

Farouq Adepetu's Rendering Engine

Generated by Doxygen 1.9.4

1 Namespace Index	1
1.1 Namespace List	1
2 Hierarchical Index	3
2.1 Class Hierarchy	3
3 Class Index	5
3.1 Class List	5
4 File Index	7
4.1 File List	7
5 Namespace Documentation	9
5.1 FACamera Namespace Reference	9
5.1.1 Detailed Description	9
5.2 FAColor Namespace Reference	9
5.2.1 Detailed Description	10
5.2.2 Function Documentation	10
5.2.2.1 operator*() [1/3]	10
5.2.2.2 operator*() [2/3]	10
5.2.2.3 operator*() [3/3]	11
5.2.2.4 operator+()	11
5.2.2.5 operator-()	11
5.3 FAProjection Namespace Reference	11
5.3.1 Detailed Description	11
5.4 FARender Namespace Reference	12
5.4.1 Detailed Description	12
5.5 FATime Namespace Reference	12
5.5.1 Detailed Description	13
5.6 FAWindow Namespace Reference	13
5.6.1 Detailed Description	13
6 Class Documentation	15
6.1 FACamera::Camera Class Reference	15
6.1.1 Detailed Description	16
6.1.2 Constructor & Destructor Documentation	16
6.1.2.1 Camera() [1/2]	17
6.1.2.2 Camera() [2/2]	17
6.1.3 Member Function Documentation	17
6.1.3.1 Backward()	17
6.1.3.2 Down()	18
6.1.3.3 Foward()	18
6.1.3.4 GetAngularVelocity()	18
6.1.3.5 GetCameraPosition()	18

6.1.3.6 GetCameraVelocity()	18
6.1.3.7 GetViewMatrix()	19
6.1.3.8 GetX()	19
6.1.3.9 GetY()	19
6.1.3.10 GetZ()	19
6.1.3.11 KeyboardInput()	19
6.1.3.12 KeyboardInputArrow()	20
6.1.3.13 KeyboardInputWASD()	20
6.1.3.14 Left()	20
6.1.3.15 LookAt()	20
6.1.3.16 MouseInput()	21
6.1.3.17 Right()	21
6.1.3.18 RotateCameraLeftRight()	21
6.1.3.19 RotateCameraUpDown()	21
6.1.3.20 SetAngularVelocity()	22
6.1.3.21 SetCameraPosition()	22
6.1.3.22 SetCameraVelocity()	22
6.1.3.23 SetProperties()	22
6.1.3.24 SetX()	23
6.1.3.25 SetY()	23
6.1.3.26 SetZ()	23
6.1.3.27 Up()	23
6.1.3.28 UpdateViewMatrix()	23
6.2 FAColor::Color Class Reference	24
6.2.1 Detailed Description	24
6.2.2 Constructor & Destructor Documentation	25
6.2.2.1 Color() [1/2]	25
6.2.2.2 Color() [2/2]	25
6.2.3 Member Function Documentation	25
6.2.3.1 GetAlpha()	25
6.2.3.2 GetBlue()	25
6.2.3.3 GetColor()	25
6.2.3.4 GetGreen()	26
6.2.3.5 GetRed()	26
6.2.3.6 operator*=() [1/2]	26
6.2.3.7 operator*=() [2/2]	26
6.2.3.8 operator+=()	26
6.2.3.9 operator-=()	27
6.2.3.10 SetAlpha()	27
6.2.3.11 SetBlue()	27
6.2.3.12 SetColor()	27
6.2.3.13 SetGreen()	27

6.2.3.14 SetRed()	28
6.3 FAREnder::DepthStencilBuffer Class Reference	28
6.3.1 Detailed Description	28
6.3.2 Constructor & Destructor Documentation	29
6.3.2.1 DepthStencilBuffer() [1/2]	29
6.3.2.2 DepthStencilBuffer() [2/2]	29
6.3.3 Member Function Documentation	29
6.3.3.1 ClearDepthStencilBuffer()	29
6.3.3.2 CreateDepthStencilBufferAndView()	30
6.3.3.3 GetDepthStencilFormat()	30
6.3.3.4 ReleaseBuffer()	30
6.4 FAREnder::DeviceResources Class Reference	31
6.4.1 Detailed Description	32
6.4.2 Constructor & Destructor Documentation	32
6.4.2.1 ~DeviceResources()	32
6.4.3 Member Function Documentation	32
6.4.3.1 AfterTextDraw()	32
6.4.3.2 BeforeTextDraw()	32
6.4.3.3 Execute()	33
6.4.3.4 FlushCommandQueue()	33
6.4.3.5 GetBackBufferFormat()	33
6.4.3.6 GetCBVSRVUAVSize()	33
6.4.3.7 GetCommandList()	33
6.4.3.8 GetCurrentFrame()	33
6.4.3.9 GetDepthStencilFormat()	34
6.4.3.10 GetDevice()	34
6.4.3.11 GetInstance()	34
6.4.3.12 GetTextResources()	34
6.4.3.13 NextFrame()	35
6.4.3.14 Present()	35
6.4.3.15 Resize()	35
6.4.3.16 RTBufferTransition()	35
6.4.3.17 Signal()	36
6.4.3.18 UpdateCurrentFrameFenceValue()	36
6.4.3.19 WaitForGPU()	36
6.4.4 Member Data Documentation	36
6.4.4.1 NUM_OF_FRAMES	36
6.5 DirectXException Class Reference	36
6.5.1 Detailed Description	37
6.5.2 Constructor & Destructor Documentation	37
6.5.2.1 DirectXException()	37
6.5.3 Member Function Documentation	37

6.5.3.1 ErrorMessage()	37
6.6 FAREnder::DynamicBuffer Class Reference	38
6.6.1 Detailed Description	38
6.6.2 Constructor & Destructor Documentation	39
6.6.2.1 DynamicBuffer() [1/3]	39
6.6.2.2 DynamicBuffer() [2/3]	39
6.6.2.3 DynamicBuffer() [3/3]	39
6.6.2.4 ~DynamicBuffer()	40
6.6.3 Member Function Documentation	40
6.6.3.1 CopyData()	40
6.6.3.2 CreateConstantBufferView()	40
6.6.3.3 CreateDynamicBuffer() [1/2]	41
6.6.3.4 CreateDynamicBuffer() [2/2]	41
6.6.3.5 GetGPUAddress()	41
6.6.3.6 GetIndexBufferView()	41
6.6.3.7 GetVertexBufferView()	42
6.6.3.8 ReleaseBuffer()	42
6.7 FAProjection::IProjection Class Reference	42
6.7.1 Detailed Description	42
6.7.2 Constructor & Destructor Documentation	43
6.7.2.1 IProjection()	43
6.7.3 Member Function Documentation	43
6.7.3.1 GetProjectionMatrix()	43
6.7.3.2 UpdateProjectionMatrix()	43
6.8 FAREnder::MultiSampling Class Reference	43
6.8.1 Detailed Description	44
6.8.2 Constructor & Destructor Documentation	44
6.8.2.1 MultiSampling() [1/2]	45
6.8.2.2 MultiSampling() [2/2]	45
6.8.3 Member Function Documentation	45
6.8.3.1 CheckMultiSamplingSupport()	45
6.8.3.2 ClearDepthStencilBuffer()	46
6.8.3.3 ClearRenderTargetBuffer()	46
6.8.3.4 CreateDepthStencilBufferAndView()	46
6.8.3.5 CreateRenderTargetBufferAndView()	47
6.8.3.6 GetDepthStencilFormat()	47
6.8.3.7 GetRenderTargetBuffer()	48
6.8.3.8 GetRenderTargetFormat()	48
6.8.3.9 ReleaseBuffers()	48
6.8.3.10 Transition()	48
6.9 FAProjection::PerspectiveProjection Class Reference	48
6.9.1 Detailed Description	49

