4.41.2 Member Enumeration Documentation

4.41.2.1 anonymous enum

Enumerator

InstructionOnly Show only instructions in disassembly output.

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

- Shader.h

4.42 LLGL::ShaderProgram Class Reference

Shader program interface.

`#include <ShaderProgram.h>`

Public Member Functions

- virtual void AttachShader (Shader &shader)=0

  Attaches the specified shader to this shader program.
- virtual bool LinkShaders ()=0

  Links all attached shaders to the final shader program.
- virtual std::string QueryInfoLog ()=0

  Returns the information log after the shader linkage.
- virtual std::vector< VertexAttribute > QueryVertexAttributes () const =0

  Returns a list of vertex attributes, which describe all vertex attributes within this shader program.
- virtual std::vector< ConstantBufferDescriptor > QueryConstantBuffers () const =0

  Returns a list of constant buffer descriptors, which describe all constant buffers (also "Uniform Buffer Object") within this shader program.
- virtual std::vector< StorageBufferDescriptor > QueryStorageBuffers () const =0

  Returns a list of storage buffer descriptors, which describe all storage buffers (also "Shader Storage Buffer Object" or "Read/Write Buffer") within this shader program.
- virtual std::vector< UniformDescriptor > QueryUniforms () const =0

  Returns a list of uniform descriptors, which describe all uniforms within this shader program.
- virtual void BindVertexAttributes (const std::vector< VertexAttribute > &vertexAttribs)=0

  Binds the specified vertex attributes to this shader program.
- virtual void BindConstantBuffer (const std::string &name, unsigned int bindingIndex)=0

  Binds the specified constant buffer to this shader.
- virtual void BindStorageBuffer (const std::string &name, unsigned int bindingIndex)=0

  Binds the specified storage buffer to this shader.
- virtual ShaderUniform * LockShaderUniform ()=0

  Locks the shader uniform handler.
- virtual void UnlockShaderUniform ()=0

  Unlocks the shader uniform handler.

4.42.1 Detailed Description

Shader program interface.

4.42.2 Member Function Documentation

4.42.2.1 virtual void LLGL::ShaderProgram::AttachShader (Shader & *shader*) [pure virtual]

Attaches the specified shader to this shader program.

Parameters

in	*shader*	Specifies the shader which is to be attached to this shader program. Each shader type can only be added once for each shader program.

Remarks

This must be called, before "LinkShaders" is called.

Exceptions

std::invalid_argument	If a shader is attached to this shader program, which is not allow in the current state. This will happend if a different shader of the same type has already been attached to this shader program.

See also

Shader::GetType

4.42.2.2 virtual void LLGL::ShaderProgram::BindConstantBuffer (const std::string & *name*, unsigned int *bindingIndex*) [pure virtual]

Binds the specified constant buffer to this shader.

Parameters

in	*name*	Specifies the name of the constant buffer within this shader.
in	*bindingIndex*	Specifies the binding index. This index must match the index which will be used for "RenderContext::BindConstantBuffer".

See also

QueryConstantBuffers
RenderContext::BindConstantBuffer

4.42.2.3 virtual void LLGL::ShaderProgram::BindStorageBuffer (const std::string & *name*, unsigned int *bindingIndex*) [pure virtual]

Binds the specified storage buffer to this shader.

Parameters

in	*name*	Specifies the name of the storage buffer within this shader.
in	*bindingIndex*	Specifies the binding index. This index must match the index which will be used for "RenderContext::BindStorageBuffer".

See also

RenderContext::BindStorageBuffer

4.42.2.4 virtual void LLGL::ShaderProgram::BindVertexAttributes (const std::vector< VertexAttribute > & *vertexAttribs*) [pure virtual]

Binds the specified vertex attributes to this shader program.

Parameters

in	*vertexAttribs*	Specifies the vertex attributes.

