

Farouq Adepetu's Rendering Engine

Generated by Doxygen 1.9.4

1 Namespace Index	1
1.1 Namespace List	1
2 Class Index	3
2.1 Class List	3
3 File Index	5
3.1 File List	5
4 Namespace Documentation	7
4.1 RenderingEngine Namespace Reference	7
4.1.1 Detailed Description	9
4.1.2 Function Documentation	9
4.1.2.1 Backward()	10
4.1.2.2 CreateChildWindow()	11
4.1.2.3 CreateControlWindow()	12
4.1.2.4 CreateParentWindow()	12
4.1.2.5 Down()	13
4.1.2.6 Foward()	13
4.1.2.7 GetHeight()	14
4.1.2.8 GetWidth()	14
4.1.2.9 GetX()	14
4.1.2.10 GetY()	14
4.1.2.11 InitializeTime()	14
4.1.2.12 Left()	14
4.1.2.13 LookAt()	14
4.1.2.14 operator*() [1/3]	15
4.1.2.15 operator*() [2/3]	15
4.1.2.16 operator*() [3/3]	15
4.1.2.17 operator+()	15
4.1.2.18 operator-()	16
4.1.2.19 Reset()	16
4.1.2.20 Right()	16
4.1.2.21 RotateCameraLeftRight()	16
4.1.2.22 RotateCameraUpDown()	17
4.1.2.23 SetProperties() [1/3]	17
4.1.2.24 SetProperties() [2/3]	17
4.1.2.25 SetProperties() [3/3]	18
4.1.2.26 Start()	18
4.1.2.27 Stop()	18
4.1.2.28 Tick()	19
4.1.2.29 Up()	19
4.1.2.30 UpdateProjectionMatrix() [1/2]	19

4.1.2.31 UpdateProjectionMatrix() [2/2]	19
4.1.2.32 UpdateViewMatrix()	20
5 Class Documentation	21
5.1 RenderingEngine::Camera Struct Reference	21
5.1.1 Detailed Description	21
5.2 RenderingEngine::Color Class Reference	21
5.2.1 Detailed Description	22
5.2.2 Constructor & Destructor Documentation	22
5.2.2.1 Color() [1/2]	23
5.2.2.2 Color() [2/2]	23
5.2.3 Member Function Documentation	23
5.2.3.1 GetAlpha()	23
5.2.3.2 GetBlue()	23
5.2.3.3 GetColor()	23
5.2.3.4 GetGreen()	24
5.2.3.5 GetRed()	24
5.2.3.6 operator*=() [1/2]	24
5.2.3.7 operator*=() [2/2]	24
5.2.3.8 operator+=()	24
5.2.3.9 operator-=()	25
5.2.3.10 SetAlpha()	25
5.2.3.11 SetBlue()	25
5.2.3.12 SetColor()	25
5.2.3.13 SetGreen()	25
5.2.3.14 SetRed()	26
5.3 RenderingEngine::DepthStencilBuffer Class Reference	26
5.3.1 Detailed Description	26
5.3.2 Constructor & Destructor Documentation	27
5.3.2.1 DepthStencilBuffer() [1/2]	27
5.3.2.2 DepthStencilBuffer() [2/2]	27
5.3.3 Member Function Documentation	27
5.3.3.1 ClearDepthStencilBuffer()	27
5.3.3.2 CreateDepthStencilBufferAndView()	28
5.3.3.3 GetDepthStencilFormat()	28
5.3.3.4 ReleaseBuffer()	28
5.4 RenderingEngine::DeviceResources Class Reference	29
5.4.1 Detailed Description	30
5.4.2 Constructor & Destructor Documentation	30
5.4.2.1 ~DeviceResources()	30
5.4.3 Member Function Documentation	30
5.4.3.1 AfterTextDraw()	30

5.4.3.2 BeforeTextDraw()	30
5.4.3.3 Execute()	31
5.4.3.4 FlushCommandQueue()	31
5.4.3.5 GetBackBufferFormat()	31
5.4.3.6 GetCBVSRVUAVSize()	31
5.4.3.7 GetCommandList()	31
5.4.3.8 GetCurrentFrame()	31
5.4.3.9 GetDepthStencilFormat()	32
5.4.3.10 GetDevice()	32
5.4.3.11 GetInstance()	32
5.4.3.12 GetTextResources()	32
5.4.3.13 NextFrame()	33
5.4.3.14 Present()	33
5.4.3.15 Resize()	33
5.4.3.16 RTBufferTransition()	33
5.4.3.17 Signal()	34
5.4.3.18 UpdateCurrentFrameFenceValue()	34
5.4.3.19 WaitForGPU()	34
5.4.4 Member Data Documentation	34
5.4.4.1 NUM_OF_FRAMES	34
5.5 DirectXException Class Reference	34
5.5.1 Detailed Description	35
5.5.2 Constructor & Destructor Documentation	35
5.5.2.1 DirectXException()	35
5.5.3 Member Function Documentation	35
5.5.3.1 ErrorMsg()	35
5.6 RenderingEngine::DrawArguments Struct Reference	36
5.6.1 Detailed Description	36
5.7 RenderingEngine::DynamicBuffer Class Reference	36
5.7.1 Detailed Description	37
5.7.2 Constructor & Destructor Documentation	37
5.7.2.1 DynamicBuffer() [1/3]	37
5.7.2.2 DynamicBuffer() [2/3]	37
5.7.2.3 DynamicBuffer() [3/3]	38
5.7.2.4 ~DynamicBuffer()	38
5.7.3 Member Function Documentation	38
5.7.3.1 CopyData()	38
5.7.3.2 CreateConstantBufferView()	39
5.7.3.3 CreateDynamicBuffer() [1/2]	39
5.7.3.4 CreateDynamicBuffer() [2/2]	39
5.7.3.5 GetGPUAddress()	40
5.7.3.6 GetIndexBufferView()	40

5.7.3.7 GetVertexBufferView()	40
5.7.3.8 ReleaseBuffer()	40
5.8 RenderingEngine::MultiSampling Class Reference	40
5.8.1 Detailed Description	41
5.8.2 Constructor & Destructor Documentation	41
5.8.2.1 MultiSampling() [1/2]	42
5.8.2.2 MultiSampling() [2/2]	42
5.8.3 Member Function Documentation	42
5.8.3.1 CheckMultiSamplingSupport()	42
5.8.3.2 ClearDepthStencilBuffer()	43
5.8.3.3 ClearRenderTargetBuffer()	43
5.8.3.4 CreateDepthStencilBufferAndView()	43
5.8.3.5 CreateRenderTargetBufferAndView()	44
5.8.3.6 GetDepthStencilFormat()	44
5.8.3.7 GetRenderTargetBuffer()	45
5.8.3.8 GetRenderTargetFormat()	45
5.8.3.9 ReleaseBuffers()	45
5.8.3.10 Transition()	45
5.9 RenderingEngine::OrthographicProjection Struct Reference	45
5.9.1 Detailed Description	46
5.10 RenderingEngine::PerspectiveProjection Struct Reference	46
5.10.1 Detailed Description	46
5.11 RenderingEngine::RenderScene Class Reference	46
5.11.1 Detailed Description	50
5.11.2 Member Function Documentation	50
5.11.2.1 AfterRender()	50
5.11.2.2 AfterRenderObjects()	50
5.11.2.3 AfterRenderText()	50
5.11.2.4 BeforeRenderObjects()	51
5.11.2.5 BeforeRenderText()	51
5.11.2.6 CompileShader() [1/2]	51
5.11.2.7 CompileShader() [2/2]	51
5.11.2.8 CopyDataIntoDynamicBuffer() [1/2]	52
5.11.2.9 CopyDataIntoDynamicBuffer() [2/2]	52
5.11.2.10 CreateDescriptorRange() [1/2]	53
5.11.2.11 CreateDescriptorRange() [2/2]	53
5.11.2.12 CreateDescriptorTable() [1/2]	54
5.11.2.13 CreateDescriptorTable() [2/2]	54
5.11.2.14 CreateDynamicBuffer() [1/4]	54
5.11.2.15 CreateDynamicBuffer() [2/4]	55
5.11.2.16 CreateDynamicBuffer() [3/4]	55
5.11.2.17 CreateDynamicBuffer() [4/4]	56

5.11.2.18 CreateInputElementDescription() [1/2]	56
5.11.2.19 CreateInputElementDescription() [2/2]	57
5.11.2.20 CreatePSO() [1/2]	57
5.11.2.21 CreatePSO() [2/2]	58
5.11.2.22 CreateRootConstants() [1/2]	59
5.11.2.23 CreateRootConstants() [2/2]	59
5.11.2.24 CreateRootDescriptor() [1/2]	59
5.11.2.25 CreateRootDescriptor() [2/2]	60
5.11.2.26 CreateRootSignature() [1/4]	60
5.11.2.27 CreateRootSignature() [2/4]	60
5.11.2.28 CreateRootSignature() [3/4]	61
5.11.2.29 CreateRootSignature() [4/4]	61
5.11.2.30 CreateStaticBuffer() [1/6]	61
5.11.2.31 CreateStaticBuffer() [2/6]	62
5.11.2.32 CreateStaticBuffer() [3/6]	62
5.11.2.33 CreateStaticBuffer() [4/6]	63
5.11.2.34 CreateStaticBuffer() [5/6]	63
5.11.2.35 CreateStaticBuffer() [6/6]	63
5.11.2.36 CreateStaticSampler() [1/2]	64
5.11.2.37 CreateStaticSampler() [2/2]	64
5.11.2.38 CreateTextureViewHeap()	66
5.11.2.39 ExecuteAndFlush()	66
5.11.2.40 LinkDynamicBuffer() [1/2]	66
5.11.2.41 LinkDynamicBuffer() [2/2]	67
5.11.2.42 LinkPSOAndRootSignature() [1/2]	67
5.11.2.43 LinkPSOAndRootSignature() [2/2]	68
5.11.2.44 LinkStaticBuffer() [1/2]	68
5.11.2.45 LinkStaticBuffer() [2/2]	68
5.11.2.46 LinkTexture() [1/2]	70
5.11.2.47 LinkTexture() [2/2]	70
5.11.2.48 LinkTextureViewHeap()	70
5.11.2.49 LoadShader() [1/2]	71
5.11.2.50 LoadShader() [2/2]	71
5.11.2.51 RemoveShader() [1/2]	71
5.11.2.52 RemoveShader() [2/2]	71
5.11.2.53 RenderObject()	72
5.11.2.54 RenderText()	72
5.11.2.55 Resize()	73
5.11.2.56 SetConstants()	73
5.12 RenderingEngine::RenderTargetBuffer Class Reference	73
5.12.1 Detailed Description	74
5.12.2 Constructor & Destructor Documentation	74

5.12.2.1 RenderTargetBuffer() [1/2]	74
5.12.2.2 RenderTargetBuffer() [2/2]	75
5.12.3 Member Function Documentation	75
5.12.3.1 ClearRenderTargetBuffer()	75
5.12.3.2 CreateRenderTargetBufferAndView()	76
5.12.3.3 GetRenderTargetBuffer() [1/2]	76
5.12.3.4 GetRenderTargetBuffer() [2/2]	76
5.12.3.5 GetRenderTargetFormat()	76
5.12.3.6 ReleaseBuffer()	77
5.13 RenderingEngine::StaticBuffer Class Reference	77
5.13.1 Detailed Description	78
5.13.2 Constructor & Destructor Documentation	78
5.13.2.1 StaticBuffer() [1/4]	78
5.13.2.2 StaticBuffer() [2/4]	78
5.13.2.3 StaticBuffer() [3/4]	79
5.13.2.4 StaticBuffer() [4/4]	79
5.13.3 Member Function Documentation	79
5.13.3.1 CreateStaticBuffer() [1/3]	79
5.13.3.2 CreateStaticBuffer() [2/3]	80
5.13.3.3 CreateStaticBuffer() [3/3]	80
5.13.3.4 CreateTexture2DMSView()	81
5.13.3.5 CreateTexture2DView()	81
5.13.3.6 GetIndexBufferView()	81
5.13.3.7 GetVertexBufferView()	82
5.13.3.8 ReleaseBuffer()	82
5.14 RenderingEngine::SwapChain Class Reference	82
5.14.1 Detailed Description	83
5.14.2 Constructor & Destructor Documentation	83
5.14.2.1 SwapChain() [1/2]	83
5.14.2.2 SwapChain() [2/2]	83
5.14.3 Member Function Documentation	84
5.14.3.1 ClearCurrentBackBuffer()	84
5.14.3.2 ClearDepthStencilBuffer()	84
5.14.3.3 CreateDepthStencilBufferAndView()	85
5.14.3.4 CreateRenderTargetBuffersAndViews()	85
5.14.3.5 CreateSwapChain()	86
5.14.3.6 GetBackBufferFormat()	86
5.14.3.7 GetCurrentBackBuffer()	86
5.14.3.8 GetCurrentBackBufferIndex()	87
5.14.3.9 GetDepthStencilFormat()	87
5.14.3.10 GetNumRenderTargetBuffers()	87
5.14.3.11 Present()	87

5.14.3.12 ReleaseBuffers()	87
5.14.3.13 Transition()	87
5.15 RenderingEngine::Text Class Reference	88
5.15.1 Detailed Description	88
5.15.2 Constructor & Destructor Documentation	88
5.15.2.1 Text()	89
5.15.3 Member Function Documentation	89
5.15.3.1 GetTextColor()	89
5.15.3.2 GetTextLocation()	89
5.15.3.3 GetTextSize()	89
5.15.3.4 GetTextString()	90
5.15.3.5 SetTextColor()	90
5.15.3.6 SetTextLocation()	90
5.15.3.7 SetTextSize()	90
5.15.3.8 SetTextString()	90
5.16 RenderingEngine::TextResources Class Reference	91
5.16.1 Detailed Description	91
5.16.2 Constructor & Destructor Documentation	91
5.16.2.1 TextResources()	91
5.16.3 Member Function Documentation	92
5.16.3.1 AfterRenderText()	92
5.16.3.2 BeforeRenderText()	92
5.16.3.3 GetDirect2DDeviceContext()	92
5.16.3.4 GetDirectWriteFactory()	92
5.16.3.5 ResetBuffers()	93
5.16.3.6 ResizeBuffers()	93
5.17 RenderingEngine::Time Struct Reference	93
5.17.1 Detailed Description	93
5.18 RenderingEngine::Window Struct Reference	94
5.18.1 Detailed Description	94
6 File Documentation	95
6.1 Buffer.h	95
6.2 Camera.h	97
6.3 Color.h	98
6.4 DDSTextureLoader.h	99
6.5 DeviceResources.h	99
6.6 Direct3DLink.h	101
6.7 DirectXException.h	101
6.8 DrawArguments.h	103
6.9 GameTime.h	103
6.10 Multisampling.h	103

6.11 OrthographicProjection.h	104
6.12 PerspectiveProjection.h	104
6.13 RenderScene.h	105
6.14 SwapChain.h	111
6.15 Text.h	112
6.16 TextResources.h	112
6.17 Window.h	113
Index	115

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

[RenderingEngine](#)

Has classes that are used for rendering objects and text through the Direct3D 12 API [7](#)

Chapter 2

Class Index

2.1 Class List

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

RenderingEngine::Camera	
Simple first person style camera class that lets the viewer explore the 3D scene. It keeps track of the camera coordinate system relative to the world space so that the view matrix can be constructed.	
21	
RenderingEngine::Color	
This class stores a RGBA color in a 4D vector using floats. The range of each component is [0.0, 1.0]. The first componet is red, second component is green, third component is blue and the 4th component is alpha	21
RenderingEngine::DepthStencilBuffer	
A wrapper for depth stencil buffer resources. Uses DirectD 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable	26
RenderingEngine::DeviceResources	
A wrapper for resources that are needed to render objects and text using the Direct3D 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable	29
DirectXException	
A class for handling Direct3D and DXGI errors from functions that return a HRESULT value . .	34
RenderingEngine::DrawArguments	
Data that are used as parameters to draw an object	36
RenderingEngine::DynamicBuffer	
This class stores data in a Direct3D 12 upload buffer. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable	36
RenderingEngine::MultiSampling	
A wrapper for multisampling resources. Uses DirectD 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable	40
RenderingEngine::OrthogrpahicProjection	
A struct that holds the properties for doing orthographics projection	45
RenderingEngine::PerspectiveProjection	
A struct that holds the properties for doing perspective projection	46
RenderingEngine::RenderScene	
This class is used to render a scene using Direct3D 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable	46
RenderingEngine::RenderTargetBuffer	
A wrapper for render target buffer resources. Uses DirectD 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable	73

[RenderingEngine::StaticBuffer](#)

This class stores data in a Direct3D 12 default buffer. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable 77

[RenderingEngine::SwapChain](#)

A wrapper for swap chain resources. Uses DirectD 12 API and DXGI API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable 82

[RenderingEngine::Text](#)

This class is used to help render text. Stores the location of the text, the text string, text size and the color of the text 88

[RenderingEngine::TextResources](#)

A wrapper for resources that are needed to render text using Direct3D 11on12, Direct2D and DirectWrite 91

[RenderingEngine::Time](#)

A struct that holds the properties for time 93

[RenderingEngine::Window](#)

The window struct is used to make a [Window](#) using Win32 API 94

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Buffer.h	??
Camera.h	??
Color.h	??
DDSTextureLoader.h	??
DeviceResources.h	??
Direct3DLink.h	??
DirectXException.h	??
DrawArguments.h	??
GameTime.h	??
Multisampling.h	??
OrthographicProjection.h	??
PerspectiveProjection.h	??
RenderScene.h	??
SwapChain.h	??
Text.h	??
TextResources.h	??
Window.h	??

Chapter 4

Namespace Documentation

4.1 RenderingEngine Namespace Reference

Has classes that are used for rendering objects and text through the Direct3D 12 API.

Classes

- struct [Camera](#)
Simple first person style camera class that lets the viewer explore the 3D scene. It keeps track of the camera coordinate system relative to the world space so that the view matrix can be constructed.
- class [Color](#)
This class stores a RGBA color in a 4D vector using floats. The range of each component is [0.0, 1.0]. The first component is red, second component is green, third component is blue and the 4th component is alpha.
- class [DepthStencilBuffer](#)
A wrapper for depth stencil buffer resources. Uses DirectD 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.
- class [DeviceResources](#)
A wrapper for resources that are needed to render objects and text using the Direct3D 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.
- struct [DrawArguments](#)
Data that are used as parameters to draw an object.
- class [DynamicBuffer](#)
This class stores data in a Direct3D 12 upload buffer. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.
- class [MultiSampling](#)
A wrapper for multisampling resources. Uses DirectD 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.
- struct [OrthographicProjection](#)
A struct that holds the properties for doing orthographics projection.
- struct [PerspectiveProjection](#)
A struct that holds the properties for doing perspective projection.
- class [RenderScene](#)
This class is used to render a scene using Direct3D 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.
- class [RenderTargetBuffer](#)

A wrapper for render target buffer resources. Uses DirectD 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

- class [StaticBuffer](#)

This class stores data in a Direct3D 12 default buffer. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

- class [SwapChain](#)

A wrapper for swap chain resources. Uses DirectD 12 API and DXGI API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

- class [Text](#)

This class is used to help render text. Stores the location of the text, the text string, text size and the color of the text.

- class [TextResources](#)

A wrapper for resources that are needed to render text using Direct3D 11on12, Direct2D and DirectWrite.

- struct [Time](#)

A struct that holds the properties for time.

- struct [Window](#)

The window struct is used to make a [Window](#) using Win32 API.

Enumerations

- enum **BufferTypes** { VERTEX_BUFFER , INDEX_BUFFER , CONSTANT_BUFFER , TEXTURE_BUFFER }
- enum **TextureTypes** { TEX2D , TEX2D_MS }

Functions

- void [SetProperties](#) ([Camera](#) &camera, vec3 position, vec3 x, vec3 y, vec3 z, float linearSpeed, float angularSpeed)

Sets the properties of the camera.
- void [LookAt](#) ([Camera](#) &camera, vec3 position, vec3 target, vec3 up)

Defines the camera space using UVN.
- void [UpdateViewMatrix](#) ([Camera](#) &camera)

After modifying the camera position and/or orientation, call this to rebuild the view transformation matrix.
- void [Left](#) ([Camera](#) &camera, float dt)

Moves the camera left along the camera's x-axis.
- void [Right](#) ([Camera](#) &camera, float dt)

Moves the camera right along the camera's x-axis.
- void [Foward](#) ([Camera](#) &camera, float dt)

Moves the camera foward along the camera's z-axis.
- void [Backward](#) ([Camera](#) &camera, float dt)

Moves the camera backward along the camera's z-axis.
- void [Up](#) ([Camera](#) &camera, float dt)

Moves the camera up along the camera's y-axis.
- void [Down](#) ([Camera](#) &camera, float dt)

Moves the camera down along the camera's y-axis.
- void [RotateCameraLeftRight](#) ([Camera](#) &camera, float xDiff)

Rotates the camera to look left and right.
- void [RotateCameraUpDown](#) ([Camera](#) &camera, float yDiff)

Rotates the camera to look up and down.
- [Color operator+](#) (const [Color](#) &c1, const [Color](#) &c2)

Returns the result of c1 + c2.

- **Color operator-** (const [Color](#) &c1, const [Color](#) &c2)
Returns the result of $c1 - c2$.
- **Color operator*** (const [Color](#) &c, float k)
*Returns the result of $c * k$.*
- **Color operator*** (float k, const [Color](#) &c)
*Returns the result of $k * c$.*
- **Color operator*** (const [Color](#) &c1, const [Color](#) &c2)
*Returns the result of $c1 * c2$.*
- void **InitializeTime** ([Time](#) &time)
- void **Reset** ([Time](#) &time)
- void **Tick** ([Time](#) &time)
- void **Start** ([Time](#) &time)
- void **Stop** ([Time](#) &time)
- void **SetProperties** ([OrthographicProjection](#) &ortho, float width, float height, float znear, float zfar)
Sets the properties of the [OrthographicProjection](#) object to the specified values.
- void **UpdateProjectionMatrix** ([OrthographicProjection](#) &ortho)
Updates the perspective projection matrix of the [PerspectiveProjection](#) object.
- void **SetProperties** ([PerspectiveProjection](#) &perspective, float znear, float zfar, float vFov, float aspectRatio)
Sets the properties of the [PerspectiveProjection](#) object to the specified values.
- void **UpdateProjectionMatrix** ([PerspectiveProjection](#) &perspective)
Updates the perspective projection matrix of the [PerspectiveProjection](#) object.
- void **CreateParentWindow** ([Window](#) &window, const HINSTANCE &hInstance, WNDPROC windowProc, const [RenderingEngine::Color](#) &backgroundColor, const std::wstring &windowClassName, const std::wstring &windowName, unsigned int styles, unsigned int x, unsigned int y, unsigned int width, unsigned int height, void *additionalData=nullptr)
Creates a parent window.
- void **CreateChildWindow** ([Window](#) &window, const HINSTANCE &hInstance, HWND parent, unsigned long long int identifier, WNDPROC windowProcedure, const [RenderingEngine::Color](#) &backgroundColor, const std::wstring &windowClassName, const std::wstring &windowName, unsigned int styles, unsigned int x, unsigned int y, unsigned int width, unsigned int height, void *additionalData=nullptr)
Creates a non-control child window.
- void **CreateControlWindow** ([Window](#) &window, const HINSTANCE &hInstance, HWND parent, unsigned long long int identifier, const std::wstring &windowClassName, const std::wstring &windowName, unsigned int styles, unsigned int x, unsigned int y, unsigned int width, unsigned int height, void *additionalData=nullptr)
Creates a control window.
- unsigned int **GetWidth** (const [Window](#) &window)
- unsigned int **GetHeight** (const [Window](#) &window)
- unsigned int **GetX** (const [Window](#) &window)
- unsigned int **GetY** (const [Window](#) &window)

4.1.1 Detailed Description

Has classes that are used for rendering objects and text through the Direct3D 12 API.

4.1.2 Function Documentation

4.1.2.1 Backward()

```
void RenderingEngine::Backward (
    Camera & camera,
    float dt )
```

Moves the camera backward along the camera's z-axis.

Parameters

in	<i>camera</i>	The camera object.
in	<i>dt</i>	The time between frames.

4.1.2.2 CreateChildWindow()

```
void RenderingEngine::CreateChildWindow (
    Window & window,
    const HINSTANCE & hInstance,
    HWND parent,
    unsigned long long int identifier,
    WNDPROC windowProcedure,
    const RenderingEngine::Color & backgroundColor,
    const std::wstring & windowClassName,
    const std::wstring & windowName,
    unsigned int styles,
    unsigned int x,
    unsigned int y,
    unsigned int width,
    unsigned int height,
    void * additionalData = nullptr )
```

Creates a non-control child window.

Parameters

in	<i>window</i>	A Window object.
in	<i>hInstance</i>	The handle to a module used to identify the executable.
in	<i>parent</i>	A handle to a parent window.
in	<i>identifier</i>	An unsigned integer to identify the child window.
in	<i>windowProcedure</i>	The window procedure that is called when an event occurs.
in	<i>backgroundColor</i>	The background color the window.
in	<i>windowClassName</i>	The name of the window class.
in	<i>windowName</i>	The name of the window.
in	<i>styles</i>	The style of the window. OR together the styles at https://learn.microsoft.com/en-us/windows/win32/winmsg/window-styles
in	<i>The</i>	x position of the top left corner of the window from the parent window top left corner.
in	<i>The</i>	y position of the top left corner of the window from the parent window top left corner..
in	<i>width</i>	The width of the client area of the window.
in	<i>height</i>	The height of the client area of the window.
	<i>[in,optional]</i>	additionalData A pointer to data to access in the window procedure.

4.1.2.3 CreateControlWindow()

```
void RenderingEngine::CreateControlWindow (
    Window & window,
    const HINSTANCE & hInstance,
    HWND parent,
    unsigned long long int identifier,
    const std::wstring & windowClassName,
    const std::wstring & windowName,
    unsigned int styles,
    unsigned int x,
    unsigned int y,
    unsigned int width,
    unsigned int height,
    void * additionalData = nullptr )
```

Creates a control window.

Parameters

in	<i>window</i>	A Window object.
in	<i>hInstance</i>	The handle to a module used to identify the executable.
in	<i>parent</i>	A handle to a parent window.
in	<i>identifier</i>	An unsigned integer to identify the child window.
in	<i>windowClass</i>	The name of the window class.
in	<i>windowName</i>	The name of the window.
in	<i>styles</i>	The style of the window. OR together the styles at https://learn.microsoft.com/en-us/windows/win32/winmsg/window-styles
in	<i>The</i>	x position of the top left corner of the window from the parent window top left corner.
in	<i>The</i>	y position of the top left corner of the window from the parent window top left corner.
in	<i>width</i>	The width of the client area of the window.
in	<i>height</i>	The height of the client area of the window.
	<i>[in,optional]</i>	additionalData A pointer to data to access in the window procedure.

4.1.2.4 CreateParentWindow()

```
void RenderingEngine::CreateParentWindow (
    Window & window,
    const HINSTANCE & hInstance,
    WNDPROC windowProcedure,
    const RenderingEngine::Color & backgroundColor,
    const std::wstring & windowClassName,
    const std::wstring & windowName,
    unsigned int styles,
    unsigned int x,
    unsigned int y,
    unsigned int width,
    unsigned int height,
    void * additionalData = nullptr )
```

Creates a parent window.

The window gets displayed after it is created.