6.9.2 Constructor & Destructor Documentation	49
6.9.2.1 PerspectiveProjection() [1/2]	50
6.9.2.2 PerspectiveProjection() [2/2]	50
6.9.3 Member Function Documentation	50
6.9.3.1 GetAspectRatio()	50
6.9.3.2 GetFar()	50
6.9.3.3 GetNear()	51
6.9.3.4 GetVerticalFov()	51
6.9.3.5 SetAspectRatio()	51
6.9.3.6 SetFar()	51
6.9.3.7 SetNear()	51
6.9.3.8 SetProperties()	51
6.9.3.9 SetVerticalFov()	52
6.9.3.10 UpdateProjectionMatrix()	52
6.10 FAREnder::RenderScene Class Reference	52
6.10.1 Detailed Description	54
6.10.2 Member Function Documentation	55
6.10.2.1 AfterRender()	55
6.10.2.2 AfterRenderObjects()	55
6.10.2.3 AfterRenderText()	55
6.10.2.4 BeforeRenderObjects()	55
6.10.2.5 BeforeRenderText()	56
6.10.2.6 CompileShader()	56
6.10.2.7 CopyDataIntoDynamicBuffer()	56
6.10.2.8 CreateDescriptorRange()	57
6.10.2.9 CreateDescriptorTable()	57
6.10.2.10 CreateDynamicBuffer() [1/2]	57
6.10.2.11 CreateDynamicBuffer() [2/2]	58
6.10.2.12 CreateInputElementDescription()	58
6.10.2.13 CreatePSO()	59
6.10.2.14 CreateRootConstants()	59
6.10.2.15 CreateRootDescriptor()	60
6.10.2.16 CreateRootSignature() [1/2]	60
6.10.2.17 CreateRootSignature() [2/2]	60
6.10.2.18 CreateStaticBuffer() [1/3]	61
6.10.2.19 CreateStaticBuffer() [2/3]	61
6.10.2.20 CreateStaticBuffer() [3/3]	62
6.10.2.21 CreateStaticSampler()	62
6.10.2.22 CreateTextureViewHeap()	63
6.10.2.23 ExecuteAndFlush()	63
6.10.2.24 LoadShader()	63
6.10.2.25 RemoveShader()	63

6.10.2.26 RenderObject()	64
6.10.2.27 RenderText()	64
6.10.2.28 Resize()	65
6.10.2.29 SetConstants()	65
6.10.2.30 SetDynamicBuffer()	65
6.10.2.31 SetPSOAndRootSignature()	66
6.10.2.32 SetStaticBuffer()	66
6.10.2.33 SetTexture()	67
6.11 FARender::RenderTargetBuffer Class Reference	67
6.11.1 Detailed Description	68
6.11.2 Constructor & Destructor Documentation	68
6.11.2.1 RenderTargetBuffer() [1/2]	68
6.11.2.2 RenderTargetBuffer() [2/2]	68
6.11.3 Member Function Documentation	69
6.11.3.1 ClearRenderTargetBuffer()	69
6.11.3.2 CreateRenderTargetBufferAndView()	69
6.11.3.3 GetRenderTargetBuffer() [1/2]	70
6.11.3.4 GetRenderTargetBuffer() [2/2]	70
6.11.3.5 GetRenderTargetFormat()	70
6.11.3.6 ReleaseBuffer()	70
6.12 FARender::StaticBuffer Class Reference	71
6.12.1 Detailed Description	71
6.12.2 Constructor & Destructor Documentation	72
6.12.2.1 StaticBuffer() [1/4]	72
6.12.2.2 StaticBuffer() [2/4]	72
6.12.2.3 StaticBuffer() [3/4]	72
6.12.2.4 StaticBuffer() [4/4]	73
6.12.3 Member Function Documentation	73
6.12.3.1 CreateStaticBuffer() [1/3]	73
6.12.3.2 CreateStaticBuffer() [2/3]	73
6.12.3.3 CreateStaticBuffer() [3/3]	74
6.12.3.4 CreateTexture2DMSView()	74
6.12.3.5 CreateTexture2DView()	75
6.12.3.6 GetIndexBufferView()	75
6.12.3.7 GetVertexBufferView()	75
6.12.3.8 ReleaseBuffer()	75
6.13 FARender::SwapChain Class Reference	76
6.13.1 Detailed Description	77
6.13.2 Constructor & Destructor Documentation	77
6.13.2.1 SwapChain() [1/2]	77
6.13.2.2 SwapChain() [2/2]	77
6.13.3 Member Function Documentation	78

6.13.3.1 ClearCurrentBackBuffer()	78
6.13.3.2 ClearDepthStencilBuffer()	78
6.13.3.3 CreateDepthStencilBufferAndView()	79
6.13.3.4 CreateRenderTargetBuffersAndViews()	79
6.13.3.5 CreateSwapChain()	79
6.13.3.6 GetBackBufferFormat()	80
6.13.3.7 GetCurrentBackBuffer()	80
6.13.3.8 GetCurrentBackBufferIndex()	80
6.13.3.9 GetDepthStencilFormat()	80
6.13.3.10 GetNumRenderTargetBuffers()	81
6.13.3.11 Present()	81
6.13.3.12 ReleaseBuffers()	81
6.13.3.13 Transition()	81
6.14 FARender::Text Class Reference	81
6.14.1 Detailed Description	82
6.14.2 Constructor & Destructor Documentation	82
6.14.2.1 Text()	82
6.14.3 Member Function Documentation	83
6.14.3.1 GetTextColor()	83
6.14.3.2 GetTextLocation()	83
6.14.3.3 GetTextSize()	83
6.14.3.4 GetTextString()	83
6.14.3.5 SetTextColor()	83
6.14.3.6 SetTextLocation()	84
6.14.3.7 SetTextSize()	84
6.14.3.8 SetTextString()	84
6.15 FARender::TextResources Class Reference	84
6.15.1 Detailed Description	85
6.15.2 Constructor & Destructor Documentation	85
6.15.2.1 TextResources()	85
6.15.3 Member Function Documentation	85
6.15.3.1 AfterRenderText()	85
6.15.3.2 BeforeRenderText()	85
6.15.3.3 GetDirect2DDeviceContext()	86
6.15.3.4 GetDirectWriteFactory()	86
6.15.3.5 ResetBuffers()	86
6.15.3.6 ResizeBuffers()	86
6.16 FATime::Time Class Reference	87
6.16.1 Detailed Description	87
6.16.2 Constructor & Destructor Documentation	87
6.16.2.1 Time()	87
6.16.3 Member Function Documentation	87