Remarks

This is only required for a shader program, which has an attached vertex shader. Moreover, this can only be called after shader compilation but before shader program linking!

See also

AttachShader(VertexShader&)
Shader::Compile
LinkShaders

Exceptions

std::invalid_argument	If the name of an vertex attribute is invalid or the maximal number of available vertex attributes is exceeded.

4.42.2.5 virtual bool LLGL::ShaderProgram::LinkShaders ( ) [pure virtual]

Links all attached shaders to the final shader program.

Returns

True on success, otherwise "QueryInfoLog" can be used to query the reason for failure.

Remarks

Each attached shader must be compiled first!

See also

QueryInfoLog

4.42.2.6 virtual ShaderUniform* LLGL::ShaderProgram::LockShaderUniform ( ) [pure virtual]

Locks the shader uniform handler.

Returns

Pointer to the shader uniform handler or null if the render system does not support individual shader uniforms.

Remarks

This must be called to set individual shader uniforms.

```
auto uniform = shaderProgram->LockShaderUniform();
if (uniform)
{
    uniform->SetUniform("mySampler1", 0);
    uniform->SetUniform("mySampler2", 1);
    uniform->SetUniform("projection", myProjectionMatrix);
}
shaderProgram->UnlockShaderUniform();
```

Note

Only a shader program from an OpenGL render system will return a non-null pointer!

4.42.2.7 virtual std::vector<UniformDescriptor> LLGL::ShaderProgram::QueryUniforms ( ) const [pure virtual]

Returns a list of uniform descriptors, which describe all uniforms within this shader program.

Remarks

Shader uniforms are only supported in OpenGL 2.0+.

4.42.2.8 virtual void LLGL::ShaderProgram::UnlockShaderUniform ( ) [pure virtual]

Unlocks the shader uniform handler.

See also

LockShaderUniform

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

- ShaderProgram.h

4.43 LLGL::ShaderSource Union Reference

Shader source code union.

`#include <Shader.h>`

Classes

- struct GLSL

  Shader source descriptor for GLSL.
- struct HLSL

  Shader source descriptor for HLSL.

Public Member Functions

- ShaderSource (const std::string &sourceCode)

  Specifies the shader source code GLSL.
- ShaderSource (const std::string &sourceCode, const std::string &entryPoint, const std::string &target, int flags=0)

  Specifies the shader source code for HLSL.

Public Attributes

- struct LLGL::ShaderSource::GLSL **sourceGLSL**
- struct LLGL::ShaderSource::HLSL **sourceHLSL**

4.43.1 Detailed Description

Shader source code union.

4.43.2 Constructor & Destructor Documentation

4.43.2.1 LLGL::ShaderSource::ShaderSource (const std::string & *sourceCode*) [inline]

Specifies the shader source code GLSL.

Parameters

in	*sourceCode*	Specifies the shader source code.

Note

Only supported with: OpenGL (for GLSL).

4.43.2.2 LLGL::ShaderSource::ShaderSource (const std::string & *sourceCode*, const std::string & *entryPoint*, const std::string & *target*, int *flags* = 0) [inline]

Specifies the shader source code for HLSL.

Parameters

in	*sourceCode*	Specifies the shader source code.
in	*entryPoint*	Specifies the shader entry point.
in	*target*	Specifies the shader version target (see https://msdn.microsoft.↩ com/en-us/library/windows/desktop/jj215820(v=vs.85).aspx).
in	*flags*	Specifies optional compilation flags. This can be a bitwise OR combination of the 'ShaderCompileFlags' enumeration entries. By default 0.

Note

Only supported with: Direct3D 11, Direct3D 12 (for HLSL).

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

- Shader.h

4.44 LLGL::ShaderUniform Class Reference

Shader uniform setter interface.