Parameters

in	<i>window</i>	A Window object.
in	<i>hInstance</i>	The handle to a module used to identify the executable.
in	<i>windowProcedure</i>	The window procedure that is called when an event occurs.
in	<i>backgroundColor</i>	The background color the window.
in	<i>windowClassName</i>	The name of the window class.
in	<i>windowName</i>	The name of the window.
in	<i>styles</i>	The style of the window. OR together the styles at https://learn.microsoft.com/en-us/windows/win32/winmsg/window-styles
in	<i>The</i>	x position of the top left corner of the window from the desktops top left corner. Use CW_USEDEFAULT to let system select a default position for you.
in	<i>The</i>	y position of the top left corner of the window from the desktops top left corner. Use CW_USEDEFAULT to let system select a default position for you.
in	<i>width</i>	The width of the client area of the window.
in	<i>height</i>	The height of the client area of the window.
	<i>[in,optional]</i>	additionalData A pointer to data to access in the window procedure.

4.1.2.5 Down()

```
void RenderingEngine::Down (
    Camera & camera,
    float dt )
```

Moves the camera down along the camera's y-axis.

Parameters

in	<i>camera</i>	The camera object.
in	<i>dt</i>	The time between frames.

4.1.2.6 Foward()

```
void RenderingEngine::Foward (
    Camera & camera,
    float dt )
```

Moves the camera forward along the camera's z-axis.

Parameters

in	<i>camera</i>	The camera object.
in	<i>dt</i>	The time between frames.

4.1.2.7 GetHeight()

```
unsigned int RenderingEngine::GetHeight (
    const Window & window )
```

brief Returns the height of the client area of the specified window.

4.1.2.8 GetWidth()

```
unsigned int RenderingEngine::GetWidth (
    const Window & window )
```

brief Returns the width of the client area of the specified window.

4.1.2.9 GetX()

```
unsigned int RenderingEngine::GetX (
    const Window & window )
```

brief Returns the x location of the specified window.

4.1.2.10 GetY()

```
unsigned int RenderingEngine::GetY (
    const Window & window )
```

brief Returns the y location of the specified window.

4.1.2.11 InitializeTime()

```
void RenderingEngine::InitializeTime (
    Time & time )
```

brief Initializes the specified [Time](#) object.

4.1.2.12 Left()

```
void RenderingEngine::Left (
    Camera & camera,
    float dt )
```

Moves the camera left along the camera's x-axis.

4.1.2.13 LookAt()

```
void RenderingEngine::LookAt (
    Camera & camera,
    vec3 position,
    vec3 target,
    vec3 up )
```

Defines the camera space using UVN.

Parameters

in	<i>camera</i>	The camera object.
in	<i>position</i>	The position of the camera.
in	<i>target</i>	The point the camera is looking at.
in	<i>up</i>	The up direction of the world.

4.1.2.14 operator*() [1/3]

```
Color RenderingEngine::operator* (
    const Color & c,
    float k )
```

Returns the result of $c * k$.

If $|k| < 0.0f$, no multiplication happens and [Color](#) c is returned.
If any of the resultant components are $> 1.0f$, they are set to $1.0f$.

4.1.2.15 operator*() [2/3]

```
Color RenderingEngine::operator* (
    const Color & c1,
    const Color & c2 )
```

Returns the result of $c1 * c2$.

If any of the resultant components are $> 1.0f$, they are set to $1.0f$.

4.1.2.16 operator*() [3/3]

```
Color RenderingEngine::operator* (
    float k,
    const Color & c )
```

Returns the result of $k * c$.

If $k < 0.0f$, no multiplication happens and [Color](#) c is returned.
If any of the resultant components are $> 1.0f$, they are set to $1.0f$.

4.1.2.17 operator+()

```
Color RenderingEngine::operator+ (
    const Color & c1,
    const Color & c2 )
```

Returns the result of $c1 + c2$.

Does component-wise addition. If any of the resultant components are $> 1.0f$, they are set to $1.0f$.

4.1.2.18 operator-()

```
Color RenderingEngine::operator- (
    const Color & c1,
    const Color & c2 )
```

Returns the result of $c1 - c2$.

Does component-wise subtraction. If any of the resultant components are $< 0.0f$, they are set to $0.0f$.

4.1.2.19 Reset()

```
void RenderingEngine::Reset (
    Time & time )
```

brief Resets the specified [Time](#) object.

4.1.2.20 Right()

```
void RenderingEngine::Right (
    Camera & camera,
    float dt )
```

Moves the camera right along the camera's x-axis.

Parameters

in	<i>camera</i>	The camera object.
in	<i>dt</i>	The time between frames.

4.1.2.21 RotateCameraLeftRight()

```
void RenderingEngine::RotateCameraLeftRight (
    Camera & camera,
    float xDiff )
```

Rotates the camera to look left and right.

Parameters

in	<i>camera</i>	The camera object.
in	<i>xDiff</i>	How many degrees to rotate.

4.1.2.22 RotateCameraUpDown()

```
void RenderingEngine::RotateCameraUpDown (
    Camera & camera,
    float yDiff )
```

Rotates the camera to look up and down.

Parameters

in	<i>camera</i>	The camera object.
in	<i>yDiff</i>	How many degrees to rotate.

4.1.2.23 SetProperties() [1/3]

```
void RenderingEngine::SetProperties (
    Camera & camera,
    vec3 position,
    vec3 x,
    vec3 y,
    vec3 z,
    float linearSpeed,
    float angularSpeed )
```

Sets the properties of the camera.

Parameters

in	<i>camera</i>	The camera object.
in	<i>position</i>	The position of the camera.
in	<i>x</i>	The x axis of the local coordinate system of the camera.
in	<i>y</i>	The y axis of the local coordinate system of the camera.
in	<i>z</i>	The z axis of the local coordinate system of the camera.
in	<i>linearVelocity</i>	The translational velocity of the camera.
in	<i>angularVelocity</i>	The angular velocity of the camera.

4.1.2.24 SetProperties() [2/3]

```
void RenderingEngine::SetProperties (
    OrthographicProjection & ortho,
    float width,
    float height,
    float znear,
    float zfar )
```

Sets the properties of the [OrthographicProjection](#) object to the specified values.

Parameters

in	<i>ortho</i>	The OrthographicProjection object.
in	<i>width</i>	The width of the box.
in	<i>height</i>	The height of the box.
in	<i>znear</i>	The z value of the near plane.
in	<i>zfar</i>	The z value of the far plane.

4.1.2.25 SetPropertyes() [3/3]

```
void RenderingEngine::SetProperties (
    PerspectiveProjection & perspective,
    float znear,
    float zfar,
    float vFov,
    float aspectRatio )
```

Sets the properties of the [PerspectiveProjection](#) object to the specified values.

Parameters

in	<i>perspective</i>	The PerspectiveProjection object.
in	<i>znear</i>	The z value of where the near plane of the frustrum intersects the z-axis.
in	<i>zfar</i>	The z value of where the far plane of the frustrum intersects the z-axis.
in	<i>vFov</i>	The vertical field of view of the frustrum.
in	<i>aspectRatio</i>	The aspect ratio of the view plane.

4.1.2.26 Start()

```
void RenderingEngine::Start (
    Time & time )
```

brief Starts the time for the specified [Time](#) object.

4.1.2.27 Stop()

```
void RenderingEngine::Stop (
    Time & time )
```

brief Stops the time for the specified [Time](#) object.

4.1.2.28 Tick()

```
void RenderingEngine::Tick (
    Time & time )
```

Computes the delta time (time between each frame) for the specified [Time](#) object.

4.1.2.29 Up()

```
void RenderingEngine::Up (
    Camera & camera,
    float dt )
```

Moves the camera up along the camera's y-axis.

Parameters

in	<i>camera</i>	The camera object.
in	<i>dt</i>	The time between frames.

4.1.2.30 UpdateProjectionMatrix() [1/2]

```
void RenderingEngine::UpdateProjectionMatrix (
    OrthographicProjection & ortho )
```

Updates the perspective projection matrix of the [PerspectiveProjection](#) object.

Parameters

in	<i>p</i>	The PerspectiveProjection object.
----	----------	---

4.1.2.31 UpdateProjectionMatrix() [2/2]

```
void RenderingEngine::UpdateProjectionMatrix (
    PerspectiveProjection & perspective )
```

Updates the perspective projection matrix of the [PerspectiveProjection](#) object.

Parameters

in	<i>p</i>	The PerspectiveProjection object.
----	----------	---

4.1.2.32 UpdateViewMatrix()

```
void RenderingEngine::UpdateViewMatrix (
    Camera & camera )
```

After modifying the camera position and/or orientation, call this to rebuild the view transformation matrix.

Chapter 5

Class Documentation

5.1 RenderingEngine::Camera Struct Reference

Simple first person style camera class that lets the viewer explore the 3D scene. It keeps track of the camera coordinate system relative to the world space so that the view matrix can be constructed.

```
#include "Camera.h"
```

Public Attributes

- vec3 **position**
- vec3 **x**
- vec3 **y**
- vec3 **z**
- mat4 **viewMatrix**
- float **linearSpeed** = 0.0f
- float **angularSpeed** = 0.0f

5.1.1 Detailed Description

Simple first person style camera class that lets the viewer explore the 3D scene. It keeps track of the camera coordinate system relative to the world space so that the view matrix can be constructed.

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

- Camera.h

5.2 RenderingEngine::Color Class Reference

This class stores a RGBA color in a 4D vector using floats. The range of each component is [0.0, 1.0]. The first componet is red, second component is green, third component is blue and the 4th component is alpha.

```
#include "Color.h"
```

Public Member Functions

- **Color** (float r=0.0f, float g=0.0f, float b=0.0f, float a=1.0f)
Initializes the color to the specified RGBA values.
- **Color** (const vec4 &color)
Initializes the color to the specified color.
- const vec4 & **GetColor** () const
Returns the color.
- float **GetRed** () const
Returns the value of the red component.
- float **GetGreen** () const
Returns the value of the green component.
- float **GetBlue** () const
Returns the value of the blue component.
- float **GetAlpha** () const
Returns the value of the alpha component.
- void **SetColor** (const vec4 &color)
Sets the color to the specified color.
- void **SetRed** (float r)
Sets the red component to the specified float value.
- void **SetGreen** (float g)
Sets the green component to the specified float value.
- void **SetBlue** (float b)
Sets the blue component to the specified float value.
- void **SetAlpha** (float a)
Sets the alpha component to the specified float value.
- **Color** & **operator+=** (const **Color** &c)
Adds this objects color to the specified color c and stores the result in this object.
- **Color** & **operator-=** (const **Color** &c)
Subtracts the specified color c from this objects color and stores the result in this object.
- **Color** & **operator*=** (float k)
Multiplies this objects color by the specified value k and stores the result in this object.
- **Color** & **operator*=** (const **Color** &c)
Multiplies this objects color by the specified color c and stores the result in this object.

5.2.1 Detailed Description

This class stores a RGBA color in a 4D vector using floats. The range of each component is [0.0, 1.0]. The first component is red, second component is green, third component is blue and the 4th component is alpha.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Color() [1/2]

```
RenderingEngine::Color::Color (
    float r = 0.0f,
    float g = 0.0f,
    float b = 0.0f,
    float a = 1.0f )
```

Initializes the color to the specified RGBA values.

5.2.2.2 Color() [2/2]

```
RenderingEngine::Color::Color (
    const vec4 & color )
```

Initializes the color to the specified *color*.

5.2.3 Member Function Documentation

5.2.3.1 GetAlpha()

```
float RenderingEngine::Color::GetAlpha ( ) const
```

Returns the value of the alpha component.

5.2.3.2 GetBlue()

```
float RenderingEngine::Color::GetBlue ( ) const
```

Returns the value of the green component.

5.2.3.3 GetColor()

```
const vec4 & RenderingEngine::Color::GetColor ( ) const
```

Returns the color.

5.2.3.4 GetGreen()

```
float RenderingEngine::Color::GetGreen ( ) const
```

Returns the value of the blue component.

5.2.3.5 GetRed()

```
float RenderingEngine::Color::GetRed ( ) const
```

Returns the value of the red component.

5.2.3.6 operator*=() [1/2]

```
Color & RenderingEngine::Color::operator*= (
    const Color & c )
```

Multiplies this objects color by the specified color *c* and stores the result in this object.

If any of the resultant components are $> 1.0f$, they are set to $1.0f$.
Does component-wise multiplication.

5.2.3.7 operator*=() [2/2]

```
Color & RenderingEngine::Color::operator*= (
    float k )
```

Multiplies this objects color by the specified value *k* and stores the result in this object.

If $k < 0.0f$, no multiplication happens and this objects color does not get modified.
If any of the resultant components are $> 1.0f$, they are set to $1.0f$.

5.2.3.8 operator+=()

```
Color & RenderingEngine::Color::operator+= (
    const Color & c )
```

Adds this objects color to the specified color *c* and stores the result in this object.

Does component-wise addition. If any of the resultant components are $> 1.0f$, they are set to $1.0f$.

5.2.3.9 operator-=()

```
Color & RenderingEngine::Color::operator-= (
    const Color & c )
```

Subtracts the specified color *c* from this objects color and stores the result in this object.

Does component-wise subtraction. If any of the resultant components are $< 0.0f$, they are set to $0.0f$.

5.2.3.10 SetAlpha()

```
void RenderingEngine::Color::SetAlpha (
    float a )
```

Sets the alpha component to the specified float value.

5.2.3.11 SetBlue()

```
void RenderingEngine::Color::SetBlue (
    float b )
```

Sets the blue component to the specified float value.

5.2.3.12 SetColor()

```
void RenderingEngine::Color::SetColor (
    const vec4 & color )
```

Sets the color to the specified color.

5.2.3.13 SetGreen()

```
void RenderingEngine::Color::SetGreen (
    float g )
```

Sets the green component to the specified float value.

5.2.3.14 SetRed()

```
void RenderingEngine::Color::SetRed (
    float r )
```

Sets the red component to the specified float value.

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

- Color.h

5.3 RenderingEngine::DepthStencilBuffer Class Reference

A wrapper for depth stencil buffer resources. Uses DirectD 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

```
#include "Buffer.h"
```

Public Member Functions

- **DepthStencilBuffer** (const [DepthStencilBuffer](#) &)=delete
- **DepthStencilBuffer** & **operator=** (const [DepthStencilBuffer](#) &)=delete
- **DepthStencilBuffer** ()
Creates a depth stencil buffer object. Call the [CreateDepthStencilBufferAndView\(\)](#) to allocate memory for the buffer.
- **DepthStencilBuffer** (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &dsvHeap, unsigned int index, unsigned int dsvSize, unsigned int width, unsigned int height, DXGI_FORMAT format=DXGI_FORMAT_D24_UNORM_S8_UINT, unsigned int sampleCount=1)
Creates the depth stencil buffer and view.
- DXGI_FORMAT **GetDepthStencilFormat** () const
Returns the format of the depth stencil buffer.
- void **CreateDepthStencilBufferAndView** (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &dsvHeap, unsigned int index, unsigned int dsvSize, unsigned int width, unsigned int height, DXGI_FORMAT format=DXGI_FORMAT_D24_UNORM_S8_UINT, unsigned int sampleCount=1)
Creates the depth stencil buffer and view.
- void **ReleaseBuffer** ()
Frees the memory of the buffer.
- void **ClearDepthStencilBuffer** (const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &dsvHeap, unsigned int indexInView, unsigned int dsvSize, float clearValue)
Clears the depth stencil buffer with the specified clear value.

5.3.1 Detailed Description

A wrapper for depth stencil buffer resources. Uses DirectD 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 DepthStencilBuffer() [1/2]

```
RenderingEngine::DepthStencilBuffer::DepthStencilBuffer ( )
```

Creates a depth stencil buffer object. Call the [CreateDepthStencilBufferAndView\(\)](#) to allocate memory for the buffer.

5.3.2.2 DepthStencilBuffer() [2/2]

```
RenderingEngine::DepthStencilBuffer::DepthStencilBuffer (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
    const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & dsvHeap,
    unsigned int index,
    unsigned int dsvSize,
    unsigned int width,
    unsigned int height,
    DXGI_FORMAT format = DXGI_FORMAT_D24_UNORM_S8_UINT,
    unsigned int sampleCount = 1 )
```

Creates the depth stencil buffer and view.

Parameters

in	<i>device</i>	A Direct3D 12 device.
in	<i>dsvHeap</i>	A descriptor heap for storing depth stencil descriptors.
in	<i>indexOfWhereToStoreView</i>	The index of where to store the created descriptor in the descriptor heap.
in	<i>dsvSize</i>	The size of a depth stencil descriptor.
in	<i>width</i>	The width of the depth stencil buffer.
in	<i>height</i>	The height of the depth stencil buffer.
in	<i>sampleCount</i>	The sample count of the depth stencil buffer.

5.3.3 Member Function Documentation

5.3.3.1 ClearDepthStencilBuffer()

```
void RenderingEngine::DepthStencilBuffer::ClearDepthStencilBuffer (
    const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & commandList,
    const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & dsvHeap,
    unsigned int indexOfView,
    unsigned int dsvSize,
    float clearValue )
```

Clears the depth stencil buffer with the specified clear value.

Parameters

in	<i>commadList</i>	A Direct3D 12 graphics command list.
in	<i>dsvHeap</i>	A depth stencil descriptor heap.
in	<i>indexOfView</i>	The index of where the depth stencil descriptor of the depth stencil buffer is stored in the descriptor heap.
in	<i>dsvSize</i>	The size of a depth stencil descriptor.
in	<i>clearValue</i>	The value of what to set every element in the depth stencil buffer to.

5.3.3.2 CreateDepthStencilBufferAndView()

```
void RenderingEngine::DepthStencilBuffer::CreateDepthStencilBufferAndView (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
    const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & dsvHeap,
    unsigned int index,
    unsigned int dsvSize,
    unsigned int width,
    unsigned int height,
    DXGI_FORMAT format = DXGI_FORMAT_D24_UNORM_S8_UINT,
    unsigned int sampleCount = 1 )
```

Creates the depth stencil buffer and view.

Parameters

in	<i>device</i>	A Direct3D 12 device.
in	<i>dsvHeap</i>	A descriptor heap for storing depth stencil descriptors.
in	<i>indexOfWhereToStoreView</i>	The index of where to store the created descriptor in the descriptor heap.
in	<i>dsvSize</i>	The size of a depth stenicl descriptor.
in	<i>width</i>	The width of the depth stenicl buffer.
in	<i>height</i>	The height of the depth stenicl buffer.
in	<i>sampleCount</i>	The sample count of the depth stenicl buffer.

5.3.3.3 GetDepthStencilFormat()

```
DXGI_FORMAT RenderingEngine::DepthStencilBuffer::GetDepthStencilFormat ( ) const
```

Returns the format of the depth stencil buffer.

5.3.3.4 ReleaseBuffer()

```
void RenderingEngine::DepthStencilBuffer::ReleaseBuffer ( )
```

Frees the memory of the buffer.

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

- Buffer.h

5.4 RenderingEngine::DeviceResources Class Reference

A wrapper for resources that are needed to render objects and text using the Direct3D 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

```
#include "DeviceResources.h"
```

Public Member Functions

- **DeviceResources** (const [DeviceResources](#) &)=delete
- [DeviceResources](#) & **operator=** (const [DeviceResources](#) &)=delete
- [~DeviceResources](#) ()
Flushes the command queue.
- const Microsoft::WRL::ComPtr< ID3D12Device > & [GetDevice](#) () const
Returns a constant reference to the ID3D12Device object.
- const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & [GetCommandList](#) () const
Returns a constant reference to the ID3D12GraphicsCommandList object.
- DXGI_FORMAT [GetBackBufferFormat](#) () const
Returns a constant reference to the back buffer format.
- DXGI_FORMAT [GetDepthStencilFormat](#) () const
Returns a constant reference to the depth stencil format.
- unsigned int [GetCBVSRVUAVSize](#) () const
The size of a constant buffer/shader resource/unordered access view.
- unsigned int [GetCurrentFrame](#) () const
Returns the current frame.
- const [TextResources](#) & [GetTextResources](#) () const
Returns a constant reference to the [TextResources](#) object.
- void [UpdateCurrentFrameFenceValue](#) ()
Updates the current frames fence value.
- void [FlushCommandQueue](#) ()
Synchronizes the CPU and GPU.
- void [WaitForGPU](#) () const
Waits for the GPU to execute all of the commands of the current frame.
- void [Signal](#) ()
Adds an instruction to the GPU to set the fence value to the current fence value.
- void [Resize](#) (int width, int height, const HWND &handle, bool isMSAAEnabled, bool isTextEnabled)
Call when the window gets resized.
- void [RTBufferTransition](#) (bool isMSAAEnabled, bool isTextEnabled)
Transitions the render target buffer.
- void [BeforeTextDraw](#) ()
Prepares to render text.
- void [AfterTextDraw](#) ()
Executes the text commands.
- void [Execute](#) () const
Executes the command list.
- void [Present](#) ()
Swaps the front and back buffers.
- void **Draw** (bool isMSAAEnabled)
- void [NextFrame](#) ()
Updates the current frame value to go to the next frame.

Static Public Member Functions

- static [DeviceResources](#) & [GetInstance](#) (unsigned int width, unsigned int height, HWND windowHandle, bool isMSAAEnabled, bool isTextEnabled)

Call to make an object of [DeviceResources](#).

Static Public Attributes

- static const unsigned int [NUM_OF_FRAMES](#) { 3 }

The number of frames in the circular array.

5.4.1 Detailed Description

A wrapper for resources that are needed to render objects and text using the Direct3D 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 ~DeviceResources()

```
RenderingEngine::DeviceResources::~~DeviceResources ( )
```

Flushes the command queue.

5.4.3 Member Function Documentation

5.4.3.1 AfterTextDraw()

```
void RenderingEngine::DeviceResources::AfterTextDraw ( )
```

Executes the text commands.

5.4.3.2 BeforeTextDraw()

```
void RenderingEngine::DeviceResources::BeforeTextDraw ( )
```

Prepares to render text.

5.4.3.3 Execute()

```
void RenderingEngine::DeviceResources::Execute ( ) const
```

Executes the command list.

5.4.3.4 FlushCommandQueue()

```
void RenderingEngine::DeviceResources::FlushCommandQueue ( )
```

Synchronizes the CPU and GPU.

Use this function to make sure all of the commands in command list are executed by the GPU before the CPU writes in new commands.

5.4.3.5 GetBackBufferFormat()

```
DXGI_FORMAT RenderingEngine::DeviceResources::GetBackBufferFormat ( ) const
```

Returns a constant reference to the back buffer format.

5.4.3.6 GetCBVSRVUAVSize()

```
unsigned int RenderingEngine::DeviceResources::GetCBVSRVUAVSize ( ) const
```

The size of a constant buffer/shader resource/unordered access view.

5.4.3.7 GetCommandList()

```
const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & RenderingEngine::DeviceResources↵  
::GetCommandList ( ) const
```

Returns a constant reference to the ID3D12GraphicsCommandList object.

5.4.3.8 GetCurrentFrame()

```
unsigned int RenderingEngine::DeviceResources::GetCurrentFrame ( ) const
```

Returns the current frame.

5.4.3.9 GetDepthStencilFormat()

```
DXGI_FORMAT RenderingEngine::DeviceResources::GetDepthStencilFormat ( ) const
```

Returns a constant reference to the depth stencil format.

5.4.3.10 GetDevice()

```
const Microsoft::WRL::ComPtr< ID3D12Device > & RenderingEngine::DeviceResources::GetDevice ( )
const
```

Returns a constant reference to the ID3D12Device object.

5.4.3.11 GetInstance()

```
static DeviceResources & RenderingEngine::DeviceResources::GetInstance (
    unsigned int width,
    unsigned int height,
    HWND windowHandle,
    bool isMSAAEnabled,
    bool isTextEnabled ) [static]
```

Call to make an object of [DeviceResources](#).

Only one instance of [DeviceResources](#) can exist in a program.

Parameters

in	<i>width</i>	The width of a window.
in	<i>height</i>	The height of a window.
in	<i>windowHandle</i>	A handle to a window.
in	<i>isMSAAEnabled</i>	Pass in true if you want to have MSAA enabled for the initial frame, false otherwise.
in	<i>isTextEnabled</i>	Pass in true if you want to have text enabled for the initial frame, false otherwise.

5.4.3.12 GetTextResources()

```
const TextResources & RenderingEngine::DeviceResources::GetTextResources ( ) const
```

Returns a constant reference to the [TextResources](#) object.

5.4.3.13 NextFrame()

```
void RenderingEngine::DeviceResources::NextFrame ( )
```

Updates the current frame value to go to the next frame.

5.4.3.14 Present()

```
void RenderingEngine::DeviceResources::Present ( )
```

Swaps the front and back buffers.

5.4.3.15 Resize()

```
void RenderingEngine::DeviceResources::Resize (
    int width,
    int height,
    const HWND & handle,
    bool isMSAAEnabled,
    bool isTextEnabled )
```

Call when the window gets resized.

Call when you initialize your program.

Parameters

in	<i>width</i>	The width of a window.
in	<i>height</i>	The height of a window.
in	<i>handle</i>	A handle to a window.
in	<i>isMSAAEnabled</i>	Pass in true if MSAA enabled, false otherwise.
in	<i>isTextEnabled</i>	Pass in true if text enabled, false otherwise.

5.4.3.16 RTBufferTransition()

```
void RenderingEngine::DeviceResources::RTBufferTransition (
    bool isMSAAEnabled,
    bool isTextEnabled )
```

Transitions the render target buffer.

Parameters

in	<i>isMSAAEnabled</i>	Pass in true if MSAA enabled, false otherwise.
in	<i>isTextEnabled</i>	Pass in true if text enabled, false otherwise.

5.4.3.17 Signal()

```
void RenderingEngine::DeviceResources::Signal ( )
```

Adds an instruction to the GPU to set the fence value to the current fence value.

5.4.3.18 UpdateCurrentFrameFenceValue()

```
void RenderingEngine::DeviceResources::UpdateCurrentFrameFenceValue ( )
```

Updates the current frames fence value.

5.4.3.19 WaitForGPU()

```
void RenderingEngine::DeviceResources::WaitForGPU ( ) const
```

Waits for the GPU to execute all of the commands of the current frame.

Signal should have been called before this function is called.

5.4.4 Member Data Documentation

5.4.4.1 NUM_OF_FRAMES

```
const unsigned int RenderingEngine::DeviceResources::NUM_OF_FRAMES { 3 } [static]
```

The number of frames in the ciruclar array.

Allows the CPU to produce the commands for future frames as the GPU is executing the commands for the current frame.

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

- DeviceResources.h

5.5 DirectXException Class Reference

A class for handling Direct3D and DXGI errors from functions that return a HRESULT value.

```
#include "DirectXException.h"
```

Public Member Functions

- [DirectXException](#) (HRESULT hr, const std::wstring &functionName, const std::wstring &fileName, int lineNumber)
Constructs a [DirectXException](#) object.
- std::wstring [ErrorMsg](#) () const
Returns a message describing the error.

5.5.1 Detailed Description

A class for handling Direct3D and DXGI errors from functions that return a HRESULT value.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 DirectXException()

```
DirectXException::DirectXException (
    HRESULT hr,
    const std::wstring & functionName,
    const std::wstring & fileName,
    int lineNumber )
```

Constructs a [DirectXException](#) object.