6.16.3.1 DeltaTime()	88
6.16.3.2 Reset()	88
6.16.3.3 Start()	88
6.16.3.4 Stop()	88
6.16.3.5 Tick()	88
6.16.3.6 TotalTime()	88
6.17 FAWindow::Window Class Reference	89
6.17.1 Detailed Description	90
6.17.2 Constructor & Destructor Documentation	90
6.17.2.1 Window() [1/4]	90
6.17.2.2 Window() [2/4]	90
6.17.2.3 Window() [3/4]	91
6.17.2.4 Window() [4/4]	92
6.17.3 Member Function Documentation	92
6.17.3.1 CreateChildWindow()	92
6.17.3.2 CreateControlWindow()	93
6.17.3.3 CreateParentWindow()	94
6.17.3.4 GetHeight()	95
6.17.3.5 GetWidth()	95
6.17.3.6 GetWindowHandle()	95
6.17.3.7 GetX()	95
6.17.3.8 GetY()	95
6.17.3.9 SetHeight()	95
6.17.3.10 SetWidth()	96
6.17.3.11 SetX()	96
6.17.3.12 SetY()	96
7 File Documentation	97
7.1 DDSTextureLoader.h	97
7.2 Direct3DLink.h	98
7.3 FABuffer.h File Reference	98
7.3.1 Detailed Description	99
7.4 FABuffer.h	99
7.5 FACamera.h File Reference	101
7.5.1 Detailed Description	102
7.5.2 Typedef Documentation	102
7.5.2.1 vec2	102
7.6 FACamera.h	102
7.7 FAColor.h File Reference	103
7.7.1 Detailed Description	104
7.8 FAColor.h	104
7.9 FADeviceResources.h File Reference	105

7.9.1 Detailed Description	105
7.10 FADeviceResources.h	106
7.11 FADirectXException.h	107
7.12 FAMultiSampling.h	107
7.13 FAProjection.h File Reference	108
7.13.1 Detailed Description	109
7.14 FAProjection.h	109
7.15 FARenderScene.h File Reference	110
7.15.1 Detailed Description	110
7.16 FARenderScene.h	110
7.17 FASwapChain.h	112
7.18 FAText.h File Reference	113
7.18.1 Detailed Description	114
7.19 FAText.h	114
7.20 FATextResources.h	114
7.21 FATime.h File Reference	115
7.21.1 Detailed Description	115
7.22 FATime.h	116
7.23 FAWindow.h File Reference	116
7.23.1 Detailed Description	117
7.24 FAWindow.h	117
Index	119

Chapter 1

Namespace Index

1.1 Namespace List

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

FACamera	Has Camera class	9
FAColor	Has the Color class	9
FAProjection	Within the namespace is the interface IProjection and class PerspectiveProjection	11
FARender	Has classes that are used for rendering objects and text through the Direct3D 12 API	12
FATime	Has Time class	12
FAWindow	Has Window class	13

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

FACamera::Camera	15
FAColor::Color	24
FARender::DepthStencilBuffer	28
FARender::DeviceResources	31
DirectXException	36
FARender::DynamicBuffer	38
FAProjection::IProjection	42
FAProjection::PerspectiveProjection	48
FARender::MultiSampling	43
FARender::RenderScene	52
FARender::RenderTargetBuffer	67
FARender::StaticBuffer	71
FARender::SwapChain	76
FARender::Text	81
FARender::TextResources	84
FATime::Time	87
FAWindow::Window	89

Chapter 3

Class Index

3.1 Class List

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

[FACamera::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.

15

[FAColor::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

24

[FARender::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

28

[FARender::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

31

[DirectXException](#)

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

36

[FARender::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

38

[FAProjection::IProjection](#)

An interface for projections

42

[FARender::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

43

[FAProjection::PerspectiveProjection](#)

A class for doing perspective projection

48

[FARender::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

52

[FARender::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

67

FARender::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	71
FARender::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	76
FARender::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	81
FARender::TextResources	
A wrapper for resources that are needed to render text using Direct3D 11on12, Direct2D and DirectWrite	84
FATime::Time	
This class is used to get the time between each frame. You can stop start, reset and get the total time	87
FAWindow::Window	
The window class is used to make a Window using Windows API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable	89

Chapter 4

File Index

4.1 File List

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

DDSTextureLoader.h	??
Direct3DLink.h	??
FABuffer.h File has the classes RenderTargetBuffer , DepthStencilBuffer , StaticBuffer and DynamicBuffer under namespace FARender	98
FACamera.h File that has namespace FACamera . Withn the namespace is the class Camera	101
FAColor.h File has class Color under namespace FAColor	103
FADeviceResources.h File has class DeviceResources under namespace FARender	105
FADirectXException.h	??
FAMultiSampling.h	??
FAProjection.h File that has namespace FAProjection . Within the namespace is the interface IProjection and class PerspectiveProjection	108
FARenderScene.h File has class RenderScene under namespace FARender	110
FASwapChain.h	??
FAText.h File has class Text under namespace FARender	113
FATextResources.h	??
FATime.h File that has namespace FATime . Withn the namespace is the class Time	115
FAWindow.h File that has namespace FAWindow . Withn the namespace is the class Window	116

Chapter 5

Namespace Documentation

5.1 FACamera Namespace Reference

Has [Camera](#) class.

Classes

- class [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.

5.1.1 Detailed Description

Has [Camera](#) class.

5.2 FAColor Namespace Reference

Has the [Color](#) class.

Classes

- 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 componet is red, second component is green, third component is blue and the 4th component is alpha.

Functions

- **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 $c1 * c2$.*

5.2.1 Detailed Description

Has the **Color** class.

5.2.2 Function Documentation

5.2.2.1 operator*() [1/3]

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

Returns the result of $c * k$.

If $\|a\| < 0.0f$, no multiplication happens and **Color** c is returned.
If any of the resultant components are $> 1.0f$, they are set to $1.0f$.

5.2.2.2 operator*() [2/3]

```
Color FColor::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 $1.0f$.

5.2.2.3 operator*() [3/3]

```
Color FAProjection::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 $1.0f$.

5.2.2.4 operator+()

```
Color FAProjection::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 $1.0f$.

5.2.2.5 operator-()

```
Color FAProjection::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 $0.0f$.

5.3 FAProjection Namespace Reference

Within the namespace is the interface [IProjection](#) and class [PerspectiveProjection](#).

Classes

- class [IProjection](#)
An interface for projections.
- class [PerspectiveProjection](#)
A class for doing perspective projection.

5.3.1 Detailed Description

Within the namespace is the interface [IProjection](#) and class [PerspectiveProjection](#).

5.4 FARender Namespace Reference

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

Classes

- 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.
- 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.
- 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.

Enumerations

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

5.4.1 Detailed Description

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

5.5 FATime Namespace Reference

Has [Time](#) class.

Classes

- class [Time](#)

This class is used to get the time between each frame. You can stop start, reset and get the total time.

5.5.1 Detailed Description

Has [Time](#) class.

5.6 FAWindow Namespace Reference

Has [Window](#) class.

Classes

- class [Window](#)

The window class is used to make a [Window](#) using Windows API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

5.6.1 Detailed Description

Has [Window](#) class.