`#include <ShaderUniform.h>`

Public Member Functions

- virtual void **SetUniform** (int location, const int value)=0
- virtual void **SetUniform** (int location, const Gs::Vector2i &value)=0
- virtual void **SetUniform** (int location, const Gs::Vector3i &value)=0
- virtual void **SetUniform** (int location, const Gs::Vector4i &value)=0
- virtual void **SetUniform** (int location, const float value)=0
- virtual void **SetUniform** (int location, const Gs::Vector2f &value)=0
- virtual void **SetUniform** (int location, const Gs::Vector3f &value)=0
- virtual void **SetUniform** (int location, const Gs::Vector4f &value)=0
- virtual void **SetUniform** (int location, const Gs::Matrix2f &value)=0
- virtual void **SetUniform** (int location, const Gs::Matrix3f &value)=0
- virtual void **SetUniform** (int location, const Gs::Matrix4f &value)=0
- virtual void **SetUniform** (const std::string &name, const int value)=0
- virtual void **SetUniform** (const std::string &name, const Gs::Vector2i &value)=0
- virtual void **SetUniform** (const std::string &name, const Gs::Vector3i &value)=0
- virtual void **SetUniform** (const std::string &name, const Gs::Vector4i &value)=0
- virtual void **SetUniform** (const std::string &name, const float value)=0
- virtual void **SetUniform** (const std::string &name, const Gs::Vector2f &value)=0
- virtual void **SetUniform** (const std::string &name, const Gs::Vector3f &value)=0
- virtual void **SetUniform** (const std::string &name, const Gs::Vector4f &value)=0
- virtual void **SetUniform** (const std::string &name, const Gs::Matrix2f &value)=0
- virtual void **SetUniform** (const std::string &name, const Gs::Matrix3f &value)=0
- virtual void **SetUniform** (const std::string &name, const Gs::Matrix4f &value)=0
- virtual void **SetUniformArray** (int location, const int *value, std::size_t count)=0
- virtual void **SetUniformArray** (int location, const Gs::Vector2i *value, std::size_t count)=0
- virtual void **SetUniformArray** (int location, const Gs::Vector3i *value, std::size_t count)=0
- virtual void **SetUniformArray** (int location, const Gs::Vector4i *value, std::size_t count)=0
- virtual void **SetUniformArray** (int location, const float *value, std::size_t count)=0
- virtual void **SetUniformArray** (int location, const Gs::Vector2f *value, std::size_t count)=0
- virtual void **SetUniformArray** (int location, const Gs::Vector3f *value, std::size_t count)=0
- virtual void **SetUniformArray** (int location, const Gs::Vector4f *value, std::size_t count)=0
- virtual void **SetUniformArray** (int location, const Gs::Matrix2f *value, std::size_t count)=0
- virtual void **SetUniformArray** (int location, const Gs::Matrix3f *value, std::size_t count)=0
- virtual void **SetUniformArray** (int location, const Gs::Matrix4f *value, std::size_t count)=0
- virtual void **SetUniformArray** (const std::string &name, const int *value, std::size_t count)=0
- virtual void **SetUniformArray** (const std::string &name, const Gs::Vector2i *value, std::size_t count)=0

- virtual void **SetUniformArray** (const std::string &name, const Gs::Vector3i *value, std::size_t count)=0
- virtual void **SetUniformArray** (const std::string &name, const Gs::Vector4i *value, std::size_t count)=0
- virtual void **SetUniformArray** (const std::string &name, const float *value, std::size_t count)=0
- virtual void **SetUniformArray** (const std::string &name, const Gs::Vector2f *value, std::size_t count)=0
- virtual void **SetUniformArray** (const std::string &name, const Gs::Vector3f *value, std::size_t count)=0
- virtual void **SetUniformArray** (const std::string &name, const Gs::Vector4f *value, std::size_t count)=0
- virtual void **SetUniformArray** (const std::string &name, const Gs::Matrix2f *value, std::size_t count)=0
- virtual void **SetUniformArray** (const std::string &name, const Gs::Matrix3f *value, std::size_t count)=0
- virtual void **SetUniformArray** (const std::string &name, const Gs::Matrix4f *value, std::size_t count)=0

4.44.1 Detailed Description

Shader uniform setter interface.