Parameters

in	<i>hr</i>	The HRESULT value of a function.
in	<i>functionName</i>	The name of the function.
in	<i>fileName</i>	The name of the file where the function was called.
in	<i>lineNumber</i>	The line number of the function call.

5.5.3 Member Function Documentation

5.5.3.1 ErrorMsg()

```
std::wstring DirectXException::ErrorMsg ( ) const
```

Returns a message describing the error.

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

- DirectXException.h

5.6 RenderingEngine::DrawArguments Struct Reference

Data that are used as parameters to draw an object.

```
#include "DrawArguments.h"
```

Public Attributes

- unsigned int **indexCount** = 0
- unsigned int **locationOfFirstIndex** = 0
- int **indexOfFirstVertex** = 0
- unsigned int **indexOfConstantData** = 0
- unsigned int **rootParameterIndex** = 0
- std::wstring **constantBufferKey** = L""
- D3D_PRIMITIVE_TOPOLOGY **primitive** = D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST

5.6.1 Detailed Description

Data that are used as parameters to draw an object.

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

- DrawArguments.h

5.7 RenderingEngine::DynamicBuffer Class Reference

This class stores data in a Direct3D 12 upload buffer. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

```
#include "Buffer.h"
```

Public Member Functions

- **DynamicBuffer** (const [DynamicBuffer](#) &)=delete
- **DynamicBuffer** & **operator=** (const [DynamicBuffer](#) &)=delete
- **DynamicBuffer** ()
Creates a dynamic buffer object. No memory is allocated for the buffer. Call one of the [CreateDynamicBuffer\(\)](#) functions to allocate memory for the buffer.
- **DynamicBuffer** (const Microsoft::WRL::ComPtr< ID3D12Device > &device, unsigned int numOfBytes, unsigned int stride)
Creates and maps a dynamic vertex buffer or a dynamic constant buffer.
- **DynamicBuffer** (const Microsoft::WRL::ComPtr< ID3D12Device > &device, unsigned int numOfBytes, DXGI_FORMAT format)
Creates and maps a dynamic index buffer.
- **~DynamicBuffer** ()
Unmaps the pointer to the dynamic buffer.
- void **CreateDynamicBuffer** (const Microsoft::WRL::ComPtr< ID3D12Device > &device, unsigned int num↵OfBytes, unsigned int stride)

Creates and maps a dynamic vertex buffer or a dynamic constant buffer.

- void [CreateDynamicBuffer](#) (const Microsoft::WRL::ComPtr< ID3D12Device > &device, unsigned int num↵OfBytes, DXGI_FORMAT format)

Creates and maps a dynamic index buffer.

- const D3D12_GPU_VIRTUAL_ADDRESS [GetGPUAddress](#) (unsigned int index) const

Returns the GPU address of the data at the specified index.

- void [CreateConstantBufferView](#) (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &cbvHeap, unsigned int cbvSize, unsigned int cbv↵HeapIndex, unsigned int cBufferIndex)

Creates the constant buffer view and stores it in the specified descriptor heap.

- const D3D12_VERTEX_BUFFER_VIEW [GetVertexBufferView](#) ()

Returns a the vertex buffer view of the dynamic buffer.

- const D3D12_INDEX_BUFFER_VIEW [GetIndexBufferView](#) ()

Returns the index buffer view of the dynamic buffer.

- void [CopyData](#) (unsigned int index, const void *data, unsigned long long numOfBytes)

Copies data from the given data into the dynamic buffer. Uses 0-indexing.

- void [ReleaseBuffer](#) ()

Frees the dynamic buffer memory.

5.7.1 Detailed Description

This class stores data in a Direct3D 12 upload buffer. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 DynamicBuffer() [1/3]

```
RenderingEngine::DynamicBuffer::DynamicBuffer ( )
```

Creates a dynamic buffer object. No memory is allocated for the buffer. Call one of the [CreateDynamicBuffer\(\)](#) functions to allocate memory for the buffer.

5.7.2.2 DynamicBuffer() [2/3]

```
RenderingEngine::DynamicBuffer::DynamicBuffer (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
    unsigned int numOfBytes,
    unsigned int stride )
```

Creates and maps a dynamic vertex buffer or a dynamic constant buffer.

Parameters

in	<i>device</i>	A Direct3D 12 device.
in	<i>numOfBytes</i>	The number of bytes you want to allocate for the dynamic buffer.
in	<i>stride</i>	The number of bytes to get from one element to another in the dynamic buffer.

5.7.2.3 DynamicBuffer() [3/3]

```

RenderingEngine::DynamicBuffer::DynamicBuffer (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
    unsigned int numOfBytes,
    DXGI_FORMAT format )

```

Creates and maps a dynamic index buffer.

Parameters

in	<i>device</i>	A Direct3D 12 device.
in	<i>numOfBytes</i>	The number of bytes you want to allocate for the dynamic buffer.
in	<i>format</i>	The number of bytes to get from one element to another in the dynamic buffer.

5.7.2.4 ~DynamicBuffer()

```

RenderingEngine::DynamicBuffer::~DynamicBuffer ( )

```

Unmaps the pointer to the dynamic buffer.

5.7.3 Member Function Documentation**5.7.3.1 CopyData()**

```

void RenderingEngine::DynamicBuffer::CopyData (
    unsigned int index,
    const void * data,
    unsigned long long numOfBytes )

```

Copies data from the given data into the dynamic buffer. Uses 0-indexing.

Parameters

in	<i>data</i>	The data to copy in the dynamic buffer.
in	<i>numOfBytes</i>	The number of bytes to copy.

5.7.3.2 CreateConstantBufferView()

```
void RenderingEngine::DynamicBuffer::CreateConstantBufferView (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
    const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & cbvHeap,
    unsigned int cbvSize,
    unsigned int cbvHeapIndex,
    unsigned int cBufferIndex )
```

Creates the constant buffer view and stores it in the specified descriptor heap.

Parameters

in	<i>device</i>	A Direct3D 12 device.
in	<i>cbvHeap</i>	A descriptor heap for storing constant buffer descriptors.
in	<i>cbvSize</i>	The size of a depth stencil descriptor.
in	<i>cbvHeapIndex</i>	The index of where to store the created descriptor in the descriptor heap.
in	<i>cBufferIndex</i>	The index of the constant data in the constant buffer you want to describe.

5.7.3.3 CreateDynamicBuffer() [1/2]

```
void RenderingEngine::DynamicBuffer::CreateDynamicBuffer (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
    unsigned int numOfBytes,
    DXGI_FORMAT format )
```

Creates and maps a dynamic index buffer.

Parameters

in	<i>device</i>	A Direct3D 12 device.
in	<i>numOfBytes</i>	The number of bytes you want to allocate for the dynamic buffer.
in	<i>format</i>	The number of bytes to get from one element to another in the dynamic buffer.

5.7.3.4 CreateDynamicBuffer() [2/2]

```
void RenderingEngine::DynamicBuffer::CreateDynamicBuffer (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
    unsigned int numOfBytes,
    unsigned int stride )
```

Creates and maps a dynamic vertex buffer or a dynamic constant buffer.

Parameters

in	<i>device</i>	A Direct3D 12 device.
in	<i>numOfBytes</i>	The number of bytes you want to allocate for the dynamic buffer.
in	<i>stride</i>	The number of bytes to get from one element to another in the dynamic buffer.

5.7.3.5 GetGPUAddress()

```
const D3D12_GPU_VIRTUAL_ADDRESS RenderingEngine::DynamicBuffer::GetGPUAddress (
    unsigned int index ) const
```

Returns the GPU address of the data at the specified index.

5.7.3.6 GetIndexBufferView()

```
const D3D12_INDEX_BUFFER_VIEW RenderingEngine::DynamicBuffer::GetIndexBufferView ( )
```

Returns the index buffer view of the dynamic buffer.

5.7.3.7 GetVertexBufferView()

```
const D3D12_VERTEX_BUFFER_VIEW RenderingEngine::DynamicBuffer::GetVertexBufferView ( )
```

Returns a the vertex buffer view of the dynamic buffer.

5.7.3.8 ReleaseBuffer()

```
void RenderingEngine::DynamicBuffer::ReleaseBuffer ( )
```

Frees the dynamic buffer memory.

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

- Buffer.h

5.8 RenderingEngine::MultiSampling Class Reference

A wrapper for multisampling resources. Uses DirectD 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

```
#include "Multisampling.h"
```

Public Member Functions

- **MultiSampling** (const [MultiSampling](#) &)=delete
- **MultiSampling** & **operator=** (const [MultiSampling](#) &)=delete
- **MultiSampling** ()
Creates a multisampling object. Call the function [CheckMultiSamplingSupport\(\)](#) to check if the desired formats and sample count is supported by a GPU. If they are supported, a multisampling render target buffer and a multisampling depth stencil buffer can be created.
- **MultiSampling** (const Microsoft::WRL::ComPtr< ID3D12Device > &device, DXGI_FORMAT rtFormat, unsigned int sampleCount)
Checks if the specified format and sample count are supported by the specified device for multi-sampling.
- void **CheckMultiSamplingSupport** (const Microsoft::WRL::ComPtr< ID3D12Device > &device, DXGI_FORMAT rtFormat, unsigned int sampleCount)
Checks if the specified format and sample count are supported by the specified device for multi-sampling.
- const Microsoft::WRL::ComPtr< ID3D12Resource > & **GetRenderTargetBuffer** ()
Returns a constant reference to the MSAA render target buffer.
- DXGI_FORMAT **GetRenderTargetFormat** ()
Returns the format of the MSAA render target buffer.
- DXGI_FORMAT **GetDepthStencilFormat** ()
Returns the format of the MSAA depth stencil buffer.
- void **ReleaseBuffers** ()
Resets the MSAA render target buffer and MSAA depth stencil buffer.
- void **CreateRenderTargetBufferAndView** (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &rtvHeap, unsigned int indexWhereToStoreView, unsigned int rtvSize, unsigned int width, unsigned int height)
Creates the MSAA render target buffer and a view to it.
- void **CreateDepthStencilBufferAndView** (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &dsvHeap, unsigned int indexWhereToStoreView, unsigned int dsvSize, unsigned int width, unsigned int height)
Creates the MSAA depth stencil buffer and a view to it.
- void **Transition** (const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, D3D12_RESOURCE_STATES before, D3D12_RESOURCE_STATES after)
Transitions the MSAA render target buffer from the specified before state to the specified after state.
- void **ClearRenderTargetBuffer** (const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &rtvHeap, unsigned int indexOfView, unsigned int rtvSize, const float *clearValue)
Clears the MSAA render target buffer with the specified clear value.
- void **ClearDepthStencilBuffer** (const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &dsvHeap, unsigned int indexOfView, unsigned int dsvSize, float clearValue)
Clears the MSAA depth stencil buffer with the specified clear value.

5.8.1 Detailed Description

A wrapper for multisampling resources. Uses DirectD 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 MultiSampling() [1/2]

```
RenderingEngine::MultiSampling::MultiSampling ( )
```

Creates a multisampling object. Call the function [CheckMultiSamplingSupport\(\)](#) to check if the desired formats and sample count is supported by a GPU. If they are supported, a multisampling render target buffer and a multisampling depth stencil buffer can be created.

5.8.2.2 MultiSampling() [2/2]

```
RenderingEngine::MultiSampling::MultiSampling (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
    DXGI_FORMAT rtFormat,
    unsigned int sampleCount )
```

Checks if the specified format and sample count are supported by the specified device for multi-sampling.

Throws a runtime_error if they are not supported.

Parameters

in	<i>device</i>	A Direct3D 12 device.
in	<i>rtFormat</i>	The format of the render target buffer.
in	<i>dsFormat</i>	The format of the depth stencil buffer.
in	<i>sampleCount</i>	The number of samples for the multi-sampling render target and depth stencil buffers.

5.8.3 Member Function Documentation

5.8.3.1 CheckMultiSamplingSupport()

```
void RenderingEngine::MultiSampling::CheckMultiSamplingSupport (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
    DXGI_FORMAT rtFormat,
    unsigned int sampleCount )
```

Checks if the specified format and sample count are supported by the specified device for multi-sampling.

Throws a runtime_error if they are not supported.

Parameters

in	<i>device</i>	A Direct3D 12 device.
in	<i>rtFormat</i>	The format of the render target buffer.
in	<i>dsFormat</i>	The format of the depth stencil buffer.
in	<i>sampleCount</i>	The number of samples for the multi-sampling render target and depth stencil buffers.

5.8.3.2 ClearDepthStencilBuffer()

```
void RenderingEngine::MultiSampling::ClearDepthStencilBuffer (
    const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & commandList,
    const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & dsvHeap,
    unsigned int indexOfView,
    unsigned int dsvSize,
    float clearValue )
```

Clears the MSAA depth stencil buffer with the specified clear value.

Parameters

in	<i>commadList</i>	A Direct3D 12 graphics command list.
in	<i>dsvHeap</i>	A depth stencil descriptor heap.
in	<i>indexOfView</i>	The index of where the depth stencil descriptor of the depth stencil buffer is stored in the descriptor heap.
in	<i>dsvSize</i>	The size of a depth stencil descriptor.
in	<i>clearValue</i>	The value of what to set every element in the depth stencil buffer to.

5.8.3.3 ClearRenderTargetBuffer()

```
void RenderingEngine::MultiSampling::ClearRenderTargetBuffer (
    const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & commandList,
    const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & rtvHeap,
    unsigned int indexOfView,
    unsigned int rtvSize,
    const float * clearValue )
```

Clears the MSAA render target buffer with the specified clear value.

Parameters

in	<i>commadList</i>	A Direct3D 12 graphics command list.
in	<i>rtvHeap</i>	A render target descriptor heap.
in	<i>indexOfView</i>	The index of where the render target descriptor of the render target buffer is stored in the descriptor heap.
in	<i>rtvSize</i>	The size of a render target descriptor.
in	<i>clearValue</i>	The RGBA values of what to set every element in the render target buffer to.

5.8.3.4 CreateDepthStencilBufferAndView()

```
void RenderingEngine::MultiSampling::CreateDepthStencilBufferAndView (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
```

```

const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & dsvHeap,
unsigned int indexWhereToStoreView,
unsigned int dsvSize,
unsigned int width,
unsigned int height )

```

Creates the MSAA depth stencil buffer and a view to it.

Parameters

in	<i>device</i>	A Direct3D 12 device.
in	<i>dsvHeap</i>	A descriptor heap for storing depth stencil descriptors.
in	<i>indexWhereToStoreView</i>	The index of where to store the created descriptor in the descriptor heap.
in	<i>dsvSize</i>	The size of a depth stencil descriptor.
in	<i>width</i>	The width of the depth stencil buffer.
in	<i>height</i>	The height of the depth stencil buffer.

5.8.3.5 CreateRenderTargetBufferAndView()

```

void RenderingEngine::MultiSampling::CreateRenderTargetBufferAndView (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
    const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & rtvHeap,
    unsigned int indexWhereToStoreView,
    unsigned int rtvSize,
    unsigned int width,
    unsigned int height )

```

Creates the MSAA render target buffer and a view to it.

Parameters

in	<i>device</i>	A Direct3D 12 device.
in	<i>rtvHeap</i>	A descriptor heap for storing render target descriptors.
in	<i>indexWhereToStoreView</i>	The index of where to store the created descriptor in the descriptor heap.
in	<i>rtvSize</i>	The size of a render target descriptor.
in	<i>width</i>	The width of the render target buffer.
in	<i>height</i>	The height of the render target buffer.

5.8.3.6 GetDepthStencilFormat()

```

DXGI_FORMAT RenderingEngine::MultiSampling::GetDepthStencilFormat ( )

```

Returns the format of the MSAA depth stencil buffer.

5.8.3.7 GetRenderTargetBuffer()

```
const Microsoft::WRL::ComPtr< ID3D12Resource > & RenderingEngine::MultiSampling::GetRenderTargetBuffer ( )
```

Returns a constant reference to the MSAA render target buffer.

5.8.3.8 GetRenderTargetFormat()

```
DXGI_FORMAT RenderingEngine::MultiSampling::GetRenderTargetFormat ( )
```

Returns the format of the MSAA render target buffer.

5.8.3.9 ReleaseBuffers()

```
void RenderingEngine::MultiSampling::ReleaseBuffers ( )
```

Resets the MSAA render target buffer and MSAA depth stencil buffer.

5.8.3.10 Transition()

```
void RenderingEngine::MultiSampling::Transition (
    const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & commandList,
    D3D12_RESOURCE_STATES before,
    D3D12_RESOURCE_STATES after )
```

Transitions the MSAA render target buffer from the specified *before* state to the specified *after* state.

Parameters

in	<i>commandList</i>	A Direct3D 12 graphics command list.
----	--------------------	--------------------------------------

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

- Multisampling.h

5.9 RenderingEngine::OrthographicProjection Struct Reference

A struct that holds the properties for doing orthographics projection.

```
#include "OrthographicProjection.h"
```

Public Attributes

- float **width** = 0.0f
- float **height** = 0.0f
- float **znear** = 0.0f
- float **zfar** = 0.0f
- mat4 **projectionMatrix**

5.9.1 Detailed Description

A struct that holds the properties for doing orthographics projection.

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

- OrthographicProjection.h

5.10 RenderingEngine::PerspectiveProjection Struct Reference

A struct that holds the properties for doing perspective projection.

```
#include "PerspectiveProjection.h"
```

Public Attributes

- float **znear** = 0.0f
- float **zfar** = 0.0f
- float **verticalFov** = 0.0f
- float **aspectRatio** = 0.0f
- mat4 **projectionMatrix**

5.10.1 Detailed Description

A struct that holds the properties for doing perspective projection.

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

- PerspectiveProjection.h

5.11 RenderingEngine::RenderScene Class Reference

This class is used to render a scene using Direct3D 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

```
#include "RenderScene.h"
```


Public Member Functions

- **RenderScene** (const [RenderScene](#) &)=delete
- **RenderScene operator=** (const [RenderScene](#) &)=delete
- **RenderScene** (unsigned int width, unsigned int height, HWND windowHandle, bool isMSAAEnabled=false, bool isTextEnabled=false)
- void **CreateDeviceResources** (unsigned int width, unsigned int height, HWND windowHandle, bool isMSAAEnabled=false, bool isTextEnabled=false)
- void **LoadShader** (unsigned int shaderKey, std::wstring_view filename)
Loads a shaders bytecode and maps it to the specified shaderKey.
- void **CompileShader** (unsigned int shaderKey, std::wstring_view filename, std::string_view entryPointName, std::string_view target)
Loads a shader file, compiles it into bytecode and and maps the bytecode to the specified shaderKey.
- void **RemoveShader** (unsigned int shaderKey)
Removes the shader bytecode mapped to the specified shaderKey.
- void **LoadShader** (std::wstring_view shaderKey, std::wstring_view filename)
Loads a shaders bytecode and maps it to the specified shaderKey.
- void **CompileShader** (std::wstring_view shaderKey, std::wstring_view filename, std::string_view entryPointName, std::string_view target)
Loads a shader file, compiles it into bytecode and and maps the bytecode to the specified shaderKey.
- void **RemoveShader** (std::wstring_view shaderKey)
Removes the shader bytecode mapped to the specified shaderKey.
- void **CreateInputElementDescription** (unsigned int key, const char *semanticName, unsigned int semanticIndex, DXGI_FORMAT format, unsigned int inputSlot, unsigned int byteOffset, D3D12_INPUT_CLASSIFICATION inputSlotClass=D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA, unsigned int instanceStepRate=0)
Creates an input element description and stores in an array mapped to the specified key.
- void **CreateInputElementDescription** (std::wstring_view key, const char *semanticName, unsigned int semanticIndex, DXGI_FORMAT format, unsigned int inputSlot, unsigned int byteOffset, D3D12_INPUT_CLASSIFICATION inputSlotClass=D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA, unsigned int instanceStepRate=0)
Creates an input element description and stores in an array mapped to the specified key.
- void **CreateRootDescriptor** (unsigned int rootParameterKey, unsigned int shaderRegister)
Creates a root descriptor and stores it in the array mapped to the specified rootParameterKey.
- void **CreateDescriptorRange** (unsigned int descriptorRangeKey, D3D12_DESCRIPTOR_RANGE_TYPE type, unsigned int numDescriptors, unsigned int shaderRegister, unsigned int registerSpace, unsigned int offset)
Creates a descriptor range and stores it in the array mapped to the specified descriptorRangeKey.
- void **CreateDescriptorTable** (unsigned int rootParameterKey, unsigned int descriptorRangeKey)
Creates a root descriptor table and stores it in the array mapped to the specified rootParameterKey.
- void **CreateRootConstants** (unsigned int rootParameterKey, unsigned int shaderRegister, unsigned int numValues)
Creates a root constant and stores it in the array mapped to the specified rootParameterKey.
- void **CreateRootDescriptor** (std::wstring_view rootParameterKey, unsigned int shaderRegister)
Creates a root descriptor and stores it in the array mapped to the specified rootParameterKey.
- void **CreateDescriptorRange** (std::wstring_view descriptorRangeKey, D3D12_DESCRIPTOR_RANGE_TYPE type, unsigned int numDescriptors, unsigned int shaderRegister, unsigned int registerSpace, unsigned int offset)
Creates a descriptor range and stores it in the array mapped to the specified descriptorRangeKey.
- void **CreateDescriptorTable** (std::wstring_view rootParameterKey, unsigned int descriptorRangeKey)
Creates a root descriptor table and stores it in the array mapped to the specified rootParameterKey.
- void **CreateRootConstants** (std::wstring_view rootParameterKey, unsigned int shaderRegister, unsigned int numValues)

- Creates a root constant and stores it in the array mapped to the specified rootParameterKey.*

 - void [CreateRootSignature](#) (unsigned int rootSigKey, unsigned int rootParametersKey)

Creates a root signature and maps it to the specified rootSigKey.

 - void [CreateRootSignature](#) (unsigned int rootSigKey, unsigned int rootParametersKey, unsigned int staticSamplerKey)

Creates a root signature and maps it to the specified rootSigKey.

 - void [CreateStaticSampler](#) (unsigned int staticSamplerKey, D3D12_FILTER filter, D3D12_TEXTURE_ADDRESS_MODE u, D3D12_TEXTURE_ADDRESS_MODE v, D3D12_TEXTURE_ADDRESS_MODE w, unsigned int shaderRegister)

Creates a static sampler and stores in an array mapped to the specified key.

 - void [CreateRootSignature](#) (std::wstring_view rootSigKey, std::wstring_view rootParametersKey)

Creates a root signature and maps it to the specified rootSigKey.

 - void [CreateRootSignature](#) (std::wstring_view rootSigKey, std::wstring_view rootParametersKey, std::wstring_view staticSamplerKey)

Creates a root signature and maps it to the specified rootSigKey.

 - void [CreateStaticSampler](#) (std::wstring_view staticSamplerKey, D3D12_FILTER filter, D3D12_TEXTURE_ADDRESS_MODE u, D3D12_TEXTURE_ADDRESS_MODE v, D3D12_TEXTURE_ADDRESS_MODE w, unsigned int shaderRegister)

Creates a static sampler and stores in an array mapped to the specified key.

 - void [CreatePSO](#) (unsigned int psoKey, D3D12_FILL_MODE fillMode, BOOL enableMultisample, unsigned int vsKey, unsigned int psKey, unsigned int inputElementDescriptionsKey, unsigned int rootSigKey, const D3D12_PRIMITIVE_TOPOLOGY_TYPE &primitiveType, UINT sampleCount=1)

Creates a PSO and maps it to the specified psoKey.

 - void [LinkPSOAndRootSignature](#) (unsigned int psoKey, unsigned int rootSigKey)

Links the PSO and its associated root signature to the pipeline to indicate what settings you want to use to render objects. An out_of_range exception is thrown if any of the keys don't have a mapped values.

 - void [CreatePSO](#) (std::wstring_view psoKey, D3D12_FILL_MODE fillMode, BOOL enableMultisample, std::wstring_view vsKey, std::wstring_view psKey, std::wstring_view inputElementDescriptionsKey, std::wstring_view rootSigKey, const D3D12_PRIMITIVE_TOPOLOGY_TYPE &primitiveType, UINT sampleCount=1)

Creates a PSO and maps it to the specified psoKey.

 - void [LinkPSOAndRootSignature](#) (std::wstring_view psoKey, std::wstring_view rootSigKey)

Links the PSO and its associated root signature to the pipeline to indicate what settings you want to use to render objects. An out_of_range exception is thrown if any of the keys don't have a mapped values.

 - void [CreateStaticBuffer](#) (unsigned int staticBufferKey, const void *data, unsigned numBytes, unsigned int stride)

Creates a static vertex buffer and stores the specified data in the buffer.

 - void [CreateStaticBuffer](#) (unsigned int staticBufferKey, const void *data, unsigned numBytes, DXGI_FORMAT format)

Creates a static index buffer and stores the specified data in the buffer.

 - void [CreateStaticBuffer](#) (unsigned int staticBufferKey, const wchar_t *filename, unsigned int texType, unsigned int index)

Creates a static texture buffer, stores the data from the file into the buffer and creates a view of the texture.

 - void [LinkStaticBuffer](#) (unsigned int bufferType, unsigned int staticBufferKey)

Links the static buffer mapped to the static buffer key to the pipeline.

 - void [CreateStaticBuffer](#) (std::wstring_view staticBufferKey, const void *data, unsigned numBytes, unsigned int stride)

Creates a static vertex buffer and stores the specified data in the buffer.

 - void [CreateStaticBuffer](#) (std::wstring_view staticBufferKey, const void *data, unsigned numBytes, DXGI_FORMAT format)

Creates a static index buffer and stores the specified data in the buffer.

 - void [CreateStaticBuffer](#) (std::wstring_view staticBufferKey, const wchar_t *filename, unsigned int texType, unsigned int index)

Creates a static texture buffer, stores the data from the file into the buffer and creates a view of the texture.