Chapter 6

Class Documentation

6.1 FACamera::Camera Class 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 "FACamera.h"
```

Public Member Functions

- [Camera](#) ()
Creates a new camera. Initializes all properties to 0.
- [Camera](#) (vec3 cameraPosition, vec3 x, vec3 y, vec3 z, float cameraVelocity, float angularVelocity)
Creates a new camera.
- void [SetProperties](#) (vec3 cameraPosition, vec3 x, vec3 y, vec3 z, float cameraVelocity, float angularVelocity)
Sets the properties of the camera.
- const vec3 & [GetCameraPosition](#) () const
Returns a constant reference to the position of the camera in world coordinates.
- const vec3 & [GetX](#) () const
Returns a constant reference to the x-axis of the camera.
- const vec3 & [GetY](#) () const
Returns a constant reference to the y-axis of the camera.
- const vec3 & [GetZ](#) () const
Returns a constant reference to the z-axis of the camera.
- const mat4 & [GetViewMatrix](#) () const
Returns a constant reference to the view transformation matrix of this camera.
- float [GetCameraVelocity](#) () const
Returns the camera's velocity.
- float [GetAngularVelocity](#) () const
Returns the camera's angular velocity.
- void [LookAt](#) (vec3 cameraPosition, vec3 target, vec3 up)
Defines the camera space using UVN.
- void [SetCameraPosition](#) (const vec3 &position)

- Sets the camera's position to the specified position.*
 - void [SetX](#) (const vec3 &x)
- Sets the camera's x-axis to the specified vector x.*
 - void [SetY](#) (const vec3 &y)
- Sets the camera's y-axis to the specified vector y.*
 - void [SetZ](#) (const vec3 &z)
- Sets the camera's z-axis to the specified vector z.*
 - void [SetCameraVelocity](#) (float velocity)
- Sets the camera's velocity to the specified velocity.*
 - void [SetAngularVelocity](#) (float velocity)
- Sets the camera's angular velocity to the specified angular velocity.*
 - void [UpdateViewMatrix](#) ()
- After modifying the camera position and/or orientation, call this to rebuild the view transformation matrix.*
 - void [Left](#) (float dt)
- Moves the camera left along the camera's x-axis.*
 - void [Right](#) (float dt)
- Moves the camera right along the camera's x-axis.*
 - void [Forward](#) (float dt)
- Moves the camera forward along the camera's z-axis.*
 - void [Backward](#) (float dt)
- Moves the camera backward along the camera's z-axis.*
 - void [Up](#) (float dt)
- Moves the camera up along the camera's y-axis.*
 - void [Down](#) (float dt)
- Moves the camera down along the camera's y-axis.*
 - void [RotateCameraLeftRight](#) (float xDiff)
- Rotates the camera to look left and right.*
 - void [RotateCameraUpDown](#) (float yDiff)
- Rotates the camera to look up and down.*
 - void [KeyboardInput](#) (float dt)
- Polls keyboard input and moves the camera.*
 - void [KeyboardInputWASD](#) (float dt)
- Polls keyboard input and moves the camera.*
 - void [KeyboardInputArrow](#) (float dt)
- Polls keyboard input and moves the camera.*
 - void [MouseInput](#) ()
- Polls mouse input and rotates the camera.*

6.1.1 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.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 Camera() [1/2]

```
FACamera::Camera::Camera ( )
```

Creates a new camera. Initializes all properties to 0.

6.1.2.2 Camera() [2/2]

```
FACamera::Camera::Camera (
    vec3 cameraPosition,
    vec3 x,
    vec3 y,
    vec3 z,
    float cameraVelocity,
    float angularVelocity )
```

Creates a new camera.

Parameters

in	<i>camerPosition</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>cameraVelocity</i>	The translational velocity of the camera.
in	<i>angularVelocity</i>	The angular velocity of the camera.

6.1.3 Member Function Documentation**6.1.3.1 Backward()**

```
void FACamera::Camera::Backward (
    float dt )
```

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

Parameters

in	<i>dt</i>	The time between frames.
----	-----------	--------------------------

6.1.3.2 Down()

```
void FACamera::Camera::Down (
    float dt )
```

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

Parameters

in	<i>dt</i>	The time between frames.
----	-----------	--------------------------

6.1.3.3 Foward()

```
void FACamera::Camera::Foward (
    float dt )
```

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

Parameters

in	<i>dt</i>	The time between frames.
----	-----------	--------------------------

6.1.3.4 GetAngularVelocity()

```
float FACamera::Camera::GetAngularVelocity ( ) const
```

Returns the camera's angular velocity.

6.1.3.5 GetCameraPosition()

```
const vec3 & FACamera::Camera::GetCameraPosition ( ) const
```

Returns a constant reference to the position of the camera in world coordinates.

6.1.3.6 GetCameraVelocity()

```
float FACamera::Camera::GetCameraVelocity ( ) const
```

Returns the camera's velocity.

6.1.3.7 GetViewMatrix()

```
const mat4 & FACamera::Camera::GetViewMatrix ( ) const
```

Returns a constant reference to the view transformation matrix of this camera.

6.1.3.8 GetX()

```
const vec3 & FACamera::Camera::GetX ( ) const
```

Returns a constant reference to the x-axis of the camera.

6.1.3.9 GetY()

```
const vec3 & FACamera::Camera::GetY ( ) const
```

Returns a constant reference to the y-axis of the camera.

6.1.3.10 GetZ()

```
const vec3 & FACamera::Camera::GetZ ( ) const
```

Returns a constant reference to the z-axis of the camera.

6.1.3.11 KeyboardInput()

```
void FACamera::Camera::KeyboardInput (
    float dt )
```

Polls keyboard input and moves the camera.

Moves the camera forward/backward if w/s or up/down arrow was pressed.

Moves the camera left/right if a/d or left/right arrow was pressed.

Moves the camera up/down if space/crtl was pressed.

Parameters

in	<i>dt</i>	The time between frames.
----	-----------	--------------------------

6.1.3.12 KeyboardInputArrow()

```
void FACamera::Camera::KeyboardInputArrow (
    float dt )
```

Polls keyboard input and moves the camera.

Moves the camera forward/backward if up/down arrow was pressed.

Moves the camera left/right if left/right arrow was pressed.

Moves the camera up/down if space/crtl was pressed.

Parameters

in	dt	The time between frames.
----	----	--------------------------

6.1.3.13 KeyboardInputWASD()

```
void FACamera::Camera::KeyboardInputWASD (
    float dt )
```

Polls keyboard input and moves the camera.

Moves the camera forward/backward if w/s was pressed.

Moves the camera left/right if a/d was pressed.

Moves the camera up/down if space/crtl was pressed.

Parameters

in	dt	The time between frames.
----	----	--------------------------

6.1.3.14 Left()

```
void FACamera::Camera::Left (
    float dt )
```

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

6.1.3.15 LookAt()

```
void FACamera::Camera::LookAt (
    vec3 cameraPosition,
    vec3 target,
    vec3 up )
```

Defines the camera space using UVN.

Parameters

in	<i>camerPosition</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.

6.1.3.16 MouseInput()

```
void FACamera::Camera::MouseInput ( )
```

Polls mouse input and rotates the camera.

6.1.3.17 Right()

```
void FACamera::Camera::Right (
    float dt )
```

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

Parameters

in	<i>dt</i>	The time between frames.
----	-----------	--------------------------

6.1.3.18 RotateCameraLeftRight()

```
void FACamera::Camera::RotateCameraLeftRight (
    float xDiff )
```

Rotates the camera to look left and right.

Parameters

in	<i>xDiff</i>	How many degrees to rotate.
----	--------------	-----------------------------

6.1.3.19 RotateCameraUpDown()

```
void FACamera::Camera::RotateCameraUpDown (
    float yDiff )
```

Rotates the camera to look up and down.

Parameters

in	<i>yDiff</i>	How many degrees to rotate.
----	--------------	-----------------------------

6.1.3.20 SetAngularVelocity()

```
void FACamera::Camera::SetAngularVelocity (
    float velocity )
```

Sets the camera's angular velocity to the specified *angular velocity*.

6.1.3.21 SetCameraPosition()