Remarks

This is only used by the OpenGL render system.

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

- ShaderUniform.h

4.45 LLGL::GraphicsAPIDependentStateDescriptor::StateOpenGLDescriptor Struct Reference

Public Attributes

- bool flipViewportVertical

4.45.1 Member Data Documentation

4.45.1.1 bool LLGL::GraphicsAPIDependentStateDescriptor::StateOpenGLDescriptor::flipViewportVertical

Specifies whether to flip the viewport setttings vertical. By default false.

Remarks

If this is true, the front facing will be inverted everytime "BindGraphicsPipeline" is called, and everytime the viewports and scissors are set, their origin will be lower-left instead of upper-left. This can be used for compatability with other renderers such as Direct3D when a render target is bound.

See also

RasterizerDescriptor::frontCCW
RenderContext::BindGraphicsPipeline

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

- RenderContextFlags.h

4.46 LLGL::StencilDescriptor Struct Reference

Stencil state descriptor structure.

`#include <GraphicsPipelineFlags.h>`

Public Attributes

- bool testEnabled = false

  Specifies whether the stencil test is enabled or disabled.
- StencilFaceDescriptor front

  Specifies the front face settings for the stencil test.
- StencilFaceDescriptor back

  Specifies the back face settings for the stencil test.

4.46.1 Detailed Description

Stencil state descriptor structure.

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

- GraphicsPipelineFlags.h

4.47 LLGL::StencilFaceDescriptor Struct Reference

Stencil face descriptor structure.

`#include <GraphicsPipelineFlags.h>`

Public Attributes

- StencilOp stencilFailOp = StencilOp::Keep

  Specifies the operation to take when the stencil test fails.
- StencilOp depthFailOp = StencilOp::Keep

  Specifies the operation to take when the stencil test passes but the depth test fails.
- StencilOp depthPassOp = StencilOp::Keep

  Specifies the operation to take when both the stencil test and the depth test pass.
- CompareOp compareOp = CompareOp::Less

  Specifies the stencil compare operation.
- std::uint32_t **compareMask** = 0
- std::uint32_t **writeMask** = 0
- std::uint32_t **reference** = 0

4.47.1 Detailed Description

Stencil face descriptor structure.

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

- GraphicsPipelineFlags.h

4.48 LLGL::StorageBuffer Class Reference

Storage buffer (also "Shader Sotrage Object" or "Read/Write Buffer") interface.

`#include <StorageBuffer.h>`

4.48.1 Detailed Description

Storage buffer (also "Shader Sotrage Object" or "Read/Write Buffer") interface.

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

- StorageBuffer.h

4.49 LLGL::StorageBufferDescriptor Struct Reference

Storage buffer descriptor structure.

`#include <StorageBuffer.h>`

Public Attributes

- std::string name
  Storage buffer name.
- unsigned int index = 0
  Index of the storage buffer within the respective shader.

4.49.1 Detailed Description

Storage buffer descriptor structure.

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

- StorageBuffer.h

4.50 LLGL::Texture Class Reference

Texture interface.

`#include <Texture.h>`

Public Member Functions

- TextureType GetType () const

  Returns the texture type. This type can only be changed by the render system.
- virtual Gs::Vector3i QueryMipLevelSize (int mipLevel) const =0

  Returns the texture size for the specified MIP-level.

Protected Member Functions

- void **SetType** (const TextureType type)

4.50.1 Detailed Description

Texture interface.

4.50.2 Member Function Documentation

4.50.2.1 TextureType LLGL::Texture::GetType ( ) const [inline]

Returns the texture type. This type can only be changed by the render system.