- void [LinkStaticBuffer](#) (unsigned int bufferType, std::wstring_view staticBufferKey)
Links the static buffer mapped to the static buffer key to the pipeline.
- void [CreateDynamicBuffer](#) (unsigned int dynamicBufferKey, unsigned numBytes, const void *data, unsigned int stride)
Creates a dynamic vertex buffer or a dynamic constant buffer.
- void [CreateDynamicBuffer](#) (unsigned int dynamicBufferKey, unsigned numBytes, const void *data, DXGI_FORMAT format)
Creates a dynamic index buffer.
- void [LinkDynamicBuffer](#) (unsigned int bufferType, unsigned int dynamicBufferKey, unsigned int index↔ ConstantData=0, unsigned int rootParameterIndex=0)
Links the dynamic buffer mapped to the dynamic buffer key to the pipeline.
- void [CopyDataIntoDynamicBuffer](#) (unsigned int dynamicBufferKey, unsigned int index, const void *data, UINT64 numBytes)
Copies the specified data into the dynamic buffer mapped to the dynamic buffer key.
- void [CreateDynamicBuffer](#) (std::wstring_view dynamicBufferKey, unsigned numBytes, const void *data, unsigned int stride)
Creates a dynamic vertex buffer or a dynamic constant buffer.
- void [CreateDynamicBuffer](#) (std::wstring_view dynamicBufferKey, unsigned numBytes, const void *data, DXGI_FORMAT format)
Creates a dynamic index buffer.
- void [LinkDynamicBuffer](#) (unsigned int bufferType, std::wstring_view dynamicBufferKey, unsigned int index↔ ConstantData=0, unsigned int rootParameterIndex=0)
Links the dynamic buffer mapped to the dynamic buffer key to the pipeline.
- void [CopyDataIntoDynamicBuffer](#) (std::wstring_view dynamicBufferKey, unsigned int index, const void *data, UINT64 numBytes)
Copies the specified data into the dynamic buffer mapped to the dynamic buffer key.
- void [CreateTextureViewHeap](#) (unsigned int numDescriptors)
Creates a descriptor heap to store views of textures.
- void [LinkTextureViewHeap](#) ()
Links the texture view heap to the pipeline.
- void [LinkTexture](#) (unsigned int rootParameterIndex)
Links the set of textures in the descriptor table to the pipeline.
- void [LinkTexture](#) (unsigned int rootParameterIndex, unsigned int textureViewIndex)
Links a texture to the pipeline.
- void [BeforeRenderObjects](#) (bool isMSAAEnabled=false)
Puts all of the commands needed in the command list before drawing the objects of the scene.
- void [RenderObject](#) (unsigned int indexCount, unsigned int locationFirstIndex, int indexOfFirstVertex, D3D_↔ PRIMITIVE_TOPOLOGY primitive)
Renders an object with the specified draw arguments.
- void [AfterRenderObjects](#) (bool isMSAAEnabled=false, bool isTextEnabled=false)
Transitions the render target buffer to the correct state and executes commands.
- void [BeforeRenderText](#) ()
Puts all of the commands needed in the command list before drawing the text of the scene. Call before calling the first RenderText function.
- void [RenderText](#) (const vec4 &textLocation, const [Color](#) &textColor, float textSize, const std::wstring &text↔ String, DWRITE_PARAGRAPH_ALIGNMENT alignment=DWRITE_PARAGRAPH_ALIGNMENT_CENTER)
Draws the [Text](#) object mapped to the specified textKey. Call in between a BeforeRenderText function and a After↔ RenderText function.
- void [AfterRenderText](#) ()
Transitions the render target buffer and executes all of the text drawing commands.
- void [AfterRender](#) ()

- Presents and signals (puts a fence command in the command queue).*
 - void [ExecuteAndFlush](#) ()
 - Executes the commands to fill the vertex and index buffer with data and flushes the queue.*
 - void [Resize](#) (unsigned int width, unsigned int height, HWND windowHandle, bool isMSAAEnabled=false, bool isTextEnabled=false)
 - Resizes the window-dependent resources when the window gets resized.*
 - void [SetConstants](#) (unsigned int rootParameterIndex, unsigned int numValues, void *data, unsigned int index)
 - Links an array of 32-bit values to the pipeline.*

5.11.1 Detailed Description

This class is used to render a scene using Direct3D 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

5.11.2 Member Function Documentation

5.11.2.1 AfterRender()

```
void RenderingEngine::RenderScene::AfterRender ( )
```

Presents and signals (puts a fence command in the command queue).

Call after rendering all your objects and text.

5.11.2.2 AfterRenderObjects()

```
void RenderingEngine::RenderScene::AfterRenderObjects (
    bool isMSAAEnabled = false,
    bool isTextEnabled = false )
```

Transitions the render target buffer to the correct state and executes commands.

Parameters

<i>[in,optional]</i>	isMSAAEnabled Pass in true if MSAA is enabled.
<i>[in,optional]</i>	isTextEnabled Pass in true if text is enabled.

5.11.2.3 AfterRenderText()

```
void RenderingEngine::RenderScene::AfterRenderText ( )
```

Transitions the render target buffer and executes all of the text drawing commands.

Call after calling all the RenderText functions.

5.11.2.4 BeforeRenderObjects()

```
void RenderingEngine::RenderScene::BeforeRenderObjects (
    bool isMSAAEnabled = false )
```

Puts all of the commands needed in the command list before drawing the objects of the scene.

Call before calling the first RenderObjects function.

Parameters

<i>[in, optional]</i>	isMSAAEnabled Pass in true if MSAA is enabled.
-----------------------	--

5.11.2.5 BeforeRenderText()

```
void RenderingEngine::RenderScene::BeforeRenderText ( )
```

Puts all of the commands needed in the command list before drawing the text of the scene. Call before calling the first RenderText function.

5.11.2.6 CompileShader() [1/2]

```
void RenderingEngine::RenderScene::CompileShader (
    std::wstring_view shaderKey,
    std::wstring_view filename,
    std::string_view entryPointName,
    std::string_view target )
```

Loads a shader file, compiles it into bytecode and and maps the bytecode to the specified *shaderKey*.

Parameters

in	<i>shaderKey</i>	The key to map the bytecode to.
in	<i>filename</i>	The name of the .hlsl file.
in	<i>entryPointName</i>	The name of the entry point in the .hlsl file.
in	<i>target</i>	The name of the shader target to compile with.

5.11.2.7 CompileShader() [2/2]

```
void RenderingEngine::RenderScene::CompileShader (
    unsigned int shaderKey,
    std::wstring_view filename,
```

```
std::string_view entryPointName,
std::string_view target )
```

Loads a shader file, compiles it into bytecode and maps the bytecode to the specified *shaderKey*.

Parameters

in	<i>shaderKey</i>	The key to map the bytecode to.
in	<i>filename</i>	The name of the .hlsl file.
in	<i>entryPointName</i>	The name of the entry point in the .hlsl file.
in	<i>target</i>	The name of the shader target to compile with.

5.11.2.8 CopyDataIntoDynamicBuffer() [1/2]

```
void RenderingEngine::RenderScene::CopyDataIntoDynamicBuffer (
    std::wstring_view dynamicBufferKey,
    unsigned int index,
    const void * data,
    UINT64 numOfBytes )
```

Copies the specified data into the dyanmic buffer mapped to the dynamic buffer key.

Parameters

in	<i>dynamicBufferKey</i>	The key mapped to a dynamic buffer.
in	<i>index</i>	The index of where to copy the data to.
in	<i>data</i>	The data to copy.
in	<i>numOfBytes</i>	The number of bytes to copy.

5.11.2.9 CopyDataIntoDynamicBuffer() [2/2]

```
void RenderingEngine::RenderScene::CopyDataIntoDynamicBuffer (
    unsigned int dynamicBufferKey,
    unsigned int index,
    const void * data,
    UINT64 numOfBytes )
```

Copies the specified data into the dyanmic buffer mapped to the dynamic buffer key.

Parameters

in	<i>dynamicBufferKey</i>	The key mapped to a dynamic buffer.
in	<i>index</i>	The index of where to copy the data to.
in	<i>data</i>	The data to copy.
in	<i>numOfBytes</i>	The number of bytes to copy.

5.11.2.10 CreateDescriptorRange() [1/2]

```
void RenderingEngine::RenderScene::CreateDescriptorRange (
    std::wstring_view descriptorRangeKey,
    D3D12_DESCRIPTOR_RANGE_TYPE type,
    unsigned int numDescriptors,
    unsigned int shaderRegister,
    unsigned int registerSpace,
    unsigned int offset )
```

Creates a descriptor range and stores it in the array mapped to the specified *descriptorRangeKey*.

Parameters

in	<i>descriptorRangeKey</i>	The key to an array of descriptor ranges to store the descriptor range in.
in	<i>type</i>	The type of descriptor range.
in	<i>numDescriptors</i>	The number of descriptors in the range.
in	<i>shaderRegister</i>	The shader register the views are mapped to.
in	<i>registerSpace</i>	The space of the shader register.
in	<i>offset</i>	The offset in descriptors, from the start of the descriptor table.

5.11.2.11 CreateDescriptorRange() [2/2]

```
void RenderingEngine::RenderScene::CreateDescriptorRange (
    unsigned int descriptorRangeKey,
    D3D12_DESCRIPTOR_RANGE_TYPE type,
    unsigned int numDescriptors,
    unsigned int shaderRegister,
    unsigned int registerSpace,
    unsigned int offset )
```

Creates a descriptor range and stores it in the array mapped to the specified *descriptorRangeKey*.

Parameters

in	<i>descriptorRangeKey</i>	The key to an array of descriptor ranges to store the descriptor range in.
in	<i>type</i>	The type of descriptor range.
in	<i>numDescriptors</i>	The number of descriptors in the range.
in	<i>shaderRegister</i>	The shader register the views are mapped to.
in	<i>registerSpace</i>	The space of the shader register.
in	<i>offset</i>	The offset in descriptors, from the start of the descriptor table.

5.11.2.12 CreateDescriptorTable() [1/2]

```
void RenderingEngine::RenderScene::CreateDescriptorTable (
    std::wstring_view rootParameterKey,
    unsigned int descriptorRangeKey )
```

Creates a root descriptor table and stores it in the array mapped to the specified *rootParameterKey*.

Parameters

in	<i>rootParameterKey</i>	The key to a mapped array to store the created root parameter in.
in	<i>descriptorRangeKey</i>	The key to an array of descriptor ranges.

5.11.2.13 CreateDescriptorTable() [2/2]

```
void RenderingEngine::RenderScene::CreateDescriptorTable (
    unsigned int rootParameterKey,
    unsigned int descriptorRangeKey )
```

Creates a root descriptor table and stores it in the array mapped to the specified *rootParameterKey*.

Parameters

in	<i>rootParameterKey</i>	The key to a mapped array to store the created root parameter in.
in	<i>descriptorRangeKey</i>	The key to an array of descriptor ranges.

5.11.2.14 CreateDynamicBuffer() [1/4]

```
void RenderingEngine::RenderScene::CreateDynamicBuffer (
    std::wstring_view dynamicBufferKey,
    unsigned numBytes,
    const void * data,
    DXGI_FORMAT format )
```

Creates a dynamic index buffer.

The user can update the data on a per-frame basis.

If the specified key is already mapped to a dynamic buffer, this function does nothing.

Parameters

in	<i>dynamicBufferKey</i>	The key to map the dynamic buffer to.
in	<i>numBytes</i>	The number of bytes to allocate for the dynamic buffer.
in	<i>data</i>	The data you want to copy into the dynamic buffer. Pass in nullptr if you don't want to copy data into the buffer on creation.
in	<i>format</i>	The number of bytes to get from one element to the next element.

5.11.2.15 CreateDynamicBuffer() [2/4]

```
void RenderingEngine::RenderScene::CreateDynamicBuffer (
    std::wstring_view dynamicBufferKey,
    unsigned numBytes,
    const void * data,
    unsigned int stride )
```

Creates a dynamic vertex buffer or a dynamic constant buffer.

The user can update the data on a per-frame basis.

If the specified key is already mapped to a dynamic buffer, this function does nothing.

Parameters

in	<i>dynamicBufferKey</i>	The key to map the dynamic buffer to.
in	<i>numBytes</i>	The number of bytes to allocate for the dynamic buffer.
in	<i>data</i>	The data you want to copy into the dynamic buffer. Pass in nullptr if you don't want to copy data into the buffer on creation.
in	<i>stride</i>	The number of bytes to get from one element to the next element.

5.11.2.16 CreateDynamicBuffer() [3/4]

```
void RenderingEngine::RenderScene::CreateDynamicBuffer (
    unsigned int dynamicBufferKey,
    unsigned numBytes,
    const void * data,
    DXGI_FORMAT format )
```

Creates a dynamic index buffer.

The user can update the data on a per-frame basis.

If the specified key is already mapped to a dynamic buffer, this function does nothing.

Parameters

in	<i>dynamicBufferKey</i>	The key to map the dynamic buffer to.
in	<i>numBytes</i>	The number of bytes to allocate for the dynamic buffer.
in	<i>data</i>	The data you want to copy into the dynamic buffer. Pass in nullptr if you don't want to copy data into the buffer on creation.
in	<i>format</i>	The number of bytes to get from one element to the next element.

5.11.2.17 CreateDynamicBuffer() [4/4]

```
void RenderingEngine::RenderScene::CreateDynamicBuffer (
    unsigned int dynamicBufferKey,
    unsigned numBytes,
    const void * data,
    unsigned int stride )
```

Creates a dynamic vertex buffer or a dynamic constant buffer.

The user can update the data on a per-frame basis.

If the specified key is already mapped to a dynamic buffer, this function does nothing.

Parameters

in	<i>dynamicBufferKey</i>	The key to map the dynamic buffer to.
in	<i>numBytes</i>	The number of bytes to allocate for the dynamic buffer.
in	<i>data</i>	The data you want to copy into the dynamic buffer. Pass in nullptr if you don't want to copy data into the buffer on creation.
in	<i>stride</i>	The number of bytes to get from one element to the next element.

5.11.2.18 CreateInputElementDescription() [1/2]

```
void RenderingEngine::RenderScene::CreateInputElementDescription (
    std::wstring_view key,
    const char * semanticName,
    unsigned int semanticIndex,
    DXGI_FORMAT format,
    unsigned int inputSlot,
    unsigned int byteOffset,
    D3D12_INPUT_CLASSIFICATION inputSlotClass = D3D12_INPUT_CLASSIFICATION_PER_↔
    VERTEX_DATA,
    unsigned int instanceStepRate = 0 )
```

Creates an input element description and stores in an array mapped to the specified *key*.

Parameters

in	<i>key</i>	The key to a mapped array to store the created input element description.
in	<i>semanticName</i>	The name of the application variable linked to a shader variable.
in	<i>semanticIndex</i>	The index to attach to the semanticName.
in	<i>format</i>	The data type of input element being described.
in	<i>inputSlot</i>	The input slot the input element will come from.
in	<i>byteOffset</i>	The offset in bytes to get to the input element being described.
	<i>[in,optional]</i>	<i>inputSlotClass</i> The data class for an input slot. Used for instancing.
	<i>[in,optional]</i>	<i>instanceStepRate</i> The number of instances to render. Used for instancing.

5.11.2.19 CreateInputElementDescription() [2/2]

```
void RenderingEngine::RenderScene::CreateInputElementDescription (
    unsigned int key,
    const char * semanticName,
    unsigned int semanticIndex,
    DXGI_FORMAT format,
    unsigned int inputSlot,
    unsigned int byteOffset,
    D3D12_INPUT_CLASSIFICATION inputSlotClass = D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA,
    unsigned int instanceStepRate = 0 )
```

Creates an input element description and stores in an array mapped to the specified *key*.

Parameters

in	<i>key</i>	The key to a mapped array to store the created input element description.
in	<i>semanticName</i>	The name of the application variable linked to a shader variable.
in	<i>semanticIndex</i>	The index to attach to the semanticName.
in	<i>format</i>	The data type of input element being described.
in	<i>inputSlot</i>	The input slot the input element will come from.
in	<i>byteOffset</i>	The offset in bytes to get to the input element being described.
	[in,optional]	inputSlotClass The data class for an input slot. Used for instancing.
	[in,optional]	instanceStepRate The number of instances to render. Used for instancing.

5.11.2.20 CreatePSO() [1/2]

```
void RenderingEngine::RenderScene::CreatePSO (
    std::wstring_view psoKey,
    D3D12_FILL_MODE fillMode,
    BOOL enableMultisample,
    std::wstring_view vsKey,
    std::wstring_view psKey,
    std::wstring_view inputElementDescriptionsKey,
    std::wstring_view rootSigKey,
    const D3D12_PRIMITIVE_TOPOLOGY_TYPE & primitiveType,
    UINT sampleCount = 1 )
```

Creates a PSO and maps it to the specified *psoKey*.

If any of the shader keys or the input element descripton key or the root signature key does not have a mapped value an out_of_range exception is thrown.

Parameters

in	<i>psoKey</i>	The key to map the created PSO to.
in	<i>fillMode</i>	The fill mode to use when rendering triangles. Use D3D12_FILL_MODE_WIREFRAME for wireframe and D3D12_FILL_MODE_SOLID for solid.
in	<i>enableMultisample</i>	Pass in TRUE to use multi-sampling, FALSE otherwise.

Parameters

in	<i>vsKey</i>	A key to a mapped vertex shader.
in	<i>psKey</i>	A key to a mapped pixel shader.
in	<i>inputElementDescriptionsKey</i>	A key to a mapped array of input element descriptions for the specified vertex and pixel shaders.
in	<i>rootSigKey</i>	A key to a mapped root signature.
in	<i>primitiveType</i>	The type of primitive to connect vertices into.
	<i>[in,optional]</i>	sampleCount The number of samples. If enableMultiSample is TRUE pass in 4. All other values will cause an error.

5.11.2.21 CreatePSO() [2/2]

```
void RenderingEngine::RenderScene::CreatePSO (
    unsigned int psoKey,
    D3D12_FILL_MODE fillMode,
    BOOL enableMultisample,
    unsigned int vsKey,
    unsigned int psKey,
    unsigned int inputElementDescriptionsKey,
    unsigned int rootSigKey,
    const D3D12_PRIMITIVE_TOPOLOGY_TYPE & primitiveType,
    UINT sampleCount = 1 )
```

Creates a PSO and maps it to the specified *psoKey*.

If any of the shader keys or the input element descripton key or the root signature key does not have a mapped value an out_of_range exception is thrown.

Parameters

in	<i>psoKey</i>	The key to map the created PSO to.
in	<i>fillMode</i>	The fill mode to use when rendering triangles. Use D3D12_FILL_MODE_WIREFRAME for wireframe and D3D12_FILL_MODE_SOLID for solid.
in	<i>enableMultisample</i>	Pass in TRUE to use multi-sampling, FALSE otherwise.
in	<i>vsKey</i>	A key to a mapped vertex shader.
in	<i>psKey</i>	A key to a mapped pixel shader.
in	<i>inputElementDescriptionsKey</i>	A key to a mapped array of input element descriptions for the specified vertex and pixel shaders.
in	<i>rootSigKey</i>	A key to a mapped root signature.
in	<i>primitiveType</i>	The type of primitive to connect vertices into.
	<i>[in,optional]</i>	sampleCount The number of samples. If enableMultiSample is TRUE pass in 4. All other values will cause an error.

5.11.2.22 CreateRootConstants() [1/2]

```
void RenderingEngine::RenderScene::CreateRootConstants (
    std::wstring_view rootParameterKey,
    unsigned int shaderRegister,
    unsigned int numValues )
```

Creates a root constant and stores it in the array mapped to the specified *rootParameterKey*.

Parameters

in	<i>rootParameterKey</i>	The key to a mapped array to store the created root parameter in.
in	<i>shaderRegister</i>	The register where constant data will be stored.
in	<i>numValues</i>	The number of 32-bit values.

5.11.2.23 CreateRootConstants() [2/2]

```
void RenderingEngine::RenderScene::CreateRootConstants (
    unsigned int rootParameterKey,
    unsigned int shaderRegister,
    unsigned int numValues )
```

Creates a root constant and stores it in the array mapped to the specified *rootParameterKey*.

Parameters

in	<i>rootParameterKey</i>	The key to a mapped array to store the created root parameter in.
in	<i>shaderRegister</i>	The register where constant data will be stored.
in	<i>numValues</i>	The number of 32-bit values.

5.11.2.24 CreateRootDescriptor() [1/2]

```
void RenderingEngine::RenderScene::CreateRootDescriptor (
    std::wstring_view rootParameterKey,
    unsigned int shaderRegister )
```

Creates a root descriptor and stores it in the array mapped to the specified *rootParameterKey*.

Parameters

in	<i>rootParameterKey</i>	The key to a mapped array to store the created root parameter in.
in	<i>shaderRegister</i>	The register where constant data will be stored.

5.11.2.25 CreateRootDescriptor() [2/2]

```
void RenderingEngine::RenderScene::CreateRootDescriptor (
    unsigned int rootParameterKey,
    unsigned int shaderRegister )
```

Creates a root descriptor and stores it in the array mapped to the specified *rootParameterKey*.

Parameters

in	<i>rootParameterKey</i>	The key to a mapped array to store the created root parameter in.
in	<i>shaderRegister</i>	The register where constant data will be stored.

5.11.2.26 CreateRootSignature() [1/4]

```
void RenderingEngine::RenderScene::CreateRootSignature (
    std::wstring_view rootSigKey,
    std::wstring_view rootParametersKey )
```

Creates a root signature and maps it to the specified *rootSigKey*.

If the *rootParameterKey* does not have a mapped value an `out_of_range` exception is thrown.

Parameters

in	<i>rootSigKey</i>	The key to map the created root signature to.
in	<i>rootParameterKey</i>	The key to a mapped array of root parameters.

5.11.2.27 CreateRootSignature() [2/4]

```
void RenderingEngine::RenderScene::CreateRootSignature (
    std::wstring_view rootSigKey,
    std::wstring_view rootParametersKey,
    std::wstring_view staticsSamplerKey )
```

Creates a root signature and maps it to the specified *rootSigKey*.

If the *rootParameterKey* or *staticsSamplerKey* does not have a mapped value an `out_of_range` exception is thrown.

Parameters

in	<i>rootSigKey</i>	The key to map the created root signature to.
in	<i>rootParameterKey</i>	The key to a mapped array of root parameters.
in	<i>numStaticSamplers</i>	The number of static samplers.
in	<i>staticsSamplerKey</i>	The key to an array of static samplers.

5.11.2.28 CreateRootSignature() [3/4]

```
void RenderingEngine::RenderScene::CreateRootSignature (
    unsigned int rootSigKey,
    unsigned int rootParametersKey )
```

Creates a root signature and maps it to the specified *rootSigKey*.

If the *rootParameterKey* does not have a mapped value an out_of_range exception is thrown.

Parameters

in	<i>rootSigKey</i>	The key to map the created root signature to.
in	<i>rootParameterKey</i>	The key to a mapped array of root parameters.

5.11.2.29 CreateRootSignature() [4/4]

```
void RenderingEngine::RenderScene::CreateRootSignature (
    unsigned int rootSigKey,
    unsigned int rootParametersKey,
    unsigned int staticsSamplerKey )
```

Creates a root signature and maps it to the specified *rootSigKey*.

If the *rootParameterKey* or *staticsSamplerKey* does not have a mapped value an out_of_range exception is thrown.

Parameters

in	<i>rootSigKey</i>	The key to map the created root signature to.
in	<i>rootParameterKey</i>	The key to a mapped array of root parameters.
in	<i>numStaticSamplers</i>	The number of static samplers.
in	<i>staticsSamplerKey</i>	The key to an array of static samplers.

5.11.2.30 CreateStaticBuffer() [1/6]

```
void RenderingEngine::RenderScene::CreateStaticBuffer (
    std::wstring_view staticBufferKey,
    const void * data,
    unsigned numBytes,
    DXGI_FORMAT format )
```

Creates a static index buffer and stores the specified *data* in the buffer.

The user cannot update/change the data once it is stored in the buffer.

If the specified key is already mapped to a static buffer, this function does nothing.

Parameters

in	<i>staticBufferKey</i>	The key to map the static buffer to.
in	<i>numBytes</i>	The number of bytes to allocate for the static buffer.
in	<i>format</i>	The number of bytes to get from one element to the next element.

5.11.2.31 CreateStaticBuffer() [2/6]

```
void RenderingEngine::RenderScene::CreateStaticBuffer (
    std::wstring_view staticBufferKey,
    const void * data,
    unsigned numBytes,
    unsigned int stride )
```

Creates a static vertex buffer and stores the specified *data* in the buffer.

The user cannot update/change the data once it is stored in the buffer.

If the specified key is already mapped to a static buffer, this function does nothing.

Parameters

in	<i>staticBufferKey</i>	The key to map the static buffer to.
in	<i>numBytes</i>	The number of bytes to allocate for the static buffer.
in	<i>stride</i>	The number of bytes to get from one element to the next element.

5.11.2.32 CreateStaticBuffer() [3/6]

```
void RenderingEngine::RenderScene::CreateStaticBuffer (
    std::wstring_view staticBufferKey,
    const wchar_t * filename,
    unsigned int texType,
    unsigned int index )
```

Creates a static texture buffer, stores the data from the file into the buffer and creates a view of the texture.

The user cannot update/change the data once it is stored in the buffer.

If the specified key is already mapped to a static buffer, this function does nothing.

Parameters

in	<i>staticBufferKey</i>	The key to map the static buffer to.
in	<i>numBytes</i>	The number of bytes to allocate for the static buffer.
in	<i>filename</i>	The filename of the texture.
in	<i>texType</i>	The type of texture. Pass in FARender::Tex2D for a 2D texture or FARender::Tex2D_MS for a multi-sampled 2D texture.
in	<i>index</i>	Where to store the description (view) of the texture in a shader resource view heap.

5.11.2.33 CreateStaticBuffer() [4/6]

```
void RenderingEngine::RenderScene::CreateStaticBuffer (
    unsigned int staticBufferKey,
    const void * data,
    unsigned numBytes,
    DXGI_FORMAT format )
```

Creates a static index buffer and stores the specified *data* in the buffer.

The user cannot update/change the data once it is stored in the buffer.

If the specified key is already mapped to a static buffer, this function does nothing.

Parameters

in	<i>staticBufferKey</i>	The key to map the static buffer to.
in	<i>numBytes</i>	The number of bytes to allocate for the static buffer.
in	<i>format</i>	The number of bytes to get from one element to the next element.

5.11.2.34 CreateStaticBuffer() [5/6]

```
void RenderingEngine::RenderScene::CreateStaticBuffer (
    unsigned int staticBufferKey,
    const void * data,
    unsigned numBytes,
    unsigned int stride )
```

Creates a static vertex buffer and stores the specified *data* in the buffer.

The user cannot update/change the data once it is stored in the buffer.

If the specified key is already mapped to a static buffer, this function does nothing.

Parameters

in	<i>staticBufferKey</i>	The key to map the static buffer to.
in	<i>numBytes</i>	The number of bytes to allocate for the static buffer.
in	<i>stride</i>	The number of bytes to get from one element to the next element.

5.11.2.35 CreateStaticBuffer() [6/6]

```
void RenderingEngine::RenderScene::CreateStaticBuffer (
    unsigned int staticBufferKey,
    const wchar_t * filename,
```

```
    unsigned int texType,
    unsigned int index )
```

Creates a static texture buffer, stores the data from the file into the buffer and creates a view of the texture.

The user cannot update/change the data once it is stored in the buffer.

If the specified key is already mapped to a static buffer, this function does nothing.

Parameters

in	<i>staticBufferKey</i>	The key to map the static buffer to.
in	<i>numBytes</i>	The number of bytes to allocate for the static buffer.
in	<i>filename</i>	The filename of the texture.
in	<i>texType</i>	The type of texture. Pass in <code>FARender::Tex2D</code> for a 2D texture or <code>FARender::Tex2D_MS</code> for a multi-sampled 2D texture.
in	<i>index</i>	Where to store the description (view) of the texture in a shader resource view heap.

5.11.2.36 CreateStaticSampler() [1/2]

```
void RenderingEngine::RenderScene::CreateStaticSampler (
    std::wstring_view staticSamplerKey,
    D3D12_FILTER filter,
    D3D12_TEXTURE_ADDRESS_MODE u,
    D3D12_TEXTURE_ADDRESS_MODE v,
    D3D12_TEXTURE_ADDRESS_MODE w,
    unsigned int shaderRegister )
```

Creates a static sampler and stores in an an array mapped to the specified key.

Parameters

in	<i>staticSamplerKey</i>	The key to an array of static samplers.
in	<i>filter</i>	The filtering method to use when sampling a texture.
in	<i>u</i>	The address mode for the u texture coordinate.
in	<i>v</i>	The address mode for the v texture coordinate.
in	<i>w</i>	The address mode for the w texture coordinate.
in	<i>shaderRegister</i>	The register the sampler is linked to.