```
void FACamera::Camera::SetCameraPosition (
    const vec3 & position )
```

Sets the camera's position to the specified position.

6.1.3.22 SetCameraVelocity()

```
void FACamera::Camera::SetCameraVelocity (
    float velocity )
```

Sets the camera's velocity to the specified *velocity*.

6.1.3.23 SetProperties()

```
void FACamera::Camera::SetProperties (
    vec3 cameraPosition,
    vec3 x,
    vec3 y,
    vec3 z,
    float cameraVelocity,
    float angularVelocity )
```

Sets the properties of the camera.

Parameters

in	<i>camerPosition</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>cameraVelocity</i>	The translational velocity of the camera.
in	<i>angularVelocity</i>	The angular velocity of the camera.

6.1.3.24 SetX()

```
void FACamera::Camera::SetX (
    const vec3 & x )
```

Sets the camera's x-axis to the specified vector *x*.

6.1.3.25 SetY()

```
void FACamera::Camera::SetY (
    const vec3 & y )
```

Sets the camera's y-axis to the specified vector *y*.

6.1.3.26 SetZ()

```
void FACamera::Camera::SetZ (
    const vec3 & z )
```

Sets the camera's z-axis to the specified vector *z*.

6.1.3.27 Up()

```
void FACamera::Camera::Up (
    float dt )
```

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

Parameters

<i>in</i>	<i>dt</i>	The time between frames.
-----------	-----------	--------------------------

6.1.3.28 UpdateViewMatrix()

```
void FACamera::Camera::UpdateViewMatrix ( )
```

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

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

- [FACamera.h](#)

6.2 FAColor::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 "FAColor.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 FAMath::Vector4D &color)
Initializes the color to the specified color.
- const FAMath::Vector4D & [GetColor](#) () const
Returns the color.
- float [GetRed](#) () const
Returns the value of the red component.
- float [GetGreen](#) () const
Returns the value of the blue component.
- float [GetBlue](#) () const
Returns the value of the green component.
- float [GetAlpha](#) () const
Returns the value of the alpha component.
- void [SetColor](#) (const FAMath::Vector4D &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.

6.2.1 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 componet is red, second component is green, third component is blue and the 4th component is alpha.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 Color() [1/2]

```
FIColor::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.

6.2.2.2 Color() [2/2]

```
FIColor::Color::Color (
    const FIMath::Vector4D & color )
```

Initializes the color to the specified *color*.

6.2.3 Member Function Documentation

6.2.3.1 GetAlpha()

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

Returns the value of the alpha component.

6.2.3.2 GetBlue()

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

Returns the value of the green component.

6.2.3.3 GetColor()

```
const FIMath::Vector4D & FIColor::Color::GetColor ( ) const
```

Returns the color.

6.2.3.4 GetGreen()

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

Returns the value of the blue component.

6.2.3.5 GetRed()

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

Returns the value of the red component.

6.2.3.6 operator*=() [1/2]

```
Color & FColor::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.

6.2.3.7 operator*=() [2/2]

```
Color & FColor::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 $1.0f$.

6.2.3.8 operator+=()

```
Color & FColor::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 $1.0f$.

6.2.3.9 operator-=()

```
Color & FColor::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 $0.0f$.

6.2.3.10 SetAlpha()

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

Sets the alpha component to the specified float value.

6.2.3.11 SetBlue()

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

Sets the blue component to the specified float value.

6.2.3.12 SetColor()

```
void FColor::Color::SetColor (
    const FAMath::Vector4D & color )
```

Sets the color to the specified color.

6.2.3.13 SetGreen()

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

Sets the green component to the specified float value.

6.2.3.14 SetRed()

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

Sets the red component to the specified float value.

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

- [FColor.h](#)

6.3 FRender::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 "FABuffer.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 indexOfView, unsigned int dsvSize, float clearValue)
Clears the depth stencil buffer with the specified clear value.

6.3.1 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.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 DepthStencilBuffer() [1/2]

```
FARender::DepthStencilBuffer::DepthStencilBuffer ( )
```

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

6.3.2.2 DepthStencilBuffer() [2/2]

```
FARender::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>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.
in	<i>sampleCount</i>	The sample count of the depth stencil buffer.

6.3.3 Member Function Documentation

6.3.3.1 ClearDepthStencilBuffer()

```
void FAREnder::DepthStencilBuffer::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.

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.

6.3.3.2 CreateDepthStencilBufferAndView()

```
void FARender::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.

6.3.3.3 GetDepthStencilFormat()

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

Returns the format of the depth stencil buffer.

6.3.3.4 ReleaseBuffer()

```
void FARender::DepthStencilBuffer::ReleaseBuffer ( )
```

Frees the memory of the buffer.

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

- [FABuffer.h](#)

6.4 FAREnder::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 "FADeviceResources.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.

6.4.1 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.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 ~DeviceResources()

```
FARender::DeviceResources::~~DeviceResources ( )
```

Flushes the command queue.

6.4.3 Member Function Documentation

6.4.3.1 AfterTextDraw()

```
void FARender::DeviceResources::AfterTextDraw ( )
```

Executes the text commands.

6.4.3.2 BeforeTextDraw()

```
void FARender::DeviceResources::BeforeTextDraw ( )
```

Prepares to render text.

6.4.3.3 Execute()

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

Executes the command list.

6.4.3.4 FlushCommandQueue()

```
void FAREnder::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.

6.4.3.5 GetBackBufferFormat()

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

Returns a constant reference to the back buffer format.

6.4.3.6 GetCBVSRVUAVSize()

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

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

6.4.3.7 GetCommandList()

```
const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & FAREnder::DeviceResources::Get←  
CommandList ( ) const
```

Returns a constant reference to the ID3D12GraphicsCommandList object.

6.4.3.8 GetCurrentFrame()

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

Returns the current frame.

6.4.3.9 GetDepthStencilFormat()

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

Returns a constant reference to the depth stencil format.

6.4.3.10 GetDevice()

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

Returns a constant reference to the ID3D12Device object.

6.4.3.11 GetInstance()

```
static DeviceResources & FARender::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.

6.4.3.12 GetTextResources()

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

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

6.4.3.13 NextFrame()

```
void FAREnder::DeviceResources::NextFrame ( )
```

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

6.4.3.14 Present()

```
void FAREnder::DeviceResources::Present ( )
```

Swaps the front and back buffers.

6.4.3.15 Resize()

```
void FAREnder::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.

6.4.3.16 RTBufferTransition()

```
void FAREnder::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.

6.4.3.17 Signal()

```
void FARender::DeviceResources::Signal ( )
```

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

6.4.3.18 UpdateCurrentFrameFenceValue()

```
void FARender::DeviceResources::UpdateCurrentFrameFenceValue ( )
```

Updates the current frames fence value.

6.4.3.19 WaitForGPU()

```
void FARender::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.

6.4.4 Member Data Documentation

6.4.4.1 NUM_OF_FRAMES

```
const unsigned int FARender::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:

- [FADeviceResources.h](#)

6.5 DirectXException Class Reference

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

```
#include "FADirectXException.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.

6.5.1 Detailed Description

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

6.5.2 Constructor & Destructor Documentation

6.5.2.1 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.

6.5.3 Member Function Documentation

6.5.3.1 ErrorMsg()

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

Returns a message describing the error.

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

- FADirectXException.h

6.6 FAREnder::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 "FABuffer.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 numOfBytes, 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 numOfBytes, 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 cbvHeapIndex, 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.