See also

RenderSystem::WriteTexture1D
RenderSystem::WriteTexture2D
RenderSystem::WriteTexture3D
RenderSystem::WriteTextureCube

4.50.2.2 virtual Gs::Vector3i LLGL::Texture::QueryMipLevelSize (int *mipLevel*) const [pure virtual]

Returns the texture size for the specified MIP-level.

Parameters

in	*mipLevel*	Specifies the MIP-map level to querey from. The first and largest MIP-map is level zero. If this level is greater than or equal to the number of MIP-maps this texture has, the return value is undefined (i.e. depends on the render system).

See also

RenderContext::GenerateMips

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

- Texture.h

4.51 LLGL::TextureDescriptor::Texture1DDescriptor Struct Reference

Public Attributes

- TextureType **type**
- int **width**
- unsigned int **layers**

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

- TextureFlags.h

4.52 LLGL::TextureDescriptor::Texture2DDescriptor Struct Reference

Public Attributes

- TextureType **type**
- int **width**
- int **height**
- unsigned int **layers**

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

- TextureFlags.h

4.53 LLGL::TextureDescriptor::Texture3DDescriptor Struct Reference

Public Attributes

- TextureType **type**
- int **width**
- int **height**
- int **depth**

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

- TextureFlags.h

4.54 LLGL::TextureDescriptor::TextureCubeDescriptor Struct Reference

Public Attributes

- TextureType **type**
- int **width**
- int **height**
- unsigned int **layers**

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

- TextureFlags.h

4.55 LLGL::TextureDescriptor Union Reference

Texture descriptor union.

`#include <TextureFlags.h>`

Classes

- struct Texture1DDescriptor
- struct Texture2DDescriptor
- struct Texture3DDescriptor
- struct TextureCubeDescriptor

Public Attributes

- TextureType **type**
- struct LLGL::TextureDescriptor::Texture1DDescriptor **texture1DDesc**
- struct LLGL::TextureDescriptor::Texture2DDescriptor **texture2DDesc**
- struct LLGL::TextureDescriptor::Texture3DDescriptor **texture3DDesc**
- struct LLGL::TextureDescriptor::TextureCubeDescriptor **textureCubeDesc**

4.55.1 Detailed Description

Texture descriptor union.

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

- TextureFlags.h

4.56 LLGL::Timer Class Reference

Public Types

- using **FrameCount** = unsigned long long

Public Member Functions

- virtual void Start ()=0

  Starts the timer.
- virtual double Stop ()=0

  Stops the timer and returns the elapsed time since "Start" was called.
- virtual double GetFrequency () const =0

  Returns the frequency this timer can measure time (e.g. for milliseconds this is 1000.0).
- void MeasureTime ()

  Measures the time (elapsed time, and frame count) for each frame.
- void ResetFrameCounter ()

  Restes the frame counter.
- double GetDeltaTime () const

  Returns the elapsed time (in seconds) between the current and the previous frame.
- FrameCount GetFrameCount () const

  Returns the number of counted frames.

Static Public Member Functions

- static std::unique_ptr< Timer > Create ()

  Creates a platform specific timer object.

4.56.1 Member Function Documentation

4.56.1.1 double LLGL::Timer::GetDeltaTime ( ) const [inline]

Returns the elapsed time (in seconds) between the current and the previous frame.

Remarks

This requires that "MeasureTime" is called once every frame.

See also

MeasureTime

4.56.1.2 FrameCount LLGL::Timer::GetFrameCount ( ) const [inline]

Returns the number of counted frames.

Remarks

This requires that "MeasureTime" is called once every frame.

See also

MeasureTime

4.56.1.3 void LLGL::Timer::MeasureTime ( )

Measures the time (elapsed time, and frame count) for each frame.

See also

GetDeltaTime
GetFrameCount()

4.56.1.4 void LLGL::Timer::ResetFrameCounter ( )

Restes the frame counter.