5.11.2.37 CreateStaticSampler() [2/2]

```
void RenderingEngine::RenderScene::CreateStaticSampler (
    unsigned int staticSamplerKey,
    D3D12_FILTER filter,
    D3D12_TEXTURE_ADDRESS_MODE u,
    D3D12_TEXTURE_ADDRESS_MODE v,
```

```
D3D12_TEXTURE_ADDRESS_MODE w,  
unsigned int shaderRegister )
```

Creates a static sampler and stores in an an array mapped to the specified key.

Parameters

in	<i>staticSamplerKey</i>	The key to an array of static samplers.
in	<i>filter</i>	The filtering method to use when sampling a texture.
in	<i>u</i>	The address mode for the u texture coordinate.
in	<i>v</i>	The address mode for the v texture coordinate.
in	<i>w</i>	The address mode for the w texture coordinate.
in	<i>shaderRegister</i>	The register the sampler is linked to.

5.11.2.38 CreateTextureViewHeap()

```
void RenderingEngine::RenderScene::CreateTextureViewHeap (
    unsigned int numDescriptors )
```

Creates a descriptor heap to store views of textures.

Parameters

in	<i>numDescriptors</i>	The number of views to be stored in the heap.
----	-----------------------	---

5.11.2.39 ExecuteAndFlush()

```
void RenderingEngine::RenderScene::ExecuteAndFlush ( )
```

Executes the commands to fill the vertex and index buffer with data and flushes the queue.

5.11.2.40 LinkDynamicBuffer() [1/2]

```
void RenderingEngine::RenderScene::LinkDynamicBuffer (
    unsigned int bufferType,
    std::wstring_view dynamicBufferKey,
    unsigned int indexConstantData = 0,
    unsigned int rootParameterIndex = 0 )
```

Links the dynamic buffer mapped to the dynamic buffer key to the pipeline.

An `out_of_range` exception is thrown if the dynamic buffer key does not have a mapped dynamic buffer.

Parameters

in		
----	--	--

The type of buffer. Must be the values 0, 1 or 2. If it isn't one of those values a `runtime_error` exception is thrown. If 0 the mapped dynamic vertex buffer is linked. If 1 the mapped dynamic index buffer is linked. If 2 the mapped dynamic constant buffer is linked.

Parameters

<code>in</code>	<i>dynamicBufferKey</i>	The key mapped to a dynamic buffer.
	<i>[in,optional]</i>	<code>indexConstantData</code> The index of where the constant data is in the dynamic buffer.
	<i>[in,optional]</i>	<code>rootParameterIndex</code> The index of the root parameter in the root signature that has the register the constant data in the dynamic constant buffer will be stored in.

The parameters `indexConstantData` `rootParameterIndex` are used if the dynamic buffer is a constant buffer.

5.11.2.41 LinkDynamicBuffer() [2/2]

```
void RenderingEngine::RenderScene::LinkDynamicBuffer (
    unsigned int bufferType,
    unsigned int dynamicBufferKey,
    unsigned int indexConstantData = 0,
    unsigned int rootParameterIndex = 0 )
```

Links the dynamic buffer mapped to the dynamic buffer key to the pipeline.

An `out_of_range` exception is thrown if the dynamic buffer key does not have a mapped dynamic buffer.

Parameters

<code>in</code>		
-----------------	--	--

The type of buffer. Must be the values 0, 1 or 2. If it isn't one of those values a `runtime_error` exception is thrown. If 0 the mapped dynamic vertex buffer is linked. If 1 the mapped dynamic index buffer is linked. If 2 the mapped dynamic constant buffer is linked.

Parameters

<code>in</code>	<i>dynamicBufferKey</i>	The key mapped to a dynamic buffer.
	<i>[in,optional]</i>	<code>indexConstantData</code> The index of where the constant data is in the dynamic buffer.
	<i>[in,optional]</i>	<code>rootParameterIndex</code> The index of the root parameter in the root signature that has the register the constant data in the dynamic constant buffer will be stored in.

The parameters `indexConstantData` `rootParameterIndex` are used if the dynamic buffer is a constant buffer.

5.11.2.42 LinkPSOAndRootSignature() [1/2]

```
void RenderingEngine::RenderScene::LinkPSOAndRootSignature (
    std::wstring_view psoKey,
    std::wstring_view rootSigKey )
```

Links the PSO and its associated root signature to the pipeline to indicate what settings you want to use to render objects. An `out_of_range` exception is thrown if any of the keys don't have a mapped values.

Parameters

in	<i>psoKey</i>	The key to a mapped PSO.
in	<i>rootSigKey</i>	The key to a mapped root signature.

5.11.2.43 LinkPSOAndRootSignature() [2/2]

```
void RenderingEngine::RenderScene::LinkPSOAndRootSignature (
    unsigned int psoKey,
    unsigned int rootSigKey )
```

Links the PSO and its associated root signature to the pipeline to indicate what settings you want to use to render objects. An `out_of_range` exception is thrown if any of the keys don't have a mapped values.

Parameters

in	<i>psoKey</i>	The key to a mapped PSO.
in	<i>rootSigKey</i>	The key to a mapped root signature.

5.11.2.44 LinkStaticBuffer() [1/2]

```
void RenderingEngine::RenderScene::LinkStaticBuffer (
    unsigned int bufferType,
    std::wstring_view staticBufferKey )
```

Links the static buffer mapped to the static buffer key to the pipeline.

An `out_of_range` exception is thrown if the static buffer key does not have a mapped static buffer.

Parameters

in	<i>bufferType</i>	The type of buffer. Must be the values 0 or 1. If it isn't one of those values a <code>runtime_error</code> exception is thrown. If 0 the mapped static vertex buffer is linked. If 1 the mapped static index buffer is linked.
in	<i>staticBufferKey</i>	The key to a mapped static buffer.

5.11.2.45 LinkStaticBuffer() [2/2]

```
void RenderingEngine::RenderScene::LinkStaticBuffer (
    unsigned int bufferType,
    unsigned int staticBufferKey )
```

Links the static buffer mapped to the static buffer key to the pipeline.

An `out_of_range` exception is thrown if the static buffer key does not have a mapped static buffer.

Parameters

in	<i>bufferType</i>	The type of buffer. Must be the values 0 or 1. If it isn't one of those values a <code>runtime_error</code> exception is thrown. If 0 the mapped static vertex buffer is linked. If 1 the mapped static index buffer is linked.
in	<i>staticBufferKey</i>	The key to a mapped static buffer.

5.11.2.46 LinkTexture() [1/2]

```
void RenderingEngine::RenderScene::LinkTexture (
    unsigned int rootParameterIndex )
```

Links the set of textures in the descriptor table to the pipeline.

Parameters

in	<i>rootParameterIndex</i>	The index of the root parameter in the root signature that has the register the texture will be stored in.
----	---------------------------	--

5.11.2.47 LinkTexture() [2/2]

```
void RenderingEngine::RenderScene::LinkTexture (
    unsigned int rootParameterIndex,
    unsigned int textureViewIndex )
```

Links a texture to the pipeline.

Parameters

in	<i>rootParameterIndex</i>	The index of the root parameter in the root signature that has the register the texture will be stored in.
in	<i>textureViewIndex</i>	The index of the view to the texture in a shader resource view heap.

5.11.2.48 LinkTextureViewHeap()

```
void RenderingEngine::RenderScene::LinkTextureViewHeap ( )
```

Links the texture view heap to the pipeline.

5.11.2.49 LoadShader() [1/2]

```
void RenderingEngine::RenderScene::LoadShader (
    std::wstring_view shaderKey,
    std::wstring_view filename )
```

Loads a shaders bytecode and maps it to the specified *shaderKey*.

Parameters

in	<i>shaderKey</i>	The key to map the bytecode to.
in	<i>filename</i>	The name of the .cso file.

5.11.2.50 LoadShader() [2/2]

```
void RenderingEngine::RenderScene::LoadShader (
    unsigned int shaderKey,
    std::wstring_view filename )
```

Loads a shaders bytecode and maps it to the specified *shaderKey*.

Parameters

in	<i>shaderKey</i>	The key to map the bytecode to.
in	<i>filename</i>	The name of the .cso file.

5.11.2.51 RemoveShader() [1/2]

```
void RenderingEngine::RenderScene::RemoveShader (
    std::wstring_view shaderKey )
```

Removes the shader bytecode mapped to the specified *shaderKey*.

If the *shaderKey* is not mapped to a value, an `out_of_range` exception is thrown.

5.11.2.52 RemoveShader() [2/2]

```
void RenderingEngine::RenderScene::RemoveShader (
    unsigned int shaderKey )
```

Removes the shader bytecode mapped to the specified *shaderKey*.

If the *shaderKey* is not mapped to a value, an `out_of_range` exception is thrown.

5.11.2.53 RenderObject()

```
void RenderingEngine::RenderScene::RenderObject (
    unsigned int indexCount,
    unsigned int locationFirstIndex,
    int indexOfFirstVertex,
    D3D_PRIMITIVE_TOPOLOGY primitive )
```

Renders an object with the specified draw arguments.

Call in between a BeforeRenderObjects function and a AfterRenderObjects function.

Ex.

```
BeforeRenderObjects()
RenderObject()
RenderObject()
AfterRenderObjects()
```

Parameters

in	<i>indexCount</i>	The number of indices used to connect the vertices of the objects.
in	<i>locationFirstIndex</i>	The location of the first index of the object in an index buffer.
in	<i>indexOfFirstVertex</i>	The index of the first vertex of the object in a vertex buffer.
in	<i>primitive</i>	The primitive used to render the object.

5.11.2.54 RenderText()

```
void RenderingEngine::RenderScene::RenderText (
    const vec4 & textLocation,
    const Color & textColor,
    float textSize,
    const std::wstring & textString,
    DWRITE_PARAGRAPH_ALIGNMENT alignment = DWRITE_PARAGRAPH_ALIGNMENT_CENTER )
```

Draws the [Text](#) object mapped to the specified *textKey*. Call in between a BeforeRenderText function and a AfterRenderText function.

.

Ex.

```
BeforeRenderText()
RenderText()
RenderText()
AfterRenderText()
```

Throws an out_of_range exception if the textKey is not mapped to a [Text](#) object.

Parameters

in	<i>textLocation</i>	The location of the text. The first 2 values are the top left corner and last two values are bottom right corner.
in	<i>textColor</i>	The color of the text.
in	<i>textSize</i>	The size of the size.
in	<i>textString</i>	The text to render. Generated by Doxygen
in	<i>alignment</i>	Where you want the text to start at in the rectangle.

5.11.2.55 Resize()

```
void RenderingEngine::RenderScene::Resize (
    unsigned int width,
    unsigned int height,
    HWND windowHandle,
    bool isMSAAEnabled = false,
    bool isTextEnabled = false )
```

Resizes the window-dependent resources when the window gets resized.

Parameters

in	<i>width</i>	The width of a window.
in	<i>height</i>	The height of a window.
in	<i>handle</i>	A handle to a window.
	<i>[in,optional]</i>	isMSAAEnabled Pass in true if MSAA enabled, false otherwise.
	<i>[in,optional]</i>	isTextEnabled Pass in true if text enabled, false otherwise.

5.11.2.56 SetConstants()

```
void RenderingEngine::RenderScene::SetConstants (
    unsigned int rootParameterIndex,
    unsigned int numValues,
    void * data,
    unsigned int index )
```

Links an array of 32-bit values to the pipeline.

Parameters

in	<i>rootParameterIndex</i>	The index of the root parameter in the root signature that has the register the texture will be stored in.
in	<i>numValues</i>	The number of 32-bit values.
in	<i>data</i>	Pointer to an array of 32-bit values.
in	<i>index</i>	The index of the the first 32-bit value in the hlsl constant buffer.

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

- RenderScene.h

5.12 RenderingEngine::RenderTargetBuffer Class Reference

A wrapper for render target buffer resources. Uses DirectD 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

```
#include "Buffer.h"
```

Public Member Functions

- **RenderTargetBuffer** (const [RenderTargetBuffer](#) &)=delete
- **RenderTargetBuffer** & **operator=** (const [RenderTargetBuffer](#) &)=delete
- **RenderTargetBuffer** ()
Creates a render target buffer object. No memory is allocated. Called the [CreateRenderTargetBufferAndView\(\)](#) function to allocate memory for the buffer.
- **RenderTargetBuffer** (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &rtvHeap, unsigned int index, unsigned int rtvSize, unsigned int width, unsigned int height, DXGI_FORMAT format=DXGI_FORMAT_R8G8B8A8_UNORM, unsigned int sampleCount=1)
Creates the render target buffer and view.
- DXGI_FORMAT **GetRenderTargetFormat** () const
Returns the format of the render target buffer.
- Microsoft::WRL::ComPtr< ID3D12Resource > & **GetRenderTargetBuffer** ()
Returns a reference to the render target buffer.
- const Microsoft::WRL::ComPtr< ID3D12Resource > & **GetRenderTargetBuffer** () const
Returns a constant reference to the render target buffer.
- void **CreateRenderTargetBufferAndView** (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &rtvHeap, unsigned int index, unsigned int rtvSize, unsigned int width, unsigned int height, DXGI_FORMAT format=DXGI_FORMAT_R8G8B8A8_UNORM, unsigned int sampleCount=1)
Creates the render target buffer and view.
- void **ReleaseBuffer** ()
Frees the memory of the buffer.
- void **ClearRenderTargetBuffer** (const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &rtvHeap, unsigned int indexOfView, unsigned int rtvSize, const float *clearValue)
Clears the render target buffer with the specified clear value.

5.12.1 Detailed Description

A wrapper for render target buffer resources. Uses DirectD 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 **RenderTargetBuffer**() [1/2]

```
RenderingEngine::RenderTargetBuffer::RenderTargetBuffer ( )
```

Creates a render target buffer object. No memory is allocated. Called the [CreateRenderTargetBufferAndView\(\)](#) function to allocate memory for the buffer.

5.12.2.2 RenderTargetBuffer() [2/2]

```
RenderingEngine::RenderTargetBuffer::RenderTargetBuffer (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
    const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & rtvHeap,
    unsigned int index,
    unsigned int rtvSize,
    unsigned int width,
    unsigned int height,
    DXGI_FORMAT format = DXGI_FORMAT_R8G8B8A8_UNORM,
    unsigned int sampleCount = 1 )
```

Creates the render target buffer and view.

Parameters

in	<i>device</i>	A Direct3D 12 device.
in	<i>rtvHeap</i>	A descriptor heap for storing render target descriptors.
in	<i>index</i>	The index of where to store the created descriptor in the descriptor heap.
in	<i>rtvSize</i>	The size of a render target descriptor.
in	<i>width</i>	The width of the render target buffer.
in	<i>height</i>	The height of the render target buffer.
in	<i>sampleCount</i>	The sample count of the render target buffer.

5.12.3 Member Function Documentation

5.12.3.1 ClearRenderTargetBuffer()

```
void RenderingEngine::RenderTargetBuffer::ClearRenderTargetBuffer (
    const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & commandList,
    const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & rtvHeap,
    unsigned int indexOfView,
    unsigned int rtvSize,
    const float * clearValue )
```

Clears the render target buffer with the specified clear value.

Parameters

in	<i>commadList</i>	A Direct3D 12 graphics command list.
in	<i>rtvHeap</i>	A render target descriptor heap.
in	<i>indexOfView</i>	The index of where the render target descriptor of the render target buffer is stored in the descriptor heap.
in	<i>rtvSize</i>	The size of a render target descriptor.
in	<i>clearValue</i>	The RGBA values of what to set every element in the render target buffer to.

5.12.3.2 CreateRenderTargetBufferAndView()

```
void RenderingEngine::RenderTargetBuffer::CreateRenderTargetBufferAndView (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
    const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & rtvHeap,
    unsigned int index,
    unsigned int rtvSize,
    unsigned int width,
    unsigned int height,
    DXGI_FORMAT format = DXGI_FORMAT_R8G8B8A8_UNORM,
    unsigned int sampleCount = 1 )
```

Creates the render target buffer and view.

Parameters

in	<i>device</i>	A Direct3D 12 device.
in	<i>rtvHeap</i>	A descriptor heap for storing render target descriptors.
in	<i>index</i>	The index of where to store the created descriptor in the descriptor heap.
in	<i>rtvSize</i>	The size of a render target descriptor.
in	<i>width</i>	The width of the render target buffer.
in	<i>height</i>	The height of the render target buffer.
in	<i>sampleCount</i>	The sample count of the render target buffer.

5.12.3.3 GetRenderTargetBuffer() [1/2]

```
Microsoft::WRL::ComPtr< ID3D12Resource > & RenderingEngine::RenderTargetBuffer::GetRender↵
TargetBuffer ( )
```

Returns a reference to the render target buffer.

5.12.3.4 GetRenderTargetBuffer() [2/2]

```
const Microsoft::WRL::ComPtr< ID3D12Resource > & RenderingEngine::RenderTargetBuffer::Get↵
RenderTargetBuffer ( ) const
```

Returns a constant reference to the render target buffer.

5.12.3.5 GetRenderTargetFormat()

```
DXGI_FORMAT RenderingEngine::RenderTargetBuffer::GetRenderTargetFormat ( ) const
```

Returns the format of the render target buffer.

5.12.3.6 ReleaseBuffer()

```
void RenderingEngine::RenderTargetBuffer::ReleaseBuffer ( )
```

Frees the memory of the buffer.

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

- Buffer.h

5.13 RenderingEngine::StaticBuffer Class Reference

This class stores data in a Direct3D 12 default buffer. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

```
#include "Buffer.h"
```

Public Member Functions

- **StaticBuffer** (const [StaticBuffer](#) &)=delete
- **StaticBuffer & operator=** (const [StaticBuffer](#) &)=delete
- **StaticBuffer** ()
Creates a static buffer object. No memory is allocated for the buffer. Call one of the [CreateStaticBuffer\(\)](#) functions to allocate memory for the buffer and store data in the buffer.
- **StaticBuffer** (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const void *data, unsigned int numBytes, unsigned int stride)
Creates a static vertex buffer and stores all of the specified data in the buffer.
- **StaticBuffer** (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const void *data, unsigned int numBytes, DXGI_FORMAT format)
Creates a static index buffer and stores all of the specified data in the buffer.
- **StaticBuffer** (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const wchar_t *filename)
Creates a static texture buffer and stores all of the data from the file in the buffer.
- void **CreateStaticBuffer** (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const void *data, unsigned int numBytes, unsigned int stride)
Creates a static vertex buffer and stores all of the specified data in the buffer.
- void **CreateStaticBuffer** (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const void *data, unsigned int numBytes, DXGI_FORMAT format)
Creates a static index buffer and stores all of the specified data in the buffer.
- void **CreateStaticBuffer** (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const wchar_t *filename)
Creates a static texture buffer and stores all of the data from the file in the buffer.
- const D3D12_VERTEX_BUFFER_VIEW **GetVertexBufferView** () const
Returns the vertex buffer view of the static buffer.
- const D3D12_INDEX_BUFFER_VIEW **GetIndexBufferView** () const
Returns the index buffer view of the static buffers.

- void [CreateTexture2DView](#) (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &srvHeap, unsigned int srvSize, unsigned int index)
Creates a 2D texture view and stores it in the specified heap.
- void [CreateTexture2DMSView](#) (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &srvHeap, unsigned int srvSize, unsigned int index)
Creates a multi-sampled 2D texture view and stores it in the specified heap.
- void [ReleaseBuffer](#) ()
Frees the static buffer memory.

5.13.1 Detailed Description

This class stores data in a Direct3D 12 default buffer. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 StaticBuffer() [1/4]

```
RenderingEngine::StaticBuffer::StaticBuffer ( )
```

Creates a static buffer object. No memory is allocated for the buffer. Call one of the [CreateStaticBuffer\(\)](#) functions to allocate memory for the buffer and store data in the buffer.

5.13.2.2 StaticBuffer() [2/4]

```
RenderingEngine::StaticBuffer::StaticBuffer (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
    const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & commandList,
    const void * data,
    unsigned int numBytes,
    unsigned int stride )
```

Creates a static vertex buffer and stores all of the specified data in the buffer.

Parameters

in	<i>device</i>	A Direct3D 12 device.
in	<i>commadList</i>	A Direct3D 12 graphics command list.
in	<i>data</i>	The data to store in the static vertex buffer.
in	<i>numBytes</i>	The number of bytes to store in the static vertex buffer.
in	<i>stride</i>	The number of bytes to get from one element to the next element.

5.13.2.3 StaticBuffer() [3/4]

```
RenderingEngine::StaticBuffer::StaticBuffer (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
    const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & commandList,
    const void * data,
    unsigned int numBytes,
    DXGI_FORMAT format )
```

Creates a static index buffer and stores all of the specified data in the buffer.

Parameters

in	<i>device</i>	A Direct3D 12 device.
in	<i>commadList</i>	A Direct3D 12 graphics command list.
in	<i>data</i>	The data to store in the static index buffer.
in	<i>numBytes</i>	The number of bytes to store in the static index buffer.
in	<i>format</i>	The number of bytes to get from one element to the next element.

5.13.2.4 StaticBuffer() [4/4]

```
RenderingEngine::StaticBuffer::StaticBuffer (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
    const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & commandList,
    const wchar_t * filename )
```

Creates a static texture buffer and stores all of the data from the file in the buffer.

Parameters

in	<i>device</i>	A Direct3D 12 device.
in	<i>commadList</i>	A Direct3D 12 graphics command list.
in	<i>data</i>	The data to store in the static texture buffer.
in	<i>numBytes</i>	The number of bytes to store in the static texture buffer.
in	<i>filename</i>	The name of the texture file.

5.13.3 Member Function Documentation

5.13.3.1 CreateStaticBuffer() [1/3]

```
void RenderingEngine::StaticBuffer::CreateStaticBuffer (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
    const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & commandList,
    const void * data,
```

```

    unsigned int numBytes,
    DXGI_FORMAT format )

```

Creates a static index buffer and stores all of the specified data in the buffer.

Parameters

in	<i>device</i>	A Direct3D 12 device.
in	<i>commadList</i>	A Direct3D 12 graphics command list.
in	<i>data</i>	The data to store in the static index buffer.
in	<i>numBytes</i>	The number of bytes to store in the static index buffer.
in	<i>format</i>	The number of bytes to get from one element to the next element.

5.13.3.2 CreateStaticBuffer() [2/3]

```

void RenderingEngine::StaticBuffer::CreateStaticBuffer (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
    const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & commandList,
    const void * data,
    unsigned int numBytes,
    unsigned int stride )

```

Creates a static vertex buffer and stores all of the specified data in the buffer.

Parameters

in	<i>device</i>	A Direct3D 12 device.
in	<i>commadList</i>	A Direct3D 12 graphics command list.
in	<i>data</i>	The data to store in the static vertex buffer.
in	<i>numBytes</i>	The number of bytes to store in the static vertex buffer.
in	<i>stride</i>	The number of bytes to get from one element to the next element.

5.13.3.3 CreateStaticBuffer() [3/3]

```

void RenderingEngine::StaticBuffer::CreateStaticBuffer (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
    const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & commandList,
    const wchar_t * filename )

```

Creates a static texture buffer and stores all of the data from the file in the buffer.

Parameters

in	<i>device</i>	A Direct3D 12 device.
in	<i>commadList</i>	A Direct3D 12 graphics command list.
in	<i>data</i>	The data to store in the static texture buffer.
in	<i>numBytes</i>	The number of bytes to store in the static texture buffer.
in	<i>filename</i>	The name of the texture file.

5.13.3.4 CreateTexture2DMSView()

```
void RenderingEngine::StaticBuffer::CreateTexture2DMSView (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
    const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & srvHeap,
    unsigned int srvSize,
    unsigned int index )
```

Creates a multi-sampled 2D texture view and stores it in the specified heap.

Parameters

in	<i>device</i>	A Direct3D 12 device.
in	<i>srvHeap</i>	A shader resource view heap.
in	<i>srvSize</i>	The size of a shader resource view.
in	<i>index</i>	The index of where to store the texture view in the shader resource view heap.

5.13.3.5 CreateTexture2DView()

```
void RenderingEngine::StaticBuffer::CreateTexture2DView (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
    const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & srvHeap,
    unsigned int srvSize,
    unsigned int index )
```

Creates a 2D texture view and stores it in the specified heap.

Parameters

in	<i>device</i>	A Direct3D 12 device.
in	<i>srvHeap</i>	A shader resource view heap.
in	<i>srvSize</i>	The size of a shader resource view.
in	<i>index</i>	The index of where to store the texture view in the shader resource view heap.

5.13.3.6 GetIndexBufferView()

```
const D3D12_INDEX_BUFFER_VIEW RenderingEngine::StaticBuffer::GetIndexBufferView ( ) const
```

Returns the index buffer view of the static buffers.

5.13.3.7 GetVertexBufferView()

```
const D3D12_VERTEX_BUFFER_VIEW RenderingEngine::StaticBuffer::GetVertexBufferView ( ) const
```

Returns the vertex buffer view of the static buffer.

5.13.3.8 ReleaseBuffer()

```
void RenderingEngine::StaticBuffer::ReleaseBuffer ( )
```

Frees the static buffer memory.

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

- Buffer.h

5.14 RenderingEngine::SwapChain Class Reference

A wrapper for swap chain resources. Uses DirectD 12 API and DXGI API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

```
#include "SwapChain.h"
```

Public Member Functions

- **SwapChain** (const [SwapChain](#) &)=delete
- **SwapChain** & **operator=** (const [SwapChain](#) &)=delete
- **SwapChain** ()
Creates a swap chain object. Call the [CreateSwapChain\(\)](#) function to create a swap chain.
- **SwapChain** (const Microsoft::WRL::ComPtr< IDXGIFactory4 > &dxgiFactory, const Microsoft::WRL::ComPtr< ID3D12CommandQueue > &commandQueue, HWND windowHandle, DXGI_FORMAT rtFormat=DXGI_FORMAT_R8G8B8A8_UNORM, DXGI_FORMAT dsFormat=DXGI_FORMAT_D24_UNORM_S8_UINT, unsigned int numRenderTargetBuffers=2)
Creates a swap chain.
- void **CreateSwapChain** (const Microsoft::WRL::ComPtr< IDXGIFactory4 > &dxgiFactory, const Microsoft::WRL::ComPtr< ID3D12CommandQueue > &commandQueue, HWND windowHandle, DXGI_FORMAT rtFormat=DXGI_FORMAT_R8G8B8A8_UNORM, DXGI_FORMAT dsFormat=DXGI_FORMAT_D24_UNORM_S8_UINT, unsigned int numRenderTargetBuffers=2)
Creates a swap chain.
- const Microsoft::WRL::ComPtr< ID3D12Resource > & **GetCurrentBackBuffer** () const
Returns a constant reference to the current render target buffer.
- unsigned int **GetNumRenderTargetBuffers** () const
Returns the number of swap chain buffers.
- unsigned int **GetCurrentBackBufferIndex** () const
Returns the current back buffer index.
- DXGI_FORMAT **GetBackBufferFormat** () const
Returns the format of the swap chain.