6.6.1 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.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 DynamicBuffer() [1/3]

```
FARender::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.

6.6.2.2 DynamicBuffer() [2/3]

```
FARender::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.

6.6.2.3 DynamicBuffer() [3/3]

```
FARender::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.

6.6.2.4 ~DynamicBuffer()

```
FARender::DynamicBuffer::~~DynamicBuffer ( )
```

Unmaps the pointer to the dynamic buffer.

6.6.3 Member Function Documentation

6.6.3.1 CopyData()

```
void FARender::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.

6.6.3.2 CreateConstantBufferView()

```
void FARender::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.

6.6.3.3 CreateDynamicBuffer() [1/2]

```
void FAREnder::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.

6.6.3.4 CreateDynamicBuffer() [2/2]

```
void FAREnder::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.

6.6.3.5 GetGPUAddress()

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

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

6.6.3.6 GetIndexBufferView()

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

Returns the index buffer view of the dynamic buffer.

6.6.3.7 GetVertexBufferView()

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

Returns a the vertex buffer view of the dynamic buffer.

6.6.3.8 ReleaseBuffer()

```
void FARender::DynamicBuffer::ReleaseBuffer ( )
```

Frees the dynamic buffer memory.

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

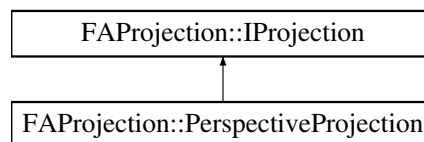
- [FABuffer.h](#)

6.7 FAProjection::IProjection Class Reference

An interface for projections.

```
#include "FAProjection.h"
```

Inheritance diagram for FAProjection::IProjection:



Public Member Functions

- [IProjection](#) ()
Default Constructor.
- const FAMath::Matrix4x4 & [GetProjectionMatrix](#) () const
Returns the projection matrix.
- virtual void [UpdateProjectionMatrix](#) ()=0

Protected Attributes

- FAMath::Matrix4x4 **mProjectionMatrix**
- bool **mUpdateProjectionMatrix**

6.7.1 Detailed Description

An interface for projections.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 IProjection()

```
FAProjection::IProjection::IProjection ( )
```

Default Constructor.

6.7.3 Member Function Documentation

6.7.3.1 GetProjectionMatrix()

```
const FAMath::Matrix4x4 & FAProjection::IProjection::GetProjectionMatrix ( ) const
```

Returns the projection matrix.

6.7.3.2 UpdateProjectionMatrix()

```
virtual void FAProjection::IProjection::UpdateProjectionMatrix ( ) [pure virtual]
```

Implemented in [FAProjection::PerspectiveProjection](#).

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

- [FAProjection.h](#)

6.8 FARender::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 "FAMultiSampling.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.

6.8.1 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.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 MultiSampling() [1/2]

```
FARender::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.

6.8.2.2 MultiSampling() [2/2]

```
FARender::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.

6.8.3 Member Function Documentation

6.8.3.1 CheckMultiSamplingSupport()

```
void FAREnder::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.

6.8.3.2 ClearDepthStencilBuffer()

```
void FARender::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.

6.8.3.3 ClearRenderTargetBuffer()

```
void FARender::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.

6.8.3.4 CreateDepthStencilBufferAndView()

```
void FARender::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.

6.8.3.5 CreateRenderTargetBufferAndView()

```

void FAREnder::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.

6.8.3.6 GetDepthStencilFormat()

```

DXGI_FORMAT FAREnder::MultiSampling::GetDepthStencilFormat ( )

```

Returns the format of the MSAA depth stencil buffer.

6.8.3.7 GetRenderTargetBuffer()

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

Returns a constant reference to the MSAA render target buffer.

6.8.3.8 GetRenderTargetFormat()

```
DXGI_FORMAT FARender::MultiSampling::GetRenderTargetFormat ( )
```

Returns the format of the MSAA render target buffer.

6.8.3.9 ReleaseBuffers()

```
void FARender::MultiSampling::ReleaseBuffers ( )
```

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

6.8.3.10 Transition()

```
void FARender::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:

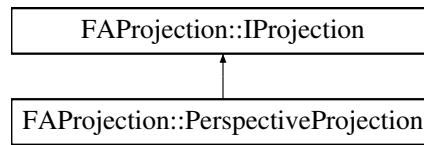
- FAMultiSampling.h

6.9 FAProjection::PerspectiveProjection Class Reference

A class for doing perspective projection.

```
#include "FAProjection.h"
```

Inheritance diagram for FAProjection::PerspectiveProjection:



Public Member Functions

- [PerspectiveProjection](#) ()
Initializes all properties to 0.0f.
- [PerspectiveProjection](#) (float znear, float zfar, float vFov, float aspectRatio)
Initializes all the properties to the specified values.
- void [SetProperties](#) (float znear, float zfar, float vFov, float aspectRatio)
Sets all the properties to the specified values.
- float [GetNear](#) () const
Returns the near value of the frustrum.
- float [GetFar](#) () const
Returns the far value of the frustrum.
- float [GetVerticalFov](#) () const
Returns the vertical field of view of the frustrum in degrees.
- float [GetAspectRatio](#) () const
Returns the aspect ratio of the frustrum.
- void [SetNear](#) (float near)
Sets the camera's near plane value to the specified value.
- void [SetFar](#) (float far)
Sets the camera's far plane value to the specified value.
- void [SetVerticalFov](#) (float fov)
Sets the camera's vertical field of view to the specified vertical field of view .
- void [SetAspectRatio](#) (float ar)
Sets the camera's aspect ratio to the specified aspect ratio.
- void [UpdateProjectionMatrix](#) () override final
Updates the projection matrix if any of the properties have changed.

Additional Inherited Members

6.9.1 Detailed Description

A class for doing perspective projection.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 PerspectiveProjection() [1/2]

```
FAProjection::PerspectiveProjection::PerspectiveProjection ( )
```

Initializes all properties to 0.0f.

6.9.2.2 PerspectiveProjection() [2/2]

```
FAProjection::PerspectiveProjection::PerspectiveProjection (
    float zNear,
    float zFar,
    float vFov,
    float aspectRatio )
```

Initializes all the properties to the specified values.

Parameters

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.

6.9.3 Member Function Documentation

6.9.3.1 GetAspectRatio()

```
float FAProjection::PerspectiveProjection::GetAspectRatio ( ) const
```

Returns the aspect ratio of the frustrum.

6.9.3.2 GetFar()

```
float FAProjection::PerspectiveProjection::GetFar ( ) const
```

Returns the far value of the frustrum.

6.9.3.3 GetNear()

```
float FAProjection::PerspectiveProjection::GetNear ( ) const
```

Returns the near value of the frustum.

6.9.3.4 GetVerticalFov()

```
float FAProjection::PerspectiveProjection::GetVerticalFov ( ) const
```

Returns the vertical field of view of the frustum in degrees.

6.9.3.5 SetAspectRatio()

```
void FAProjection::PerspectiveProjection::SetAspectRatio (
    float ar )
```

Sets the camera's aspect ratio to the specified aspect ratio.

6.9.3.6 SetFar()

```
void FAProjection::PerspectiveProjection::SetFar (
    float far )
```

Sets the camera's far plane value to the specified value.

6.9.3.7 SetNear()

```
void FAProjection::PerspectiveProjection::SetNear (
    float near )
```

Sets the camera's near plane value to the specified value.

6.9.3.8 SetProperties()