See also

GetFrameCount

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

- Timer.h

4.57 LLGL::UniformDescriptor Struct Reference

Shader uniform descriptor structure.

`#include <ShaderUniform.h>`

Public Attributes

- std::string **name**
- UniformType **type** = UniformType::Float
- int **location** = 0
- unsigned int **size** = 0

4.57.1 Detailed Description

Shader uniform descriptor structure.

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

- ShaderUniform.h

4.58 LLGL::VertexAttribute Struct Reference

Vertex attribute class.

`#include <VertexAttribute.h>`

Public Attributes

- DataType dataType = DataType::Float32

  Data type of the vertex attribute components. By default DataType::Float32.
- bool conversion = false

  Specifies whether non-floating-point data types are to be converted to floating-points. By default false.
- bool perInstance = false

  Specifies whether this is a per-instance data. If false, this is a per-vertex data.
- unsigned int components = 4

  Number of components: 1, 2, 3, or 4. By default 4.
- unsigned int offset = 0

  Byte offset for within each vertex. By default 0.
- std::string name

  Vertex attribute name (for GLSL) or semantic name (for HLSL).
- unsigned int semanticIndex = 0

  Semantic index (only relevant for HLSL).

4.58.1 Detailed Description

Vertex attribute class.

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

- VertexAttribute.h

4.59 LLGL::VertexBuffer Class Reference

Vertex buffer interface.

`#include <VertexBuffer.h>`

Public Member Functions

- const VertexFormat & **GetVertexFormat** () const

Protected Member Functions

- void **SetVertexFormat** (const VertexFormat &vertexFormat)

4.59.1 Detailed Description

Vertex buffer interface.

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

- VertexBuffer.h

4.60 LLGL::VertexFormat Class Reference

Vertex format descriptor class.

`#include <VertexFormat.h>`

Public Member Functions

- void AddAttribute (const std::string &name, const DataType dataType, unsigned int components, bool conversion=false, bool perInstance=false)

  Adds a new vertex attribute to this vertex format with a specified name (used for GLSL).
- void AddAttribute (const std::string &semanticName, unsigned int semanticIndex, const DataType dataType, unsigned int components, bool conversion=false, bool perInstance=false)

  Adds a new vertex attribute to this vertex format with a specified semantic (used for HLSL).
- const std::vector< VertexAttribute > & GetAttributes () const

  Returns the list of all vertex attributes.
- unsigned int GetFormatSize () const

  Returns the size of this vertex format (in bytes).

4.60.1 Detailed Description

Vertex format descriptor class.

Remarks

A vertex format is required to describe how the vertex attributes are supported inside a vertex buffer.

See also

VertexBuffer

4.60.2 Member Function Documentation

4.60.2.1 void LLGL::VertexFormat::AddAttribute (const std::string & *name*, const DataType *dataType*, unsigned int *components*, bool *conversion* = `false`, bool *perInstance* = `false`)

Adds a new vertex attribute to this vertex format with a specified name (used for GLSL).

Parameters

in	*name*	Specifies the attribute name.
in	*dataType*	Specifies the data type of the attribute components.
in	*components*	Specifies the number of attribute components. This must be 1, 2, 3, or 4.
in	*conversion*	Specifies whether to convert integral vertex attributes to normalized floating-point types. By default false.
in	*perInstance*	Specifies whether this is per-instance data. If false, this is per-vertex data. By default false.

Remarks

This is equivalent to:

```
AddAttribute(name, 0, dataType, components, conversion);
```

Exceptions

std::invalid_argument	If 'components' is neither 1, 2, 3, nor 4.

See also

AddAttribute(const std::string&, unsigned int, const DataType, unsigned int, bool, bool)

4.60.2.2 void LLGL::VertexFormat::AddAttribute (const std::string & *semanticName*, unsigned int *semanticIndex*, const DataType *dataType*, unsigned int *components*, bool *conversion* = `false`, bool *perInstance* = `false`)

Adds a new vertex attribute to this vertex format with a specified semantic (used for HLSL).