- DXGI_FORMAT [GetDepthStencilFormat](#) () const
Returns the format of the depth stencil buffer.
- const std::vector< std::unique_ptr< [RenderTargetBuffer](#) > > & [GetRenderTargetBuffers](#) ()
- void [ReleaseBuffers](#) ()
Frees the memory of the render target and depth stencil buffers.
- void [CreateRenderTargetBuffersAndViews](#) (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &rtvHeap, unsigned int index, unsigned int rtvSize, unsigned int width, unsigned int height)
Creates the swap chains render target buffers and views to them.
- void [CreateDepthStencilBufferAndView](#) (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &dsvHeap, unsigned int index, unsigned int dsvSize, unsigned int width, unsigned int height)
Creates the swap chains depth stencil buffer and view to it.
- void [ClearCurrentBackBuffer](#) (const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &rtvHeap, unsigned int indexOfFirstView, unsigned int rtvSize, const float *backBufferClearValue)
Clears the current render target buffer.
- void [ClearDepthStencilBuffer](#) (const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &dsvHeap, unsigned int indexOfView, unsigned int dsvSize, float clearValue)
Clears the swap chains depth stencil buffer with the specified clear value.
- void [Transition](#) (const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, D3D12_RESOURCE_STATES before, D3D12_RESOURCE_STATES after)
Transitions the current render target buffer from the specified before state to the specified after state.
- void [Present](#) ()
Swaps the front and back buffers.

5.14.1 Detailed Description

A wrapper for swap chain resources. Uses DirectD 12 API and DXGI API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

5.14.2 Constructor & Destructor Documentation

5.14.2.1 SwapChain() [1/2]

```
RenderingEngine::SwapChain::SwapChain ( )
```

Creates a swap chain object. Call the [CreateSwapChain\(\)](#) function to create a swap chain.

5.14.2.2 SwapChain() [2/2]

```
RenderingEngine::SwapChain::SwapChain (
    const Microsoft::WRL::ComPtr< IDXGIFactory4 > & dxgiFactory,
    const Microsoft::WRL::ComPtr< ID3D12CommandQueue > & commandQueue,
    HWND windowHandle,
    DXGI_FORMAT rtFormat = DXGI_FORMAT_R8G8B8A8_UNORM,
    DXGI_FORMAT dsFormat = DXGI_FORMAT_D24_UNORM_S8_UINT,
    unsigned int numRenderTargetBuffers = 2 )
```

Creates a swap chain.

Parameters

in	<i>dxgiFactory</i>	A DXGIFactory4 object.
in	<i>A</i>	Direct3D 12 command queue.
in	<i>windowHandle</i>	A handle to a window.
	<i>[in,optional]</i>	rtFormat The format of the render target buffer.
	<i>[in,optional]</i>	dsFormat The format of the depth stencil buffer.
	<i>[in,optional]</i>	numRenderTargetBuffers The number of render target buffers the swap chain has.

5.14.3 Member Function Documentation

5.14.3.1 ClearCurrentBackBuffer()

```
void RenderingEngine::SwapChain::ClearCurrentBackBuffer (
    const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & commandList,
    const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & rtvHeap,
    unsigned int indexOfFirstView,
    unsigned int rtvSize,
    const float * backBufferClearValue )
```

Clears the current render target buffer.

Parameters

in	<i>commadList</i>	A Direct3D 12 graphics command list.
in	<i>rtvHeap</i>	A render target descriptor heap.
in	<i>indexOfFirstView</i>	The index of where the render target descriptor of the first render target buffer is stored in the descriptor heap.
in	<i>rtvSize</i>	The size of a render target descriptor.
in	<i>backBufferClearValue</i>	The RGBA values of what to set every element in the current render target buffer to.

5.14.3.2 ClearDepthStencilBuffer()

```
void RenderingEngine::SwapChain::ClearDepthStencilBuffer (
    const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & commandList,
    const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & dsvHeap,
    unsigned int indexOfView,
    unsigned int dsvSize,
    float clearValue )
```

Clears the swap chains depth stencil buffer with the specified clear value.

Parameters

in	<i>commadList</i>	A Direct3D 12 graphics command list.
in	<i>dsvHeap</i>	A depth stencil descriptor heap.
in	<i>indexOfView</i>	The index of where the depth stencil descriptor of the depth stencil buffer is stored in the descriptor heap.
in	<i>dsvSize</i>	The size of a depth stencil descriptor.
in	<i>clearValue</i>	The value of what to set every element in the depth stencil buffer to.

5.14.3.3 CreateDepthStencilBufferAndView()

```
void RenderingEngine::SwapChain::CreateDepthStencilBufferAndView (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
    const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & dsvHeap,
    unsigned int index,
    unsigned int dsvSize,
    unsigned int width,
    unsigned int height )
```

Creates the swap chains depth stencil buffer and view to it.

Parameters

in	<i>device</i>	A Direct3D 12 device.
in	<i>dsvHeap</i>	A descriptor heap for storing depth stencil descriptors.
in	<i>index</i>	The index of where to store the created descriptor in the descriptor heap.
in	<i>dsvSize</i>	The size of a depth stenicl descriptor.
in	<i>width</i>	The width of the depth stenicl buffer.
in	<i>height</i>	The height of the depth stenicl buffer.

5.14.3.4 CreateRenderTargetBuffersAndViews()

```
void RenderingEngine::SwapChain::CreateRenderTargetBuffersAndViews (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
    const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & rtvHeap,
    unsigned int index,
    unsigned int rtvSize,
    unsigned width,
    unsigned height )
```

Creates the swap chains render target buffers and views to them.

Parameters

in	<i>device</i>	A Direct3D 12 device.
in	<i>rtvHeap</i>	A descriptor heap for storing render target descriptors.

Parameters

in	<i>index</i>	The index of where to store the created descriptor in the descriptor heap.
in	<i>rtvSize</i>	The size of a render target descriptor.
in	<i>width</i>	The width of the render target buffers.
in	<i>height</i>	The height of the render target buffers.

5.14.3.5 CreateSwapChain()

```
void RenderingEngine::SwapChain::CreateSwapChain (
    const Microsoft::WRL::ComPtr< IDXGIFactory4 > & dxgiFactory,
    const Microsoft::WRL::ComPtr< ID3D12CommandQueue > & commandQueue,
    HWND windowHandle,
    DXGI_FORMAT rtFormat = DXGI_FORMAT_R8G8B8A8_UNORM,
    DXGI_FORMAT dsFormat = DXGI_FORMAT_D24_UNORM_S8_UINT,
    unsigned int numRenderTargetBuffers = 2 )
```

Creates a swap chain.

Parameters

in	<i>dxgiFactory</i>	A DXGIFactory4 object.
in	<i>A</i>	Direct3D 12 command queue.
in	<i>windowHandle</i>	A handle to a window.
	<i>[in,optional]</i>	rtFormat The format of the render target buffer.
	<i>[in,optional]</i>	dsFormat The format of the depth stencil buffer.
	<i>[in,optional]</i>	numRenderTargetBuffers The number of render target buffers the swap chain has.

5.14.3.6 GetBackBufferFormat()

```
DXGI_FORMAT RenderingEngine::SwapChain::GetBackBufferFormat ( ) const
```

Returns the format of the swap chain.

5.14.3.7 GetCurrentBackBuffer()

```
const Microsoft::WRL::ComPtr< ID3D12Resource > & RenderingEngine::SwapChain::GetCurrentBackBuffer ( ) const
```

Returns a constant reference to the current render target buffer.

5.14.3.8 GetCurrentBackBufferIndex()

```
unsigned int RenderingEngine::SwapChain::GetCurrentBackBufferIndex ( ) const
```

Returns the current back buffer index.

5.14.3.9 GetDepthStencilFormat()

```
DXGI_FORMAT RenderingEngine::SwapChain::GetDepthStencilFormat ( ) const
```

Returns the format of the depth stencil buffer.

5.14.3.10 GetNumRenderTargetBuffers()

```
unsigned int RenderingEngine::SwapChain::GetNumRenderTargetBuffers ( ) const
```

Returns the number of swap chain buffers.

5.14.3.11 Present()

```
void RenderingEngine::SwapChain::Present ( )
```

Swaps the front and back buffers.

5.14.3.12 ReleaseBuffers()

```
void RenderingEngine::SwapChain::ReleaseBuffers ( )
```

Frees the memory of the render target and depth stencil buffers.

5.14.3.13 Transition()

```
void RenderingEngine::SwapChain::Transition (
    const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & commandList,
    D3D12_RESOURCE_STATES before,
    D3D12_RESOURCE_STATES after )
```

Transitions the current render target buffer from the specified *before* state to the specified *after* state.

Parameters

in	commandList	A Direct3D 12 graphics command list.
----	-------------	--------------------------------------

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

- SwapChain.h

5.15 RenderingEngine::Text Class Reference

This class is used to help render text. Stores the location of the text, the text string, text size and the color of the text.

```
#include "Text.h"
```

Public Member Functions

- [Text](#) (const vec4 &textLocation, const std::wstring &textString, float textSize, const [Color](#) &textColor)
Constructs a [Text](#) object.
- const vec4 & [GetTextLocation](#) () const
Returns a constant reference to the text location.
- const std::wstring & [GetTextString](#) () const
Returns a constant reference to the text string.
- float [GetTextSize](#) () const
Returns the text size.
- const [Color](#) & [GetTextColor](#) () const
Returns a constant reference to the text color.
- void [SetTextSize](#) (float textSize)
Changes the text size to the specified textSize.
- void [SetTextColor](#) (const [Color](#) &textColor)
Changes the text color to the specified textColor.
- void [SetTextString](#) (const std::wstring &textString)
Changes the text string to the specified textString.
- void [SetTextLocation](#) (const vec4 &textLocation)
Changes the text location to the specified textLocation.

5.15.1 Detailed Description

This class is used to help render text. Stores the location of the text, the text string, text size and the color of the text.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 Text()

```
RenderingEngine::Text::Text (
    const vec4 & textLocation,
    const std::wstring & textString,
    float textSize,
    const Color & textColor )
```

Constructs a [Text](#) object.

For text location the first two values in the vector is the top-left location of the rectangle and the last two values are the bottom-right location of the rectangle.

Parameters

in	<i>textLocation</i>	The location of the text on the window.
in	<i>textString</i>	The text to render.
in	<i>textSize</i>	How big the text is.
in	<i>textColor</i>	The color of the text.

5.15.3 Member Function Documentation

5.15.3.1 GetTextColor()

```
const Color & RenderingEngine::Text::GetTextColor ( ) const
```

Returns a constant reference to the text color.

5.15.3.2 GetTextLocation()

```
const vec4 & RenderingEngine::Text::GetTextLocation ( ) const
```

Returns a constant reference to the text location.

5.15.3.3 GetTextSize()

```
float RenderingEngine::Text::GetTextSize ( ) const
```

Returns the text size.

5.15.3.4 GetTextString()

```
const std::wstring & RenderingEngine::Text::GetTextString ( ) const
```

Returns a constant reference to the text string.

5.15.3.5 SetTextColor()

```
void RenderingEngine::Text::SetTextColor (
    const Color & textColor )
```

Changes the text color to the specified *textColor*.

5.15.3.6 SetTextLocation()

```
void RenderingEngine::Text::SetTextLocation (
    const vec4 & textLocation )
```

Changes the text location to the specified *textLocation*.

5.15.3.7 SetTextSize()

```
void RenderingEngine::Text::SetTextSize (
    float textSize )
```

Changes the text size to the specified *textSize*.

5.15.3.8 SetTextString()

```
void RenderingEngine::Text::SetTextString (
    const std::wstring & textString )
```

Changes the text string to the specified *textString*.

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

- Text.h

5.16 RenderingEngine::TextResources Class Reference

A wrapper for resources that are needed to render text using Direct3D 11on12, Direct2D and DirectWrite.

```
#include "TextResources.h"
```

Public Member Functions

- [TextResources](#) (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12CommandQueue > &commandQueue, unsigned int numSwapChainBuffers)
Initializes the text resources.
- const Microsoft::WRL::ComPtr< ID2D1DeviceContext > & [GetDirect2DDeviceContext](#) () const
Returns a constant reference to the direct 2D device context.
- const Microsoft::WRL::ComPtr< IDWriteFactory > & [GetDirectWriteFactory](#) () const
Returns a constant reference to the direct direct write factory.
- void [ResetBuffers](#) ()
Resets the text buffers.
- void [ResizeBuffers](#) (const std::vector< std::unique_ptr< [RenderTargetBuffer](#) > > &renderTargetBuffers, HWND windowHandle)
Resizes the buffers.
- void [BeforeRenderText](#) (unsigned int currentBackBuffer)
Prepares to render text.
- void [AfterRenderText](#) (unsigned int currentBackBuffer)
Executes text commands.

5.16.1 Detailed Description

A wrapper for resources that are needed to render text using Direct3D 11on12, Direct2D and DirectWrite.

5.16.2 Constructor & Destructor Documentation

5.16.2.1 TextResources()

```
RenderingEngine::TextResources::TextResources (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
    const Microsoft::WRL::ComPtr< ID3D12CommandQueue > & commandQueue,
    unsigned int numSwapChainBuffers )
```

Initializes the text resources.

Parameters

in	<i>device</i>	A Direct3D 12 device.
in	<i>commandQueue</i>	A Direct3D 12 command queue.
in	<i>numSwapChainBuffers</i>	The number of swap chain render target buffers.

5.16.3 Member Function Documentation

5.16.3.1 AfterRenderText()

```
void RenderingEngine::TextResources::AfterRenderText (
    unsigned int currentBackBuffer )
```

Executes text commands.

Parameters

in	<i>currentBackBuffer</i>	The index of the current render target buffer.
----	--------------------------	--

5.16.3.2 BeforeRenderText()

```
void RenderingEngine::TextResources::BeforeRenderText (
    unsigned int currentBackBuffer )
```

Prepares to render text.

Parameters

in	<i>currentBackBuffer</i>	The index of the current render target buffer.
----	--------------------------	--

5.16.3.3 GetDirect2DDeviceContext()

```
const Microsoft::WRL::ComPtr< ID2D1DeviceContext > & RenderingEngine::TextResources::Get↔
Direct2DDeviceContext ( ) const
```

Returns a constant reference to the direct 2D device context.

5.16.3.4 GetDirectWriteFactory()

```
const Microsoft::WRL::ComPtr< IDWriteFactory > & RenderingEngine::TextResources::GetDirect↔
WriteFactory ( ) const
```

Returns a constant reference to the direct direct write factory.

5.16.3.5 ResetBuffers()

```
void RenderingEngine::TextResources::ResetBuffers ( )
```

Resets the text buffers.

5.16.3.6 ResizeBuffers()

```
void RenderingEngine::TextResources::ResizeBuffers (
    const std::vector< std::unique_ptr< RenderTargetBuffer > > & renderTarget↵
    Buffers,
    HWND windowHandle )
```

Resizes the buffers.

Parameters

in	<i>renderTargetBuffers</i>	An array of render target buffers.
in	<i>windowHandle</i>	A handle to a window.

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

- TextResources.h

5.17 RenderingEngine::Time Struct Reference

A struct that holds the properties for time.

```
#include "GameTime.h"
```

Public Attributes

- `__int64 previousTime = 0`
- `__int64 currentTime = 0`
- `double deltaTime = 0`
- `double secondsPerCount = 0.0`
- `bool stopped = false`

5.17.1 Detailed Description

A struct that holds the properties for time.

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

- GameTime.h

5.18 RenderingEngine::Window Struct Reference

The window struct is used to make a [Window](#) using Win32 API.

```
#include "Window.h"
```

Public Attributes

- HWND **windowHandle**
- WNDCLASSEX **windowClass**

5.18.1 Detailed Description

The window struct is used to make a [Window](#) using Win32 API.

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

- Window.h

Chapter 6

File Documentation

6.1 Buffer.h

```
1 #pragma once
2
3 #include <wrl.h>
4 #include <d3d12.h>
5
6 namespace RenderingEngine
7 {
8     enum BufferTypes { VERTEX_BUFFER, INDEX_BUFFER, CONSTANT_BUFFER, TEXTURE_BUFFER };
9     enum TextureTypes { TEX2D, TEX2D_MS };
10
11     class RenderTargetBuffer
12     {
13     public:
14
15         //No copying
16         RenderTargetBuffer(const RenderTargetBuffer&) = delete;
17         RenderTargetBuffer& operator=(const RenderTargetBuffer&) = delete;
18
19         RenderTargetBuffer();
20
21         RenderTargetBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
22             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int index, unsigned int
23             rtvSize,
24             unsigned int width, unsigned int height, DXGI_FORMAT format = DXGI_FORMAT_R8G8B8A8_UNORM,
25             unsigned int sampleCount = 1);
26
27         DXGI_FORMAT GetRenderTargetFormat() const;
28
29         Microsoft::WRL::ComPtr<ID3D12Resource>& GetRenderTargetBuffer();
30
31         const Microsoft::WRL::ComPtr<ID3D12Resource>& GetRenderTargetBuffer() const;
32
33         void CreateRenderTargetBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
34             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int index, unsigned int
35             rtvSize,
36             unsigned int width, unsigned int height, DXGI_FORMAT format = DXGI_FORMAT_R8G8B8A8_UNORM,
37             unsigned int sampleCount = 1);
38
39         void ReleaseBuffer();
40
41         void ClearRenderTargetBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
42             commandList,
43             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int index, unsigned int
44             rtvSize,
45             const float* clearValue);
46
47     private:
48         Microsoft::WRL::ComPtr<ID3D12Resource> mRenderTargetBuffer;
49     };
50
51     class DepthStencilBuffer
52     {
53     public:
54
55         //No copying
56         DepthStencilBuffer(const DepthStencilBuffer&) = delete;
57         DepthStencilBuffer& operator=(const DepthStencilBuffer&) = delete;
```

```

108     DepthStencilBuffer();
109
120     DepthStencilBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
121         const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int index, unsigned
122         int dsvSize, unsigned int width, unsigned int height, DXGI_FORMAT format = DXGI_FORMAT_D24_UNORM_S8_UINT,
123         unsigned int sampleCount = 1);
124
126     DXGI_FORMAT GetDepthStencilFormat() const;
127
138     void CreateDepthStencilBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
139         const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int index, unsigned
140         int dsvSize, unsigned int width, unsigned int height, DXGI_FORMAT format = DXGI_FORMAT_D24_UNORM_S8_UINT,
141         unsigned int sampleCount = 1);
142
144     void ReleaseBuffer();
145
156     void ClearDepthStencilBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
157         commandList, const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int index,
158         unsigned int dsvSize, float clearValue);
159
160     private:
161         Microsoft::WRL::ComPtr<ID3D12Resource> mDepthStencilBuffer;
162     };
163
164     class StaticBuffer
165     {
166     public:
167
168         //No copying
169         StaticBuffer(const StaticBuffer&) = delete;
170         StaticBuffer& operator=(const StaticBuffer&) = delete;
171
172         StaticBuffer();
173
174         StaticBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
175             const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const void* data,
176             unsigned int numBytes, unsigned int stride);
177
178         StaticBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
179             const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const void* data,
180             unsigned int numBytes, DXGI_FORMAT format);
181
182         StaticBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
183             const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const wchar_t*
184             filename);
185
186         void CreateStaticBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
187             const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const void* data,
188             unsigned int numBytes, unsigned int stride);
189
190         void CreateStaticBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
191             const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const void* data,
192             unsigned int numBytes, DXGI_FORMAT format);
193
194         void CreateStaticBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
195             const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const wchar_t*
196             filename);
197
198         const D3D12_VERTEX_BUFFER_VIEW GetVertexBufferView() const;
199
200         const D3D12_INDEX_BUFFER_VIEW GetIndexBufferView() const;
201
202         void CreateTexture2DView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
203             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& srvHeap, unsigned int srvSize, unsigned
204             int index);
205
206         void CreateTexture2DMSView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
207             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& srvHeap, unsigned int srvSize, unsigned
208             int index);
209
210         void ReleaseBuffer();
211
212     private:
213         Microsoft::WRL::ComPtr<ID3D12Resource> mStaticDefaultBuffer;
214         Microsoft::WRL::ComPtr<ID3D12Resource> mStaticUploadBuffer;
215
216         union
217         {
218             unsigned int mStride;
219             DXGI_FORMAT mFormat;
220         };
221     };
222
223     };

```

```

324
329     class DynamicBuffer
330     {
331     public:
332
333         //No copying
334         DynamicBuffer(const DynamicBuffer&) = delete;
335         DynamicBuffer& operator=(const DynamicBuffer&) = delete;
336
337         DynamicBuffer();
338
339         DynamicBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device, unsigned int numOfBytes,
340             unsigned int stride);
341
342         DynamicBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device, unsigned int numOfBytes,
343             DXGI_FORMAT format);
344
345         ~DynamicBuffer();
346
347         void CreateDynamicBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device, unsigned int
348             numOfBytes, unsigned int stride);
349
350         void CreateDynamicBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device, unsigned int
351             numOfBytes, DXGI_FORMAT format);
352
353         const D3D12_GPU_VIRTUAL_ADDRESS GetGPUAddress(unsigned int index) const;
354
355         void CreateConstantBufferView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
356             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& cbvHeap, unsigned int cbvSize, unsigned
357             int cbvHeapIndex,
358             unsigned int cBufferIndex);
359
360         const D3D12_VERTEX_BUFFER_VIEW GetVertexBufferView();
361
362         const D3D12_INDEX_BUFFER_VIEW GetIndexBufferView();
363
364         void CopyData(unsigned int index, const void* data, unsigned long long numOfBytes);
365
366         void ReleaseBuffer();
367
368     private:
369         Microsoft::WRL::ComPtr<ID3D12Resource> mDynamicBuffer;
370         BYTE* mMappedData{ nullptr };
371
372         union
373         {
374             {
375                 UINT mStride;
376                 DXGI_FORMAT mFormat;
377             };
378         };
379     };
380 }

```

6.2 Camera.h

```

1  #pragma once
2
3
4  #include "MathEngine.h"
5  #include <Windows.h>
6
7
8  namespace RenderingEngine
9  {
10     struct Camera
11     {
12     {
13         //camera position in world coordinates
14         vec3 position;
15
16         //x-axis of the camera coordinate system
17         vec3 x;
18
19         //y-axis of the camera coordinate system
20         vec3 y;
21
22         //z-axis of the camera coordinate system
23         vec3 z;
24
25         //stores the world to camera transform
26         mat4 viewMatrix;
27
28         //the velocities of the camera.
29         float linearSpeed = 0.0f;
30         float angularSpeed = 0.0f;
31     }
32 }

```

```

34     };
35
46     void SetProperty(Camera& camera, vec3 position, vec3 x, vec3 y, vec3 z, float linearSpeed, float
angularSpeed);
47
55     void LookAt(Camera& camera, vec3 position, vec3 target, vec3 up);
56
59     void UpdateViewMatrix(Camera& camera);
60
63     void Left(Camera& camera, float dt);
64
70     void Right(Camera& camera, float dt);
71
77     void Forward(Camera& camera, float dt);
78
84     void Backward(Camera& camera, float dt);
85
91     void Up(Camera& camera, float dt);
92
98     void Down(Camera& camera, float dt);
99
105    void RotateCameraLeftRight(Camera& camera, float xDiff);
106
112    void RotateCameraUpDown(Camera& camera, float yDiff);
113 }

```

6.3 Color.h

```

1  #pragma once
2
3  #include "MathEngine.h"
4
5  namespace RenderingEngine
6  {
12     class Color
13     {
14     public:
15
18         Color(float r = 0.0f, float g = 0.0f, float b = 0.0f, float a = 1.0f);
19
22         Color(const vec4& color);
23
26         const vec4& GetColor() const;
27
30         float GetRed() const;
31
34         float GetGreen() const;
35
38         float GetBlue() const;
39
42         float GetAlpha() const;
43
46         void SetColor(const vec4& color);
47
50         void SetRed(float r);
51
54         void SetGreen(float g);
55
58         void SetBlue(float b);
59
62         void SetAlpha(float a);
63
68         Color& operator+=(const Color& c);
69
74         Color& operator-=(const Color& c);
75
81         Color& operator*=(float k);
82
88         Color& operator*=(const Color& c);
89
90     private:
91         vec4 mColor;
92     };
93
98     Color operator+(const Color& c1, const Color& c2);
99
104     Color operator-(const Color& c1, const Color& c2);
105
111     Color operator*(const Color& c, float k);
112
119     Color operator*(float k, const Color& c);
120
125     Color operator*(const Color& c1, const Color& c2);
126 }

```

6.4 DDSTextureLoader.h

```

1 //-----
2 // File: DDSTextureLoader.h
3 //
4 // Functions for loading a DDS texture and creating a Direct3D 11 runtime resource for it
5 //
6 // Note these functions are useful as a light-weight runtime loader for DDS files. For
7 // a full-featured DDS file reader, writer, and texture processing pipeline see
8 // the 'Texconv' sample and the 'DirectXTex' library.
9 //
10 // THIS CODE AND INFORMATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF
11 // ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO
12 // THE IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A
13 // PARTICULAR PURPOSE.
14 //
15 // Copyright (c) Microsoft Corporation. All rights reserved.
16 //
17 // http://go.microsoft.com/fwlink/?LinkId=248926
18 // http://go.microsoft.com/fwlink/?LinkId=248929
19 //-----
20
21 #ifdef _MSC_VER
22 #pragma once
23 #endif
24
25 #include <wrl.h>
26 #include <d3d11_1.h>
27 #include "d3dx12.h"
28
29 #pragma warning(push)
30 #pragma warning(disable : 4005)
31 #include <stdint.h>
32
33 #pragma warning(pop)
34
35 #if defined(_MSC_VER) && (_MSC_VER<1610) && !defined(_In_reads_)
36 #define _In_reads_(exp)
37 #define _Out_writes_(exp)
38 #define _In_reads_bytes_(exp)
39 #define _In_reads_opt_(exp)
40 #define _Outptr_opt_
41 #endif
42
43 #ifndef _Use_decl_annotations_
44 #define _Use_decl_annotations_
45 #endif
46
47 namespace DirectX
48 {
49     enum DDS_ALPHA_MODE
50     {
51         DDS_ALPHA_MODE_UNKNOWN = 0,
52         DDS_ALPHA_MODE_STRAIGHT = 1,
53         DDS_ALPHA_MODE_PREMULTIPLIED = 2,
54         DDS_ALPHA_MODE_OPAQUE = 3,
55         DDS_ALPHA_MODE_CUSTOM = 4,
56     };
57
58     HRESULT CreateDDSTextureFromMemory12(_In_ ID3D12Device* device,
59         _In_ ID3D12GraphicsCommandList* cmdList,
60         _In_reads_bytes_(ddsDataSize) const uint8_t* ddsData,
61         _In_size_t ddsDataSize,
62         _Out_ Microsoft::WRL::ComPtr<ID3D12Resource>& texture,
63         _Out_ Microsoft::WRL::ComPtr<ID3D12Resource>& textureUploadHeap,
64         _In_size_t maxsize = 0,
65         _Out_opt_ DDS_ALPHA_MODE* alphaMode = nullptr
66     );
67
68     HRESULT CreateDDSTextureFromFile12(_In_ ID3D12Device* device,
69         _In_ ID3D12GraphicsCommandList* cmdList,
70         _In_z_ const wchar_t* szFileName,
71         _Out_ Microsoft::WRL::ComPtr<ID3D12Resource>& texture,
72         _Out_ Microsoft::WRL::ComPtr<ID3D12Resource>& textureUploadHeap,
73         _In_size_t maxsize = 0,
74         _Out_opt_ DDS_ALPHA_MODE* alphaMode = nullptr
75     );
76 }