```
void FAProjection::PerspectiveProjection::SetProperties (
    float znear,
    float zfar,
    float vFov,
    float aspectRatio )
```

Sets all the properties to the specified values.

Parameters

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.

6.9.3.9 SetVerticalFov()

```
void FAProjection::PerspectiveProjection::SetVerticalFov (
    float fov )
```

Sets the camera's vertical field of view to the specified vertical field of view .

6.9.3.10 UpdateProjectionMatrix()

```
void FAProjection::PerspectiveProjection::UpdateProjectionMatrix ( ) [final], [override],
[virtual]
```

Updates the projection matrix if any of the properties have changed.

Implements [FAProjection::IProjection](#).

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

- [FAProjection.h](#)

6.10 FARender::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 "FARenderScene.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, const std::wstring &filename)
Loads a shaders bytecode and maps it to the specified shaderKey.
- void **CompileShader** (unsigned int shaderKey, const std::wstring &filename, const std::string &entryPointName, const std::string &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 **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 **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 **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 **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)
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 **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 **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 [SetPSOAndRootSignature](#) (unsigned int psoKey, unsigned int rootSigKey)

Sets the PSO and its associated root signature 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 [SetStaticBuffer](#) (unsigned int bufferType, unsigned int staticBufferKey)

Links the static buffer mapped to the static buffer key to the pipeline.
 - void [SetDynamicBuffer](#) (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 [SetTextureViewHeap](#) ()
 - void [SetTexture](#) (unsigned int rootParameterIndex, unsigned int textureViewIndex)

Links a texture to the pipeline.
 - void [SetConstants](#) (unsigned int rootParameterIndex, unsigned int numValues, void *data, unsigned int index)

Links an array of 32-bit values 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 FAMath::Vector4D &textLocation, const [FAColor::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 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 [CopyDataIntoDynamicBuffer](#) (unsigned int dynamicBufferKey, unsigned int index, const void *data, UINT64 numBytes)

Copies the specified data into the dyanmic buffer mapped to the dynamic buffer key.
 - void [CreateTextureViewHeap](#) (unsigned int numDescriptors)

Creates a descriptor heap to store views of textures.
 - 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 an array mapped to the specified key.

6.10.1 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.

6.10.2 Member Function Documentation

6.10.2.1 AfterRender()

```
void FARender::RenderScene::AfterRender ( )
```

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

Call after rendering all your objects and text.

6.10.2.2 AfterRenderObjects()

```
void FARender::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.

6.10.2.3 AfterRenderText()

```
void FARender::RenderScene::AfterRenderText ( )
```

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

Call after calling all the RenderText functions.

6.10.2.4 BeforeRenderObjects()

```
void FARender::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.
----------------------	--

6.10.2.5 BeforeRenderText()

```
void FARender::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.

6.10.2.6 CompileShader()

```
void FARender::RenderScene::CompileShader (
    unsigned int shaderKey,
    const std::wstring & filename,
    const std::string & entryPointName,
    const std::string & 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.

6.10.2.7 CopyDataIntoDynamicBuffer()

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

Copies the specified data into the dynamic 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.

6.10.2.8 CreateDescriptorRange()

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

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.

6.10.2.9 CreateDescriptorTable()

```
void FARender::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.

6.10.2.10 CreateDynamicBuffer() [1/2]

```
void FARender::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.

6.10.2.11 CreateDynamicBuffer() [2/2]

```
void FARender::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.

6.10.2.12 CreateInputElementDescription()

```
void FARender::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.
----	------------	---

Parameters

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>	inputSlotClass The data class for an input slot. Used for instancing.
	<i>[in,optional]</i>	instanceStepRate The number of instances to render. Used for instancing.

6.10.2.13 CreatePSO()

```
void FARender::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.

6.10.2.14 CreateRootConstants()

```
void FARender::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.

6.10.2.15 CreateRootDescriptor()

```

void FARender::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.

6.10.2.16 CreateRootSignature() [1/2]

```

void FARender::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.

6.10.2.17 CreateRootSignature() [2/2]

```

void FARender::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.

6.10.2.18 CreateStaticBuffer() [1/3]

```

void FAREnder::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.

6.10.2.19 CreateStaticBuffer() [2/3]

```

void FAREnder::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.

6.10.2.20 CreateStaticBuffer() [3/3]

```
void FARender::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 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.

6.10.2.21 CreateStaticSampler()

```
void FARender::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 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.

6.10.2.22 CreateTextureViewHeap()

```
void FARender::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.
----	-----------------------	---

6.10.2.23 ExecuteAndFlush()

```
void FARender::RenderScene::ExecuteAndFlush ( )
```

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

6.10.2.24 LoadShader()

```
void FARender::RenderScene::LoadShader (
    unsigned int shaderKey,
    const std::wstring & 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.

6.10.2.25 RemoveShader()

```
void FARender::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.

6.10.2.26 RenderObject()

```
void FARender::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.

6.10.2.27 RenderText()

```
void FARender::RenderScene::RenderText (
    const FAMath::Vector4D & textLocation,
    const FIColor::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 After↵RenderText 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.

6.10.2.28 Resize()

```
void FAREnder::RenderScene::Resize (
    unsigned int width,
    unsigned int height,
    HWND windowHandle,
    bool isMSAAEnabled = false,
    bool isEnabled = 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>	isEnabled Pass in true if text enabled, false otherwise.

6.10.2.29 SetConstants()

```
void FAREnder::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.

6.10.2.30 SetDynamicBuffer()

```
void FAREnder::RenderScene::SetDynamicBuffer (
    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

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

in	<i>dynamicBufferKey</i>	The key mapped to a dynamic buffer.
	<i>[in,optional]</i>	<i>indexConstantData</i> The index of where the constant data is in the dynamic buffer.
	<i>[in,optional]</i>	<i>rootParameterIndex</i> 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.

6.10.2.31 SetPSOAndRootSignature()

```

void FARender::RenderScene::SetPSOAndRootSignature (
    unsigned int psoKey,
    unsigned int rootSigKey )

```

Sets the PSO and its associated root signature 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.

6.10.2.32 SetStaticBuffer()

```

void FARender::RenderScene::SetStaticBuffer (
    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 runtime_error 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.

6.10.2.33 SetTexture()

```
void FARender::RenderScene::SetTexture (
    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.

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

- [FARenderScene.h](#)

6.11 FARender::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 "FABuffer.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 index, unsigned int rtvSize, const float *clearValue)`
Clears the render target buffer with the specified clear value.

6.11.1 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.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 RenderTargetBuffer() [1/2]

```
FARender::RenderTargetBuffer::RenderTargetBuffer ( )
```

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

6.11.2.2 RenderTargetBuffer() [2/2]

```
FARender::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.

6.11.3 Member Function Documentation

6.11.3.1 ClearRenderTargetBuffer()

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

Clears the render target buffer with the specified clear value.

Parameters

in	<i>commandList</i>	A Direct3D 12 graphics command list.
in	<i>rtvHeap</i>	A render target descriptor heap.
in	<i>indexInView</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.

6.11.3.2 CreateRenderTargetBufferAndView()

```
void FAREnder::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.

6.11.3.3 GetRenderTargetBuffer() [1/2]

```
Microsoft::WRL::ComPtr< ID3D12Resource > & FARender::RenderTargetBuffer::GetRenderTargetBuffer
( )
```

Returns a reference to the render target buffer.

6.11.3.4 GetRenderTargetBuffer() [2/2]

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

Returns a constant reference to the render target buffer.