Parameters

`in`	*semanticName*	Specifies the semantic name (For Direct3D).
`in`	*semanticIndex*	Specifies the semantic index (For Direct3D).
`in`	*dataType*	Specifies the data type of the attribute components.
`in`	*components*	Specifies the number of attribute components. This must be 1, 2, 3, or 4.
`in`	*conversion*	Specifies whether to convert integral vertex attributes to normalized floating-point types. By default false.
`in`	*perInstance*	Specifies whether this is per-instance data. If false, this is per-vertex data. By default false.

Exceptions

std::invalid_argument	If 'components' is neither 1, 2, 3, nor 4.

4.60.2.3 const std::vector<VertexAttribute>& LLGL::VertexFormat::GetAttributes ( ) const `[inline]`

Returns the list of all vertex attributes.

See also

AddAttribute

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

- VertexFormat.h

4.61 LLGL::VideoAdapterDescriptor Struct Reference

Video adapter descriptor structure.

`#include <VideoAdapter.h>`

Public Attributes

- std::wstring name
  Hardware adapter name (name of the GPU).
- std::string vendor
  Vendor name.
- unsigned long long videoMemory = 0
  Video memory size (in bytes).
- std::vector< VideoOutput > outputs
  Adapter outputs.

4.61.1 Detailed Description

Video adapter descriptor structure.

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

- VideoAdapter.h

4.62 LLGL::VideoDisplayMode Struct Reference

Public Attributes

- unsigned int **width** = 0
- unsigned int **height** = 0
- unsigned int **refreshRate** = 0

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

- VideoAdapter.h

4.63 LLGL::VideoModeDescriptor Struct Reference

Public Attributes

- Size resolution
  Screen resolution.
- int colorDepth = 32
  Color bit depth. Should be 24 or 32. By default 32.
- bool fullscreen = false
  Specifies whether to enable fullscreen mode or windowed mode. By default windowed mode.
- SwapChainMode swapChainMode = SwapChainMode::DoubleBuffering
  Swap chain buffering mode.

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

- RenderContextDescriptor.h

4.64 LLGL::VideoOutput Struct Reference

Public Attributes

- std::vector< VideoDisplayMode > **displayModes**

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

- VideoAdapter.h

4.65 LLGL::Viewport Struct Reference

Viewport dimensions.

`#include <RenderContextFlags.h>`

Public Member Functions

- **Viewport** (const Viewport &)=default
- **Viewport** (float x, float y, float width, float height)
- **Viewport** (float x, float y, float width, float height, float minDepth, float maxDepth)

Public Attributes

- float x = 0.0f
  Left-top X coordinate.
- float y = 0.0f
  Left-top Y coordinate.
- float width = 0.0f
  Right-bottom width.
- float height = 0.0f
  Right-bottom height.
- float minDepth = 0.0f
  Minimal depth range.
- float maxDepth = 1.0f
  Maximal depth range.

4.65.1 Detailed Description

Viewport dimensions.

Remarks

A viewport is in screen coordinates where the origin is in the left-top corner.

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

- RenderContextFlags.h

4.66 LLGL::VsyncDescriptor Struct Reference

Public Attributes

- bool enabled = false

  Specifies whether vertical-synchronisation (Vsync) is enabled or disabled. By default disabled.
- unsigned int refreshRate = 60

  Refresh rate (in Hz). By default 60.
- unsigned int interval = 1

  Synchronisation interval. Can be 1, 2, 3, or 4. If Vsync is disabled, this value is implicit zero.