```

6.5 DeviceResources.h

```

1 #pragma once
2

```

```

3 #include <wrl.h>
4 #include <d3dx12.h>
5 #include <dxgil_4.h>
6 #include "SwapChain.h"
7 #include "Multisampling.h"
8 #include "TextResources.h"
9
10 namespace RenderingEngine
11 {
12     class DeviceResources
13     {
14     public:
15
16         //No copying
17         DeviceResources(const DeviceResources&) = delete;
18         DeviceResources& operator=(const DeviceResources&) = delete;
19
20         static const unsigned int NUM_OF_FRAMES{ 3 };
21
22         static DeviceResources& GetInstance(unsigned int width, unsigned int height, HWND windowHandle,
23 bool isMSAAEnabled, bool isTextEnabled);
24
25         ~DeviceResources();
26
27         const Microsoft::WRL::ComPtr<ID3D12Device>& GetDevice() const;
28
29         const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& GetCommandList() const;
30
31         DXGI_FORMAT GetBackBufferFormat() const;
32
33         DXGI_FORMAT GetDepthStencilFormat() const;
34
35         unsigned int GetCBVSRVUAVSize() const;
36
37         unsigned int GetCurrentFrame() const;
38
39         const TextResources& GetTextResources() const;
40
41         void UpdateCurrentFrameFenceValue();
42
43         void FlushCommandQueue();
44
45         void WaitForGPU() const;
46
47         void Signal();
48
49         void Resize(int width, int height, const HWND& handle, bool isMSAAEnabled, bool isTextEnabled);
50
51         void RTBufferTransition(bool isMSAAEnabled, bool isTextEnabled);
52
53         void BeforeTextDraw();
54
55         void AfterTextDraw();
56
57         void Execute() const;
58
59         void Present();
60
61         /*@brief Calls the necessary functions to let the user draw their objects.
62 *
63 * @param[in] isMSAAEnabled Pass in true if MSAA enabled, false otherwise.
64 */
65         void Draw(bool isMSAAEnabled);
66
67         void NextFrame();
68
69     private:
70
71         DeviceResources(unsigned int width, unsigned int height, HWND windowHandle,
72 bool isMSAAEnabled, bool isTextEnabled);
73
74         unsigned int mCurrentFrameIndex;
75
76         Microsoft::WRL::ComPtr<ID3D12Device> mDirect3DDevice;
77
78         Microsoft::WRL::ComPtr<IDXGIFactory4> mDXGIFactory;
79
80         Microsoft::WRL::ComPtr<ID3D12Fence> mFence;
81         UINT64 mFenceValue;
82         UINT64 mCurrentFrameFenceValue[NUM_OF_FRAMES];
83
84         Microsoft::WRL::ComPtr<ID3D12CommandQueue> mCommandQueue;
85         Microsoft::WRL::ComPtr<ID3D12CommandAllocator> mCommandAllocators[NUM_OF_FRAMES];
86         Microsoft::WRL::ComPtr<ID3D12CommandAllocator> mDirectCommandAllocator;
87         Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList> mCommandList;
88
89         UINT mRTVSize;

```

```

174         UINT mDSVSize;
175         UINT mCBVSize;
176
177         Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mRTVHeap;
178         Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mDSVHeap;
179
180         SwapChain mSwapChain;
181
182         MultiSampling mMultiSampling;
183
184         D3D12_VIEWPORT mViewport{};
185         D3D12_RECT mScissor{};
186
187         TextResources mTextResources;
188
189         //Call all of these functions to initialize Direct3D
190         void mEnableDebugLayer();
191         void mCreateDirect3DDevice();
192         void mCreateDXGIFactory();
193         void mCreateFence();
194         void mQueryDescriptorSizes();
195         void mCreateRTVHeap();
196         void mCreateDSVHeap();
197         void mCreateCommandObjects();
198     };
199 }

```

6.6 Direct3DLink.h

```

1 #pragma once
2
3 //Link necessary libraries.
4 #pragma comment(lib, "D3D12.lib")
5 #pragma comment(lib, "dxgi.lib")
6 #pragma comment(lib, "dxguid.lib")
7 #pragma comment(lib, "d3dcompiler.lib")
8 #pragma comment(lib, "D3D11.lib")
9 #pragma comment(lib, "D2D1.lib")
10 #pragma comment(lib, "DWrite.lib")

```

6.7 DirectXException.h

```

1 #pragma once
2
3 #include <wrl.h>
4 #include <dxgidebug.h>
5 #include <comdef.h>
6 #include <string>
7 #include <sstream>
8 #include <vector>
9
14 inline std::wstring AnsiToWString(const std::string& str)
15 {
16     WCHAR buffer[2048];
17     MultiByteToWideChar(CP_ACP, 0, str.c_str(), -1, buffer, 1024);
18     return std::wstring(buffer);
19 }
20
24 class DirectXException
25 {
26 public:
27
35     DirectXException(HRESULT hr, const std::wstring& functionName, const std::wstring& fileName, int
        lineNumber);
36
39     std::wstring ErrorMessage() const;
40
41 private:
42     HRESULT errorCode;
43     std::wstring functionName;
44     std::wstring fileName;
45     int lineNumber;
46     Microsoft::WRL::ComPtr<IDXGIInfoQueue> mInfoQueue;
47 };
48
51 #ifndef ThrowIfFailed
52 #define ThrowIfFailed(x)
53 {
54     HRESULT hr = (x);

```

```

55 std::wstring filename(AnsiToWString(__FILE__));
56 if (FAILED(hr)) { throw DirectXException(hr, L#x, filename, __LINE__); }
57 }
58 #endif
59
60
63 inline void CreateInfoQueue(Microsoft::WRL::ComPtr<IDXGIInfoQueue>& infoQueue)
64 {
65     #if defined(_DEBUG) || defined(DEBUG)
66         //define function signature
67         typedef HRESULT(WINAPI* dxgiDebugInterface)(REFIID, void*);
68
69         //Get a handle to the dll file
70         HMODULE dxgiDebugHandle;
71         GetModuleHandleEx(GET_MODULE_HANDLE_EX_FLAG_UNCHANGED_REFCOUNT, L"Dxgidebug.dll", &dxgiDebugHandle);
72
73         //get the address of the function DXGIGetDebugInterface in the dll file
74         dxgiDebugInterface DXGIGetDebugInterface = (dxgiDebugInterface)GetProcAddress(dxgiDebugHandle,
75 "DXGIGetDebugInterface");
76         if (DXGIGetDebugInterface == nullptr)
77         {
78             exit(-1);
79         }
80         //create a DXGIInfoQueue object.
81         DXGIGetDebugInterface(IID_PPV_ARGS(&infoQueue));
82 #endif
83 }
84
85
86
89 inline std::wstring ErrorMessage(HRESULT errorCode, const std::wstring& functionName, const std::wstring&
    filename, int lineNumber,
    const Microsoft::WRL::ComPtr<IDXGIInfoQueue>& infoQueue)
90 {
91     //the _com_error class lets us retrieve the error message associated with the HRESULT error code
92     _com_error error(errorCode);
93     std::wstring msg = error.ErrorMessage();
94
95     //Get the hex value of the error code
96     std::stringstream ss;
97     ss << std::hex << errorCode;
98     std::wstring hrHex{ AnsiToWString(ss.str()) };
99
100     std::wstring eCode(std::to_wstring(errorCode));
101
102
103     std::wstring errorMessage{ L"File Name: " + filename + L"\n\n" + L"Function Name: " + functionName
+ L"\n\n" +
104     L"Line Number: " + std::to_wstring(lineNumber) + L"\n\n" + L"Error Code: " + eCode +
105     L"(0x" + hrHex + L")" + L"\n\n" + L"Error Code Description: " + msg };
106
107     std::vector<std::wstring> messages;
108
109     if (infoQueue != nullptr)
110     {
111         //Get the number of messages in the queue.
112         UINT64 numMessages = infoQueue->GetNumStoredMessages(DXGI_DEBUG_ALL);
113
114         for (UINT64 i = 0; i < numMessages; ++i)
115         {
116             //Get the length of the current message.
117             SIZE_T messageLength{ 0 };
118             infoQueue->GetMessage(DXGI_DEBUG_ALL, i, nullptr, &messageLength);
119
120             //Allocate enough memory to store the message.
121             std::unique_ptr<unsigned char[]> bytes = std::make_unique<unsigned char[]>(messageLength);
122             DXGI_INFO_QUEUE_MESSAGE* pMsg = (DXGI_INFO_QUEUE_MESSAGE*)bytes.get();
123
124             //Retrieve the message. It will be stored in pMsg.
125             infoQueue->GetMessage(DXGI_DEBUG_ALL, i, pMsg, &messageLength);
126
127             //Store the message.
128             std::string tempMessage{ pMsg->pDescription };
129             messages.emplace_back(AnsiToWString(tempMessage));
130         }
131     }
132
133     for (int i = 0; i < messages.size(); ++i)
134     {
135         errorMessage += L"\n";
136         errorMessage += messages[i];
137     }
138
139     return errorMessage;
140 }
141 }
142

```



```

145 #ifndef ExitIfFailed
146 #define ExitIfFailed(x)
147 {
148 HRESULT hr = (x);
149 if (FAILED(hr))
150 {
151 Microsoft::WRL::ComPtr<IDXGIInfoQueue> infoQueue;
152 CreateInfoQueue(infoQueue);
153 std::wstring filename(AnsiToWString(__FILE__));
154 std::wstring errMsg = ErrorMessage(hr, L"x", filename, __LINE__, infoQueue);
155 MessageBox(nullptr, errMsg.c_str(), L"DirectX Error", MB_OK);
156     exit(-1);
157 }
158 }
159 #endif

```

6.8 DrawArguments.h

```

1 #pragma once
2
3 #include <string>
4 #include <d3dcommon.h>
5
6 namespace RenderingEngine
7 {
8     struct DrawArguments
9     {
10         unsigned int indexCount = 0;
11         unsigned int locationOfFirstIndex = 0;
12         int indexOfFirstVertex = 0;
13         unsigned int indexOfConstantData = 0;
14         unsigned int rootParameterIndex = 0;
15         std::wstring constantBufferKey = L"";
16         D3D_PRIMITIVE_TOPOLOGY primitive = D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST;
17     };
18 }

```

6.9 GameTime.h

```

1 #pragma once
2
3 #include <Windows.h>
4
5 namespace RenderingEngine
6 {
7     struct Time
8     {
9         __int64 previousTime = 0;
10        __int64 currentTime = 0;
11        double deltaTime = 0;
12        double secondsPerCount = 0.0;
13        bool stopped = false;
14    };
15
16    void InitializeTime(Time& time);
17
18    void Reset(Time& time);
19
20    void Tick(Time& time);
21
22    void Start(Time& time);
23
24    void Stop(Time& time);
25 }

```

6.10 Multisampling.h

```

1 #pragma once
2
3 #include <wrl.h>
4 #include "d3dx12.h"
5 #include "Buffer.h"
6
7 namespace RenderingEngine
8 {

```

```

13     class MultiSampling
14     {
15     public:
16
17         MultiSampling(const MultiSampling&) = delete;
18         MultiSampling& operator=(const MultiSampling&) = delete;
19
20         MultiSampling();
21
22         MultiSampling(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
23             DXGI_FORMAT rtFormat, unsigned int sampleCount);
24
25         void CheckMultiSamplingSupport(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
26             DXGI_FORMAT rtFormat, unsigned int sampleCount);
27
28         const Microsoft::WRL::ComPtr<ID3D12Resource>& GetRenderTargetBuffer();
29
30         DXGI_FORMAT GetRenderTargetFormat();
31
32         DXGI_FORMAT GetDepthStencilFormat();
33
34         void ReleaseBuffers();
35
36         void CreateRenderTargetBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
37             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int
38             indexWhereToStoreView, unsigned int rtvSize,
39             unsigned int width, unsigned int height);
40
41         void CreateDepthStencilBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
42             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int
43             indexWhereToStoreView, unsigned int dsvSize,
44             unsigned int width, unsigned int height);
45
46         void Transition(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList,
47             D3D12_RESOURCE_STATES before, D3D12_RESOURCE_STATES after);
48
49         void ClearRenderTargetBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
50             commandList,
51             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int indexWhereToStoreView,
52             unsigned int rtvSize,
53             const float* clearValue);
54
55         void ClearDepthStencilBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
56             commandList,
57             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int indexWhereToStoreView,
58             unsigned int dsvSize,
59             float clearValue);
60
61     private:
62         RenderTargetBuffer mMSAARenderTargetBuffer;
63         DepthStencilBuffer mMSAADepthStencilBuffer;
64         unsigned int mSampleCount;
65     };
66 }

```

6.11 OrthographicProjection.h

```

1 #pragma once
2
3 #include "MathEngine.h"
4
5 namespace RenderingEngine
6 {
7     struct OrthographicProjection
8     {
9         float width = 0.0f;
10        float height = 0.0f;
11        float znear = 0.0f;
12        float zfar = 0.0f;
13        mat4 projectionMatrix;
14    };
15
16    void SetProperties(OrthographicProjection& ortho, float width, float height, float znear, float
17        zfar);
18
19    void UpdateProjectionMatrix(OrthographicProjection& ortho);
20 }

```

6.12 PerspectiveProjection.h

```

1 #pragma once

```

```

2
3 #include "MathEngine.h"
4
5
6 namespace RenderingEngine
7 {
11     struct PerspectiveProjection
12     {
13         float znear = 0.0f;
14         float zfar = 0.0f;
15         float verticalFov = 0.0f;
16         float aspectRatio = 0.0f;
17         mat4 projectionMatrix;
18     };
19
28     void SetProperties(PerspectiveProjection& perspective, float znear, float zfar, float vFov, float
        aspectRatio);
29
34     void UpdateProjectionMatrix(PerspectiveProjection& perspective);
35 }

```

6.13 RenderScene.h

```

1 #pragma once
2
3
4 #include <d3dcompiler.h>
5 #include <unordered_map>
6 #include "DeviceResources.h"
7 #include "Buffer.h"
8 #include "Color.h"
9 #include <string_view>
10
11 namespace RenderingEngine
12 {
17     class RenderScene
18     {
19     public:
20
21
22         //-----
23         //CONSTRUCTORS
24
25         //No copying
26         RenderScene(const RenderScene&) = delete;
27         RenderScene operator=(const RenderScene&) = delete;
28
29         /*@brief Creates a RenderScene object. Does not create the necessary resources to render a
        scene.
30 * Call the function CreateDeviceResources() to initialize all necessary resources.
31 */
32         RenderScene();
33
34         /*@brief Initializes all necessary resources.
35 *
36 * @param[in] width The width of a window.
37 * @param[in] height The height of a window.
38 * @param[in] windowHandle A handle to a window.
39 * @param[in, optional] isMSAAEnabled Pass in true if you want to have MSAA enabled, false otherwise.
40 * @param[in, optional] isTextEnabled Pass in true if you want to have text enabled, false otherwise.
41 */
42         RenderScene(unsigned int width, unsigned int height, HWND windowHandle,
43             bool isMSAAEnabled = false, bool isTextEnabled = false);
44
45         /*@brief Initializes all necessary resources.
46 *
47 * @param[in] width The width of a window.
48 * @param[in] height The height of a window.
49 * @param[in] windowHandle A handle to a window.
50 * @param[in, optional] isMSAAEnabled Pass in true if you want to have MSAA enabled, false otherwise.
51 * @param[in, optional] isTextEnabled Pass in true if you want to have text enabled, false otherwise.
52 */
53         void CreateDeviceResources(unsigned int width, unsigned int height, HWND windowHandle,
54             bool isMSAAEnabled = false, bool isTextEnabled = false);
55
56         //-----
57
58
59
60
61         //-----

```

```

62     //SHADER FUNCTIONS
63
64     void LoadShader(unsigned int shaderKey, std::wstring_view filename);
65
66     void CompileShader(unsigned int shaderKey, std::wstring_view filename,
67         std::string_view entryPointName, std::string_view target);
68
69     void RemoveShader(unsigned int shaderKey);
70
71
72     void LoadShader(std::wstring_view shaderKey, std::wstring_view filename);
73
74     void CompileShader(std::wstring_view shaderKey, std::wstring_view filename,
75         std::string_view entryPointName, std::string_view target);
76
77     void RemoveShader(std::wstring_view shaderKey);
78
79
80 //-----
81
82
83
84 //-----
85
86     //INPUT ELEMENT DESCRIPTION FUNCTIONS
87
88     void CreateInputElementDescription(unsigned int key, const char* semanticName, unsigned int
89         semanticIndex,
90         DXGI_FORMAT format, unsigned int inputSlot, unsigned int byteOffset,
91         D3D12_INPUT_CLASSIFICATION inputSlotClass = D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA,
92         unsigned int instanceStepRate = 0);
93
94
95     void CreateInputElementDescription(std::wstring_view key, const char* semanticName, unsigned int
96         semanticIndex,
97         DXGI_FORMAT format, unsigned int inputSlot, unsigned int byteOffset,
98         D3D12_INPUT_CLASSIFICATION inputSlotClass = D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA,
99         unsigned int instanceStepRate = 0);
100
101
102 //-----
103
104
105 //-----
106
107     //ROOT PARAMETER FUNCTIONS
108
109     void CreateRootDescriptor(unsigned int rootParameterKey, unsigned int shaderRegister);
110
111     void CreateDescriptorRange(unsigned int descriptorRangeKey,
112         D3D12_DESCRIPTOR_RANGE_TYPE type, unsigned int numDescriptors, unsigned int shaderRegister,
113         unsigned int registerSpace,
114         unsigned int offset);
115
116     void CreateDescriptorTable(unsigned int rootParameterKey, unsigned int descriptorRangeKey);
117
118     void CreateRootConstants(unsigned int rootParameterKey, unsigned int shaderRegister, unsigned
119         int numValues);
120
121
122     void CreateRootDescriptor(std::wstring_view rootParameterKey, unsigned int shaderRegister);
123
124     void CreateDescriptorRange(std::wstring_view descriptorRangeKey,
125         D3D12_DESCRIPTOR_RANGE_TYPE type, unsigned int numDescriptors, unsigned int shaderRegister,
126         unsigned int registerSpace,
127         unsigned int offset);
128
129     void CreateDescriptorTable(std::wstring_view rootParameterKey, unsigned int descriptorRangeKey);
130
131     void CreateRootConstants(std::wstring_view rootParameterKey, unsigned int shaderRegister,
132         unsigned int numValues);
133
134
135 //-----
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251

```

```

252
253 //-----
254 //ROOT SIGNATURE FUNCTIONS
255
256 void CreateRootSignature(unsigned int rootSigKey, unsigned int rootParametersKey);
257
258 void CreateRootSignature(unsigned int rootSigKey, unsigned int rootParametersKey,
259 unsigned int staticsSamplerKey);
260
261 void CreateStaticSampler(unsigned int staticSamplerKey, D3D12_FILTER filter,
262 D3D12_TEXTURE_ADDRESS_MODE u, D3D12_TEXTURE_ADDRESS_MODE v, D3D12_TEXTURE_ADDRESS_MODE w,
263 unsigned int shaderRegister);
264
265
266 void CreateRootSignature(std::wstring_view rootSigKey, std::wstring_view rootParametersKey);
267
268 void CreateRootSignature(std::wstring_view rootSigKey, std::wstring_view rootParametersKey,
269 std::wstring_view staticsSamplerKey);
270
271 void CreateStaticSampler(std::wstring_view staticSamplerKey, D3D12_FILTER filter,
272 D3D12_TEXTURE_ADDRESS_MODE u, D3D12_TEXTURE_ADDRESS_MODE v, D3D12_TEXTURE_ADDRESS_MODE w,
273 unsigned int shaderRegister);
274
275
276 //-----
277
278
279
280
281 //-----
282 //PIPELINE STATE OBJECT FUNCTIONS
283
284 void CreatePSO(unsigned int psKey, D3D12_FILL_MODE fillMode, BOOL enableMultisample,
285 unsigned int vsKey, unsigned int psKey, unsigned int inputElementDescriptionsKey,
286 unsigned int rootSigKey,
287 const D3D12_PRIMITIVE_TOPOLOGY_TYPE& primitiveType, UINT sampleCount = 1);
288
289 void LinkPSOAndRootSignature(unsigned int psKey, unsigned int rootSigKey);
290
291
292 void CreatePSO(std::wstring_view psKey, D3D12_FILL_MODE fillMode, BOOL enableMultisample,
293 std::wstring_view vsKey, std::wstring_view psKey, std::wstring_view
294 inputElementDescriptionsKey,
295 std::wstring_view rootSigKey,
296 const D3D12_PRIMITIVE_TOPOLOGY_TYPE& primitiveType, UINT sampleCount = 1);
297
298 void LinkPSOAndRootSignature(std::wstring_view psKey, std::wstring_view rootSigKey);
299
300
301 //-----
302
303
304
305 //-----
306 //STATIC BUFFER FUNCTIONS
307
308 void CreateStaticBuffer(unsigned int staticBufferKey, const void* data, unsigned numBytes,
309 unsigned int stride);
310
311 void CreateStaticBuffer(unsigned int staticBufferKey, const void* data, unsigned numBytes,
312 DXGI_FORMAT format);
313
314 void CreateStaticBuffer(unsigned int staticBufferKey, const wchar_t* filename, unsigned int
315 texType, unsigned int index);
316
317 void LinkStaticBuffer(unsigned int bufferType, unsigned int staticBufferKey);
318
319
320 void CreateStaticBuffer(std::wstring_view staticBufferKey, const void* data, unsigned numBytes,
321 unsigned int stride);
322
323 void CreateStaticBuffer(std::wstring_view staticBufferKey, const void* data, unsigned numBytes,
324 DXGI_FORMAT format);
325
326 void CreateStaticBuffer(std::wstring_view staticBufferKey, const wchar_t* filename, unsigned int
327 texType, unsigned int index);
328
329 void LinkStaticBuffer(unsigned int bufferType, std::wstring_view staticBufferKey);
330
331

```

```

529 //-----
530
531
532
533
534
535 //-----
536 //DYNAMIC BUFFER FUNCTIONS
537
553 void CreateDynamicBuffer(unsigned int dynamicBufferKey, unsigned numBytes, const void* data,
unsigned int stride);
554
570 void CreateDynamicBuffer(unsigned int dynamicBufferKey, unsigned numBytes, const void* data,
DXGI_FORMAT format);
571
590 void LinkDynamicBuffer(unsigned int bufferType, unsigned int dynamicBufferKey, unsigned int
indexConstantData = 0,
unsigned int rootParameterIndex = 0);
591
592
600 void CopyDataIntoDynamicBuffer(unsigned int dynamicBufferKey, unsigned int index, const void*
data, UINT64 numBytes);
601
602
603
619 void CreateDynamicBuffer(std::wstring_view dynamicBufferKey, unsigned numBytes, const void*
data, unsigned int stride);
620
636 void CreateDynamicBuffer(std::wstring_view dynamicBufferKey, unsigned numBytes, const void*
data, DXGI_FORMAT format);
637
656 void LinkDynamicBuffer(unsigned int bufferType, std::wstring_view dynamicBufferKey, unsigned int
indexConstantData = 0,
unsigned int rootParameterIndex = 0);
657
658
666 void CopyDataIntoDynamicBuffer(std::wstring_view dynamicBufferKey, unsigned int index, const
void* data, UINT64 numBytes);
667
668 //-----
669
670
671
672
673
674 //-----
675 //TEXTURE FUNCTIONS
676
680 void CreateTextureViewHeap(unsigned int numDescriptors);
681
684 void LinkTextureViewHeap();
685
691 void LinkTexture(unsigned int rootParameterIndex);
692
700 void LinkTexture(unsigned int rootParameterIndex, unsigned int textureViewIndex);
701
702 //-----
703
704 //-----
705 //RENDER OBJECTS FUNCTONS
706
713 void BeforeRenderObjects(bool isMSAAEnabled = false);
714
733 void RenderObject(unsigned int indexCount, unsigned int locationFirstIndex, int
indexOfFirstVertex,
D3D_PRIMITIVE_TOPOLOGY primitive);
734
735
741 void AfterRenderObjects(bool isMSAAEnabled = false, bool isTextEnabled = false);
742
743 //-----
744
745
746
747
748
749 //-----
750 //RENDER TEXT FUNCTIONS
751
755 void BeforeRenderText();
756
779 void RenderText(const vec4& textLocation, const Color& textColor, float textSize,

```

```

780         const std::wstring& textString, DWRITE_PARAGRAPH_ALIGNMENT alignment =
DWRITE_PARAGRAPH_ALIGNMENT_CENTER);
781
782     void AfterRenderText();
783
784
785 //-----
786
787
788
789
790
791
792
793
794
795 //----- MISCELLANEOUS FUNCTIONS
796
797     void AfterRender();
798
799     void ExecuteAndFlush();
800
801     void Resize(unsigned int width, unsigned int height, HWND windowHandle, bool isMSAAEnabled =
false, bool isTextEnabled = false);
802
803     void SetConstants(unsigned int rootParameterIndex, unsigned int numValues, void* data, unsigned
int index);
804
805
806 //-----
807
808 private:
809
810     //The device resources object that all RenderScene objects share.
DeviceResources* mDeviceResources;
811
812
813 //-----
814
815 //SHADER HASH MAPS
816
817     //Stores all of the shaders for this scene.
std::unordered_map<unsigned int, Microsoft::WRL::ComPtr<ID3DBlob>> mShaders;
818
819     //Stores all of the shaders for this scene.
std::unordered_map<std::wstring_view, Microsoft::WRL::ComPtr<ID3DBlob>> mShadersStr;
820
821 //-----
822
823
824
825
826
827
828
829 //-----
830
831 //INPUT ELEMENT DESCRIPTION HASH MAPS
832
833     //Stores input element descriptions for a set of shaders.
std::unordered_map<unsigned int, std::vector<D3D12_INPUT_ELEMENT_DESC>>
mInputElementDescriptions;
834
835     //Stores input element descriptions for a set of shaders.
std::unordered_map<std::wstring_view, std::vector<D3D12_INPUT_ELEMENT_DESC>>
mInputElementDescriptionsStr;
836
837 //-----
838
839
840
841
842
843
844
845 //-----
846
847 //ROOT PARAMETER HASH MAPS
848
849     //Stores root parameters for root signatures.
std::unordered_map<unsigned int, std::vector<D3D12_ROOT_PARAMETER>> mRootParameters;
850
851     //Stores descriptor ranges for descriptor tables.
std::unordered_map<unsigned int, std::vector<D3D12_DESCRIPTOR_RANGE>> mDescriptorRanges;
852
853     //Stores root parameters for root signatures.
std::unordered_map<std::wstring_view, std::vector<D3D12_ROOT_PARAMETER>> mRootParametersStr;
854
855     //Stores descriptor ranges for descriptor tables.
std::unordered_map<std::wstring_view, std::vector<D3D12_DESCRIPTOR_RANGE>> mDescriptorRangesStr;
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882

```

```

883
884 //-----
885
886
887
888
889
890 //-----
891 //ROOT SIGNATURE HASH MAPS
892
893 //The root signatures for the scene.
894 //Describes all of the constant data that is expected in a set of shaders.
895 //Microsoft::WRL::ComPtr<ID3D12RootSignature> mRootSignature;
896 std::unordered_map<unsigned int, Microsoft::WRL::ComPtr<ID3D12RootSignature> mRootSignatures;
897
898 //Stores static samplers.
899 std::unordered_map<unsigned int, std::vector<D3D12_STATIC_SAMPLER_DESC> mStaticSamplers;
900
901 //The root signatures for the scene.
902 //Describes all of the constant data that is expected in a set of shaders.
903 //Microsoft::WRL::ComPtr<ID3D12RootSignature> mRootSignature;
904 std::unordered_map<std::wstring_view, Microsoft::WRL::ComPtr<ID3D12RootSignature>
mRootSignaturesStr;
905
906 //Stores static samplers.
907 std::unordered_map<std::wstring_view, std::vector<D3D12_STATIC_SAMPLER_DESC> mStaticSamplersStr;
908
909 //-----
910
911
912
913
914
915 //-----
916 //PIPELINE STATE OBJECT HASH MAPS
917
918 //Stores pipeline state objects.
919 std::unordered_map<unsigned int, Microsoft::WRL::ComPtr<ID3D12PipelineState> mPSOs;
920
921 //Stores pipeline state objects.
922 std::unordered_map<std::wstring_view, Microsoft::WRL::ComPtr<ID3D12PipelineState> mPSOsStr;
923
924 //-----
925
926
927
928
929 //-----
930 //STATIC BUFFER HASH MAPS
931
932 //Stores data that will not be updated on a per-frame basis.
933 std::unordered_map<unsigned int, StaticBuffer> mStaticBuffers;
934
935 //Stores data that will not be updated on a per-frame basis.
936 std::unordered_map<std::wstring_view, StaticBuffer> mStaticBuffersStr;
937
938 //-----
939
940
941
942
943
944 //-----
945 //DYNAMIC BUFFER HASH MAPS
946
947 //Stores data that will be updated on a per-frame basis.
948 //We can't update a dynamic buffer until the GPU
949 //is done executing all the commands that reference it, so each frame needs its own dynamic
buffer.
950 std::unordered_map<unsigned int, DynamicBuffer[DeviceResources::NUM_OF_FRAMES]> mDynamicBuffers;
951
952 //Stores data that will be updated on a per-frame basis.
953 //We can't update a dynamic buffer until the GPU
954 //is done executing all the commands that reference it, so each frame needs its own dynamic
buffer.
955 std::unordered_map<std::wstring_view, DynamicBuffer[DeviceResources::NUM_OF_FRAMES]>
mDynamicBuffersStr;
956
957 //-----

```



```

957
958
959         //Used to store descriptors of textures.
960         Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mTextureViewHeap;
961     };
962 };
963 }

```

6.14 SwapChain.h

```

1  #pragma once
2
3  #include <wrl.h>
4  #include "d3dx12.h"
5  #include <dxgi1_4.h>
6  #include <vector>
7  #include <memory>
8  #include "Buffer.h"
9
10 namespace RenderingEngine
11 {
12     class SwapChain
13     {
14     public:
15
16         //No copying
17         SwapChain(const SwapChain&) = delete;
18         SwapChain& operator=(const SwapChain&) = delete;
19
20         SwapChain();
21
22         SwapChain(const Microsoft::WRL::ComPtr<IDXGIFactory4>& dxgiFactory,
23             const Microsoft::WRL::ComPtr<ID3D12CommandQueue>& commandQueue, HWND windowHandle,
24             DXGI_FORMAT rtFormat = DXGI_FORMAT_R8G8B8A8_UNORM, DXGI_FORMAT dsFormat =
25             DXGI_FORMAT_D24_UNORM_S8_UINT,
26             unsigned int numRenderTargetBuffers = 2);
27
28         void CreateSwapChain(const Microsoft::WRL::ComPtr<IDXGIFactory4>& dxgiFactory,
29             const Microsoft::WRL::ComPtr<ID3D12CommandQueue>& commandQueue, HWND windowHandle,
30             DXGI_FORMAT rtFormat = DXGI_FORMAT_R8G8B8A8_UNORM, DXGI_FORMAT dsFormat =
31             DXGI_FORMAT_D24_UNORM_S8_UINT,
32             unsigned int numRenderTargetBuffers = 2);
33
34         const Microsoft::WRL::ComPtr<ID3D12Resource>& GetCurrentBackBuffer() const;
35
36         unsigned int GetNumRenderTargetBuffers() const;
37
38         unsigned int GetCurrentBackBufferIndex() const;
39
40         DXGI_FORMAT GetBackBufferFormat() const;
41
42         DXGI_FORMAT GetDepthStencilFormat() const;
43
44         const std::vector<std::unique_ptr<RenderTargetBuffer>>& GetRenderTargetBuffers();
45
46         void ReleaseBuffers();
47
48         void CreateRenderTargetBuffersAndViews(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
49             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int index,
50             unsigned int rtvSize, unsigned width, unsigned height);
51
52         void CreateDepthStencilBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
53             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int index, unsigned
54             int dsvSize,
55             unsigned int width, unsigned int height);
56
57         void ClearCurrentBackBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
58             commandList,
59             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int indexOfFirstView,
60             unsigned int rtvSize,
61             const float* backBufferClearValue);
62
63         void ClearDepthStencilBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
64             commandList,
65             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int indexOfView,
66             unsigned int dsvSize,
67             float clearValue);
68
69         void Transition(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList,
70             D3D12_RESOURCE_STATES before, D3D12_RESOURCE_STATES after);
71
72         void Present();
73
74     };
75 }

```

```

148     private:
149         unsigned int mNumRenderTargetBuffers;
150         unsigned int mCurrentBackBufferIndex;
151
152         Microsoft::WRL::ComPtr<IDXGISwapChain1> mSwapChain;
153         std::vector<std::unique_ptr<RenderTargetBuffer> mRenderTargetBuffers;
154
155         DepthStencilBuffer mDepthStencilBuffer;
156     };
157 }

```

6.15 Text.h

```

1 #pragma once
2
3
4 #include <string>
5 #include "Color.h"
6
7 namespace RenderingEngine
8 {
9     class Text
10     {
11     public:
12
13         Text() = default;
14
15         Text(const vec4& textLocation, const std::wstring& textString, float textSize, const Color&
16             textColor);
17
18         const vec4& GetTextLocation() const;
19
20         const std::wstring& GetTextString() const;
21
22         float GetTextSize() const;
23
24         const Color& GetTextColor() const;
25
26         void SetTextSize(float textSize);
27
28         void SetTextColor(const Color& textColor);
29
30         void SetTextString(const std::wstring& textString);
31
32         void SetTextLocation(const vec4& textLocation);
33     private:
34         vec4 mTextLocation;
35         std::wstring mText;
36         float mTextSize{ 0.0f };
37         Color mTextColor;
38     };
39 }

```

6.16 TextResources.h

```

1 #pragma once
2
3 #include <wrl.h>
4 #include <d3d11.h>
5 #include <d3d11on12.h>
6 #include <d2d1_3.h>
7 #include <dwrite.h>
8 #include <vector>
9 #include <memory>
10 #include "Buffer.h"
11
12 namespace RenderingEngine
13 {
14     class TextResources
15     {
16     public:
17         TextResources() = default;
18
19         TextResources(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
20             const Microsoft::WRL::ComPtr<ID3D12CommandQueue>& commandQueue, unsigned int
21             numSwapChainBuffers);
22
23     };
24 }

```

```

33     const Microsoft::WRL::ComPtr<ID2D1DeviceContext>& GetDirect2DDeviceContext() const;
34
37     const Microsoft::WRL::ComPtr<IDWriteFactory>& GetDirectWriteFactory() const;
38
41     void ResetBuffers();
42
48     void ResizeBuffers(const std::vector<std::unique_ptr<RenderTargetBuffer>& renderTargetBuffers,
HWND windowHandle);
49
54     void BeforeRenderText(unsigned int currentBackBuffer);
55
60     void AfterRenderText(unsigned int currentBackBuffer);
61
62 private:
63     Microsoft::WRL::ComPtr<ID3D11Device> mDevice11;
64     Microsoft::WRL::ComPtr<ID3D11DeviceContext> mDevice11Context;
65     Microsoft::WRL::ComPtr<ID3D11On12Device> mDevice11on12;
66
67     Microsoft::WRL::ComPtr<ID2D1Device2> mDirect2DDevice;
68     Microsoft::WRL::ComPtr<ID2D1Factory3> mDirect2DFactory;
69     Microsoft::WRL::ComPtr<ID2D1DeviceContext> mDirect2DDeviceContext;
70
71     Microsoft::WRL::ComPtr<IDWriteFactory> mDirectWriteFactory;
72
73     std::vector<Microsoft::WRL::ComPtr<ID3D11Resource>> mWrappedBuffers;
74     std::vector<Microsoft::WRL::ComPtr<ID2D1Bitmap1>> mDirect2DBuffers;
75     std::vector<Microsoft::WRL::ComPtr<IDXGISurface> mSurfaces;
76 };
77 }

```

6.17 Window.h

```

1 #pragma once
2
3 #include <Windows.h>
4 #include <string>
5 #include "Color.h"
6
7 namespace RenderingEngine
8 {
9     struct Window
10     {
11         HWND windowHandle;
12         WNDCLASSEX windowClass;
13     };
14
15     void CreateParentWindow(Window& window, const HINSTANCE& hInstance, WNDPROC windowProcedure, const
RenderingEngine::Color& backgroundColor,
16     const std::wstring& windowClassName, const std::wstring& windowName, unsigned int styles,
17     unsigned int x, unsigned int y, unsigned int width, unsigned int height, void* additionalData =
nullptr);
18
19     void CreateChildWindow(Window& window, const HINSTANCE& hInstance, HWND parent, unsigned long long
int identifier,
20     WNDPROC windowProcedure, const RenderingEngine::Color& backgroundColor,
21     const std::wstring& windowClassName, const std::wstring& windowName, unsigned int styles,
22     unsigned int x, unsigned int y, unsigned int width, unsigned int height, void* additionalData =
nullptr);
23
24     void CreateControlWindow(Window& window, const HINSTANCE& hInstance, HWND parent, unsigned long long
int identifier,
25     const std::wstring& windowClassName, const std::wstring& windowName, unsigned int styles,
26     unsigned int x, unsigned int y, unsigned int width, unsigned int height, void* additionalData =
nullptr);
27
28     unsigned int GetWidth(const Window& window);
29
30     unsigned int GetHeight(const Window& window);
31
32     unsigned int GetX(const Window& window);
33
34     unsigned int GetY(const Window& window);
35 }

```


Index

- ~DeviceResources
 - RenderingEngine::DeviceResources, [30](#)
- ~DynamicBuffer
 - RenderingEngine::DynamicBuffer, [38](#)
- AfterRender
 - RenderingEngine::RenderScene, [50](#)
- AfterRenderObjects
 - RenderingEngine::RenderScene, [50](#)
- AfterRenderText
 - RenderingEngine::RenderScene, [50](#)
 - RenderingEngine::TextResources, [92](#)
- AfterTextDraw
 - RenderingEngine::DeviceResources, [30](#)
- Backward
 - RenderingEngine, [9](#)
- BeforeRenderObjects
 - RenderingEngine::RenderScene, [50](#)
- BeforeRenderText
 - RenderingEngine::RenderScene, [51](#)
 - RenderingEngine::TextResources, [92](#)
- BeforeTextDraw
 - RenderingEngine::DeviceResources, [30](#)
- CheckMultiSamplingSupport
 - RenderingEngine::MultiSampling, [42](#)
- ClearCurrentBackBuffer
 - RenderingEngine::SwapChain, [84](#)
- ClearDepthStencilBuffer
 - RenderingEngine::DepthStencilBuffer, [27](#)
 - RenderingEngine::MultiSampling, [43](#)
 - RenderingEngine::SwapChain, [84](#)
- ClearRenderTargetBuffer
 - RenderingEngine::MultiSampling, [43](#)
 - RenderingEngine::RenderTargetBuffer, [75](#)
- Color
 - RenderingEngine::Color, [22](#), [23](#)
- CompileShader
 - RenderingEngine::RenderScene, [51](#)
- CopyData
 - RenderingEngine::DynamicBuffer, [38](#)
- CopyDataIntoDynamicBuffer
 - RenderingEngine::RenderScene, [52](#)
- CreateChildWindow
 - RenderingEngine, [11](#)
- CreateConstantBufferView
 - RenderingEngine::DynamicBuffer, [39](#)
- CreateControlWindow
 - RenderingEngine, [11](#)
- CreateDepthStencilBufferAndView
 - RenderingEngine::DepthStencilBuffer, [28](#)
 - RenderingEngine::MultiSampling, [43](#)
 - RenderingEngine::SwapChain, [85](#)
- CreateDescriptorRange
 - RenderingEngine::RenderScene, [53](#)
- CreateDescriptorTable
 - RenderingEngine::RenderScene, [53](#), [54](#)
- CreateDynamicBuffer
 - RenderingEngine::DynamicBuffer, [39](#)
 - RenderingEngine::RenderScene, [54](#), [55](#)
- CreateInputElementDescription
 - RenderingEngine::RenderScene, [56](#)
- CreateParentWindow
 - RenderingEngine, [12](#)
- CreatePSO
 - RenderingEngine::RenderScene, [57](#), [58](#)
- CreateRenderTargetBufferAndView
 - RenderingEngine::MultiSampling, [44](#)
 - RenderingEngine::RenderTargetBuffer, [75](#)
- CreateRenderTargetBuffersAndViews
 - RenderingEngine::SwapChain, [85](#)
- CreateRootConstants
 - RenderingEngine::RenderScene, [58](#), [59](#)
- CreateRootDescriptor
 - RenderingEngine::RenderScene, [59](#)
- CreateRootSignature
 - RenderingEngine::RenderScene, [60](#), [61](#)
- CreateStaticBuffer
 - RenderingEngine::RenderScene, [61–63](#)
 - RenderingEngine::StaticBuffer, [79](#), [80](#)
- CreateStaticSampler
 - RenderingEngine::RenderScene, [64](#)
- CreateSwapChain
 - RenderingEngine::SwapChain, [86](#)
- CreateTexture2DMSView
 - RenderingEngine::StaticBuffer, [81](#)
- CreateTexture2DView
 - RenderingEngine::StaticBuffer, [81](#)
- CreateTextureViewHeap
 - RenderingEngine::RenderScene, [66](#)
- DepthStencilBuffer
 - RenderingEngine::DepthStencilBuffer, [27](#)
- DirectXException, [34](#)
 - DirectXException, [35](#)
 - ErrorMsg, [35](#)
- Down
 - RenderingEngine, [13](#)
- DynamicBuffer

- RenderingEngine::DynamicBuffer, [37](#), [38](#)
- ErrorMsg
 - DirectXException, [35](#)
- Execute
 - RenderingEngine::DeviceResources, [30](#)
- ExecuteAndFlush
 - RenderingEngine::RenderScene, [66](#)
- FlushCommandQueue
 - RenderingEngine::DeviceResources, [31](#)
- Forward
 - RenderingEngine, [13](#)
- GetAlpha
 - RenderingEngine::Color, [23](#)
- GetBackBufferFormat
 - RenderingEngine::DeviceResources, [31](#)
 - RenderingEngine::SwapChain, [86](#)
- GetBlue
 - RenderingEngine::Color, [23](#)
- GetCBVSRVUAVSize
 - RenderingEngine::DeviceResources, [31](#)
- GetColor
 - RenderingEngine::Color, [23](#)
- GetCommandList
 - RenderingEngine::DeviceResources, [31](#)
- GetCurrentBackBuffer
 - RenderingEngine::SwapChain, [86](#)
- GetCurrentBackBufferIndex
 - RenderingEngine::SwapChain, [86](#)
- GetCurrentFrame
 - RenderingEngine::DeviceResources, [31](#)
- GetDepthStencilFormat
 - RenderingEngine::DepthStencilBuffer, [28](#)
 - RenderingEngine::DeviceResources, [31](#)
 - RenderingEngine::MultiSampling, [44](#)
 - RenderingEngine::SwapChain, [87](#)
- GetDevice
 - RenderingEngine::DeviceResources, [32](#)
- GetDirect2DDeviceContext
 - RenderingEngine::TextResources, [92](#)
- GetDirectWriteFactory
 - RenderingEngine::TextResources, [92](#)
- GetGPUAddress
 - RenderingEngine::DynamicBuffer, [40](#)
- GetGreen
 - RenderingEngine::Color, [23](#)
- GetHeight
 - RenderingEngine, [14](#)
- GetIndexBufferView
 - RenderingEngine::DynamicBuffer, [40](#)
 - RenderingEngine::StaticBuffer, [81](#)
- GetInstance
 - RenderingEngine::DeviceResources, [32](#)
- GetNumRenderTargetBuffers
 - RenderingEngine::SwapChain, [87](#)
- GetRed
 - RenderingEngine::Color, [24](#)
- GetRenderTargetBuffer
 - RenderingEngine::MultiSampling, [44](#)
 - RenderingEngine::RenderTargetBuffer, [76](#)
- GetRenderTargetFormat
 - RenderingEngine::MultiSampling, [45](#)
 - RenderingEngine::RenderTargetBuffer, [76](#)
- GetTextColor
 - RenderingEngine::Text, [89](#)
- GetTextLocation
 - RenderingEngine::Text, [89](#)
- GetTextResources
 - RenderingEngine::DeviceResources, [32](#)
- GetTextSize
 - RenderingEngine::Text, [89](#)
- GetTextString
 - RenderingEngine::Text, [89](#)
- GetVertexBufferView
 - RenderingEngine::DynamicBuffer, [40](#)
 - RenderingEngine::StaticBuffer, [81](#)
- GetWidth
 - RenderingEngine, [14](#)
- GetX
 - RenderingEngine, [14](#)
- GetY
 - RenderingEngine, [14](#)
- InitializeTime
 - RenderingEngine, [14](#)
- Left
 - RenderingEngine, [14](#)
- LinkDynamicBuffer
 - RenderingEngine::RenderScene, [66](#), [67](#)
- LinkPSOAndRootSignature
 - RenderingEngine::RenderScene, [67](#), [68](#)
- LinkStaticBuffer
 - RenderingEngine::RenderScene, [68](#)
- LinkTexture
 - RenderingEngine::RenderScene, [70](#)
- LinkTextureViewHeap
 - RenderingEngine::RenderScene, [70](#)
- LoadShader
 - RenderingEngine::RenderScene, [70](#), [71](#)
- LookAt
 - RenderingEngine, [14](#)
- MultiSampling
 - RenderingEngine::MultiSampling, [41](#), [42](#)
- NextFrame
 - RenderingEngine::DeviceResources, [32](#)
- NUM_OF_FRAMES
 - RenderingEngine::DeviceResources, [34](#)
- operator*
 - RenderingEngine, [15](#)
- operator*=
 - RenderingEngine::Color, [24](#)
- operator+

- RenderingEngine, 15
- operator+=
 - RenderingEngine::Color, 24
- operator-
 - RenderingEngine, 15
- operator-=
 - RenderingEngine::Color, 24
- Present
 - RenderingEngine::DeviceResources, 33
 - RenderingEngine::SwapChain, 87
- ReleaseBuffer
 - RenderingEngine::DepthStencilBuffer, 28
 - RenderingEngine::DynamicBuffer, 40
 - RenderingEngine::RenderTargetBuffer, 76
 - RenderingEngine::StaticBuffer, 82
- ReleaseBuffers
 - RenderingEngine::MultiSampling, 45
 - RenderingEngine::SwapChain, 87
- RemoveShader
 - RenderingEngine::RenderScene, 71
- RenderingEngine, 7
 - Backward, 9
 - CreateChildWindow, 11
 - CreateControlWindow, 11
 - CreateParentWindow, 12
 - Down, 13
 - Foward, 13
 - GetHeight, 14
 - GetWidth, 14
 - GetX, 14
 - GetY, 14
 - InitializeTime, 14
 - Left, 14
 - LookAt, 14
 - operator*, 15
 - operator+, 15
 - operator-, 15
 - Reset, 16
 - Right, 16
 - RotateCameraLeftRight, 16
 - RotateCameraUpDown, 16
 - SetProperties, 17, 18
 - Start, 18
 - Stop, 18
 - Tick, 18
 - Up, 19
 - UpdateProjectionMatrix, 19
 - UpdateViewMatrix, 19
- RenderingEngine::Camera, 21
- RenderingEngine::Color, 21
 - Color, 22, 23
 - GetAlpha, 23
 - GetBlue, 23
 - GetColor, 23
 - GetGreen, 23
 - GetRed, 24
 - operator*=: 24
 - operator+=, 24
 - operator-=, 24
 - SetAlpha, 25
 - SetBlue, 25
 - SetColor, 25
 - SetGreen, 25
 - SetRed, 25
- RenderingEngine::DepthStencilBuffer, 26
 - ClearDepthStencilBuffer, 27
 - CreateDepthStencilBufferAndView, 28
 - DepthStencilBuffer, 27
 - GetDepthStencilFormat, 28
 - ReleaseBuffer, 28
- RenderingEngine::DeviceResources, 29
 - ~DeviceResources, 30
 - AfterTextDraw, 30
 - BeforeTextDraw, 30
 - Execute, 30
 - FlushCommandQueue, 31
 - GetBackBufferFormat, 31
 - GetCBVSRVUAVSize, 31
 - GetCommandList, 31
 - GetCurrentFrame, 31
 - GetDepthStencilFormat, 31
 - GetDevice, 32
 - GetInstance, 32
 - GetTextResources, 32
 - NextFrame, 32
 - NUM_OF_FRAMES, 34
 - Present, 33
 - Resize, 33
 - RTBufferTransition, 33
 - Signal, 34
 - UpdateCurrentFrameFenceValue, 34
 - WaitForGPU, 34
- RenderingEngine::DrawArguments, 36
- RenderingEngine::DynamicBuffer, 36
 - ~DynamicBuffer, 38
 - CopyData, 38
 - CreateConstantBufferView, 39
 - CreateDynamicBuffer, 39
 - DynamicBuffer, 37, 38
 - GetGPUAddress, 40
 - GetIndexBufferView, 40
 - GetVertexBufferView, 40
 - ReleaseBuffer, 40
- RenderingEngine::MultiSampling, 40
 - CheckMultiSamplingSupport, 42
 - ClearDepthStencilBuffer, 43
 - ClearRenderTargetBuffer, 43
 - CreateDepthStencilBufferAndView, 43
 - CreateRenderTargetBufferAndView, 44
 - GetDepthStencilFormat, 44
 - GetRenderTargetBuffer, 44
 - GetRenderTargetFormat, 45
 - MultiSampling, 41, 42
 - ReleaseBuffers, 45
 - Transition, 45

- RenderingEngine::OrthographicProjection, 45
- RenderingEngine::PerspectiveProjection, 46
- RenderingEngine::RenderScene, 46
 - AfterRender, 50
 - AfterRenderObjects, 50
 - AfterRenderText, 50
 - BeforeRenderObjects, 50
 - BeforeRenderText, 51
 - CompileShader, 51
 - CopyDataIntoDynamicBuffer, 52
 - CreateDescriptorRange, 53
 - CreateDescriptorTable, 53, 54
 - CreateDynamicBuffer, 54, 55
 - CreateInputElementDescription, 56
 - CreatePSO, 57, 58
 - CreateRootConstants, 58, 59
 - CreateRootDescriptor, 59
 - CreateRootSignature, 60, 61
 - CreateStaticBuffer, 61–63
 - CreateStaticSampler, 64
 - CreateTextureViewHeap, 66
 - ExecuteAndFlush, 66
 - LinkDynamicBuffer, 66, 67
 - LinkPSOAndRootSignature, 67, 68
 - LinkStaticBuffer, 68
 - LinkTexture, 70
 - LinkTextureViewHeap, 70
 - LoadShader, 70, 71
 - RemoveShader, 71
 - RenderObject, 71
 - RenderText, 72
 - Resize, 73
 - SetConstants, 73
- RenderingEngine::RenderTargetBuffer, 73
 - ClearRenderTargetBuffer, 75
 - CreateRenderTargetBufferAndView, 75
 - GetRenderTargetBuffer, 76
 - GetRenderTargetFormat, 76
 - ReleaseBuffer, 76
 - RenderTargetBuffer, 74
- RenderingEngine::StaticBuffer, 77
 - CreateStaticBuffer, 79, 80
 - CreateTexture2DMSView, 81
 - CreateTexture2DView, 81
 - GetIndexBufferView, 81
 - GetVertexBufferView, 81
 - ReleaseBuffer, 82
 - StaticBuffer, 78, 79
- RenderingEngine::SwapChain, 82
 - ClearCurrentBackBuffer, 84
 - ClearDepthStencilBuffer, 84
 - CreateDepthStencilBufferAndView, 85
 - CreateRenderTargetBuffersAndViews, 85
 - CreateSwapChain, 86
 - GetBackBufferFormat, 86
 - GetCurrentBackBuffer, 86
 - GetCurrentBackBufferIndex, 86
 - GetDepthStencilFormat, 87
 - GetNumRenderTargetBuffers, 87
 - Present, 87
 - ReleaseBuffers, 87
 - SwapChain, 83
 - Transition, 87
- RenderingEngine::Text, 88
 - GetTextColor, 89
 - GetTextLocation, 89
 - GetTextSize, 89
 - GetTextString, 89
 - SetTextColor, 90
 - SetTextLocation, 90
 - SetTextSize, 90
 - SetTextString, 90
 - Text, 88
- RenderingEngine::TextResources, 91
 - AfterRenderText, 92
 - BeforeRenderText, 92
 - GetDirect2DDeviceContext, 92
 - GetDirectWriteFactory, 92
 - ResetBuffers, 92
 - ResizeBuffers, 93
 - TextResources, 91
- RenderingEngine::Time, 93
- RenderingEngine::Window, 94
- RenderObject
 - RenderingEngine::RenderScene, 71
- RenderTargetBuffer
 - RenderingEngine::RenderTargetBuffer, 74
- RenderText
 - RenderingEngine::RenderScene, 72
- Reset
 - RenderingEngine, 16
- ResetBuffers
 - RenderingEngine::TextResources, 92
- Resize
 - RenderingEngine::DeviceResources, 33
 - RenderingEngine::RenderScene, 73
- ResizeBuffers
 - RenderingEngine::TextResources, 93
- Right
 - RenderingEngine, 16
- RotateCameraLeftRight
 - RenderingEngine, 16
- RotateCameraUpDown
 - RenderingEngine, 16
- RTBufferTransition
 - RenderingEngine::DeviceResources, 33
- SetAlpha
 - RenderingEngine::Color, 25
- SetBlue
 - RenderingEngine::Color, 25
- SetColor
 - RenderingEngine::Color, 25
- SetConstants
 - RenderingEngine::RenderScene, 73
- SetGreen
 - RenderingEngine::Color, 25

- SetProperties
 - RenderingEngine, [17](#), [18](#)
- SetRed
 - RenderingEngine::Color, [25](#)
- SetTextColor
 - RenderingEngine::Text, [90](#)
- SetTextLocation
 - RenderingEngine::Text, [90](#)
- SetTextSize
 - RenderingEngine::Text, [90](#)
- SetTextString
 - RenderingEngine::Text, [90](#)
- Signal
 - RenderingEngine::DeviceResources, [34](#)
- Start
 - RenderingEngine, [18](#)
- StaticBuffer
 - RenderingEngine::StaticBuffer, [78](#), [79](#)
- Stop
 - RenderingEngine, [18](#)
- SwapChain
 - RenderingEngine::SwapChain, [83](#)
- Text
 - RenderingEngine::Text, [88](#)
- TextResources
 - RenderingEngine::TextResources, [91](#)
- Tick
 - RenderingEngine, [18](#)
- Transition
 - RenderingEngine::MultiSampling, [45](#)
 - RenderingEngine::SwapChain, [87](#)
- Up
 - RenderingEngine, [19](#)
- UpdateCurrentFrameFenceValue
 - RenderingEngine::DeviceResources, [34](#)
- UpdateProjectionMatrix
 - RenderingEngine, [19](#)
- UpdateViewMatrix
 - RenderingEngine, [19](#)
- WaitForGPU
 - RenderingEngine::DeviceResources, [34](#)