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

- RenderContextDescriptor.h

4.67 LLGL::Window Class Reference

Classes

- class EventListener

Public Member Functions

- virtual void **SetPosition** (const Point &position)=0
- virtual Point **GetPosition** () const =0
- virtual void **SetSize** (const Size &size, bool useClientArea=true)=0
- virtual Size **GetSize** (bool useClientArea=true) const =0
- virtual void **SetTitle** (const std::wstring &title)=0
- virtual std::wstring **GetTitle** () const =0
- virtual void **Show** (bool show=true)=0
- virtual bool **IsShown** () const =0
- virtual WindowDescriptor QueryDesc () const =0

  Query a window descriptor, which describes the current state of this window.
- virtual void SetDesc (const WindowDescriptor &desc)=0

  Sets the new window descriptor.
- virtual void Recreate (const WindowDescriptor &desc)=0

  Recreates the internal window object. This may invalidate the native handle previously returned by "GetNative↩ Handle".
- virtual void GetNativeHandle (void *nativeHandle) const =0

  Returns the native window handle.
- bool ProcessEvents ()

  Processes the events for this window (i.e. mouse movement, key presses etc.).
- void **AddEventListener** (const std::shared_ptr< EventListener > &eventListener)
- void **RemoveEventListener** (const EventListener *eventListener)
- void **PostKeyDown** (Key keyCode)
- void **PostKeyUp** (Key keyCode)
- void **PostDoubleClick** (Key keyCode)
- void **PostChar** (wchar_t chr)
- void **PostWheelMotion** (int motion)
- void **PostLocalMotion** (const Point &position)
- void **PostGlobalMotion** (const Point &motion)
- void **PostResize** (const Size &clientAreaSize)
- void PostQuit ()

  Posts the 'OnQuit' event to all event listeners.

Static Public Member Functions

- static std::unique_ptr< Window > **Create** (const WindowDescriptor &desc)

Protected Member Functions

- virtual void **ProcessSystemEvents** ()=0

4.67.1 Member Function Documentation

4.67.1.1 virtual void LLGL::Window::GetNativeHandle (void * *nativeHandle*) const [pure virtual]

Returns the native window handle.

Remarks

This must be casted to a platform specific structure:

```
#include <LLGL/Platform/NativeHandle.h>
//...
LLGL::NativeHandle handle;
window.GetNativeHandle(&handle);
```

4.67.1.2 void LLGL::Window::PostQuit ( )

Posts the 'OnQuit' event to all event listeners.

Remarks

If at least one event listener returns false within the "OnQuit" callback, the window will not quit. If all event listener return true within the "OnQuit" callback, "ProcessEvents" will returns false from now on.

4.67.1.3 bool LLGL::Window::ProcessEvents ( )

Processes the events for this window (i.e. mouse movement, key presses etc.).

Returns

Once the "PostQuit" function was called on this window object, this function returns false. This will happend, when the user clicks on the close button.

4.67.1.4 virtual void LLGL::Window::Recreate (const WindowDescriptor & *desc*) [pure virtual]

Recreates the internal window object. This may invalidate the native handle previously returned by "GetNative↩ Handle".

See also

GetNativeHandle

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

- Window.h

4.68 LLGL::WindowDescriptor Struct Reference

Window descriptor structure.

`#include <Window.h>`

Public Attributes

- std::wstring **title**
- Point position

  Window position (relative to the client area).
- Size size

  Client area size.
- bool **visible** = false
- bool **borderless** = false
- bool **resizable** = false
- bool **acceptDropFiles** = false
- bool **preventForPowerSafe** = false
- bool **centered** = false
- const void * windowContext = nullptr

  Window context handle.

4.68.1 Detailed Description

Window descriptor structure.

4.68.2 Member Data Documentation

4.68.2.1 const void* LLGL::WindowDescriptor::windowContext = nullptr

Window context handle.

Remarks

If used, this must be casted from a platform specific structure:

```
#include <LLGL/Platform/NativeHandle.h>
//...
LLGL::NativeContextHandle handle;
//handle.parentWindow = ...
windowDesc.windowContext = reinterpret_cast<const void*>(&handle);
```

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

- Window.h

Index