6.11.3.5 GetRenderTargetFormat()

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

Returns the format of the render target buffer.

6.11.3.6 ReleaseBuffer()

```
void FARender::RenderTargetBuffer::ReleaseBuffer ( )
```

Frees the memory of the buffer.

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

- [FABuffer.h](#)

6.12 FAREnder::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 "FABuffer.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.

6.12.1 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.

6.12.2 Constructor & Destructor Documentation

6.12.2.1 StaticBuffer() [1/4]

```
FARender::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.

6.12.2.2 StaticBuffer() [2/4]

```
FARender::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.

6.12.2.3 StaticBuffer() [3/4]

```
FARender::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.

6.12.2.4 StaticBuffer() [4/4]

```
FARender::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.

6.12.3 Member Function Documentation

6.12.3.1 CreateStaticBuffer() [1/3]

```
void FARender::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.

6.12.3.2 CreateStaticBuffer() [2/3]

```
void FARender::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.

6.12.3.3 CreateStaticBuffer() [3/3]

```

void FAREnder::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.

6.12.3.4 CreateTexture2DMSView()

```

void FAREnder::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.

6.12.3.5 CreateTexture2DView()

```
void FARender::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.

6.12.3.6 GetIndexBufferView()

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

Returns the index buffer view of the static buffers.

6.12.3.7 GetVertexBufferView()

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

Returns the vertex buffer view of the static buffer.

6.12.3.8 ReleaseBuffer()

```
void FARender::StaticBuffer::ReleaseBuffer ( )
```

Frees the static buffer memory.

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

- [FABuffer.h](#)

6.13 FAREnder::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 "FASwapChain.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 width, unsigned 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.

6.13.1 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.

6.13.2 Constructor & Destructor Documentation

6.13.2.1 SwapChain() [1/2]

```
FARender::SwapChain::SwapChain ( )
```

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

6.13.2.2 SwapChain() [2/2]

```
FARender::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.

6.13.3 Member Function Documentation

6.13.3.1 ClearCurrentBackBuffer()

```
void FARender::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.

6.13.3.2 ClearDepthStencilBuffer()

```
void FARender::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.

6.13.3.3 CreateDepthStencilBufferAndView()

```
void FARender::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 stencil descriptor.
in	<i>width</i>	The width of the depth stencil buffer.
in	<i>height</i>	The height of the depth stencil buffer.

6.13.3.4 CreateRenderTargetBuffersAndViews()

```
void FARender::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.
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.

6.13.3.5 CreateSwapChain()

```
void FARender::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.

6.13.3.6 GetBackBufferFormat()

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

Returns the format of the swap chain.

6.13.3.7 GetCurrentBackBuffer()

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

Returns a constant reference to the current render target buffer.

6.13.3.8 GetCurrentBackBufferIndex()

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

Returns the current back buffer index.

6.13.3.9 GetDepthStencilFormat()

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

Returns the format of the depth stencil buffer.

6.13.3.10 GetNumRenderTargetBuffers()

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

Returns the number of swap chain buffers.

6.13.3.11 Present()

```
void FAREnder::SwapChain::Present ( )
```

Swaps the front and back buffers.

6.13.3.12 ReleaseBuffers()

```
void FAREnder::SwapChain::ReleaseBuffers ( )
```

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

6.13.3.13 Transition()

```
void FAREnder::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	<i>commandList</i>	A Direct3D 12 graphics command list.
----	--------------------	--------------------------------------

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

- FASwapChain.h

6.14 FAREnder::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 "FAText.h"
```

Public Member Functions

- [Text](#) (const FAMath::Vector4D &textLocation, const std::wstring &textString, float textSize, const [FAColor::Color](#) &textColor)
Constructs a [Text](#) object.
- const FAMath::Vector4D & [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 [FAColor::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 [FAColor::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 FAMath::Vector4D &textLocation)
Changes the text location to the specified textLocation.

6.14.1 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.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 Text()

```
FARender::Text::Text (
    const FAMath::Vector4D & textLocation,
    const std::wstring & textString,
    float textSize,
    const FAColor::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.

6.14.3 Member Function Documentation

6.14.3.1 GetTextColor()

```
const FColor::Color & FARender::Text::GetTextColor ( ) const
```

Returns a constant reference to the text color.

6.14.3.2 GetTextLocation()

```
const FMath::Vector4D & FARender::Text::GetTextLocation ( ) const
```

Returns a constant reference to the text location.

6.14.3.3 GetTextSize()

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

Returns the text size.

6.14.3.4 GetTextString()

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

Returns a constant reference to the text string.

6.14.3.5 SetTextColor()

```
void FARender::Text::SetTextColor (
    const FColor::Color & textColor )
```

Changes the text color to the specified *textColor*.

6.14.3.6 SetTextLocation()

```
void FARender::Text::SetTextLocation (
    const FAMath::Vector4D & textLocation )
```

Changes the text location to the specified *textLocation*.

6.14.3.7 SetTextSize()

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

Changes the text size to the specified *textSize*.

6.14.3.8 SetTextString()

```
void FARender::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:

- [FAText.h](#)

6.15 FARender::TextResources Class Reference

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

```
#include "FATextResources.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.

6.15.1 Detailed Description

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

6.15.2 Constructor & Destructor Documentation

6.15.2.1 TextResources()

```
FARender::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.

6.15.3 Member Function Documentation

6.15.3.1 AfterRenderText()

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

Executes text commands.

Parameters

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

6.15.3.2 BeforeRenderText()

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

Prepares to render text.

Parameters

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

6.15.3.3 GetDirect2DDeviceContext()

```
const Microsoft::WRL::ComPtr< ID2D1DeviceContext > & FARender::TextResources::GetDirect2DDeviceContext ( ) const
```

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

6.15.3.4 GetDirectWriteFactory()

```
const Microsoft::WRL::ComPtr< IDWriteFactory > & FARender::TextResources::GetDirectWriteFactory ( ) const
```

Returns a constant reference to the direct write factory.

6.15.3.5 ResetBuffers()

```
void FARender::TextResources::ResetBuffers ( )
```

Resets the text buffers.

6.15.3.6 ResizeBuffers()

```
void FARender::TextResources::ResizeBuffers (
    const std::vector< std::unique_ptr< RenderTargetBuffer > > & renderTargetBuffers,
    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:

- [FATextResources.h](#)

6.16 FATime::Time Class Reference

This class is used to get the time between each frame. You can stop start, reset and get the total time.

```
#include "FATime.h"
```

Public Member Functions

- [Time](#) ()
Gets and stores the seconds per count.
- void [Tick](#) ()
Stores the difference between the current time and the previous time.
- float [DeltaTime](#) () const
Returns the difference between the current time and the previous time.
- void [Reset](#) ()
Resets all time variables.
- void [Stop](#) ()
Stops the timer.
- void [Start](#) ()
Starts the timer.
- float [TotalTime](#) () const
Returns how much time has passed since [Reset\(\)](#) was called. Does not count any pause time.

6.16.1 Detailed Description

This class is used to get the time between each frame. You can stop start, reset and get the total time.

6.16.2 Constructor & Destructor Documentation

6.16.2.1 Time()

```
FATime::Time::Time ( )
```

Gets and stores the seconds per count.

6.16.3 Member Function Documentation

6.16.3.1 DeltaTime()

```
float FATime::Time::DeltaTime ( ) const
```

Returns the difference between the current time and the previous time.

6.16.3.2 Reset()

```
void FATime::Time::Reset ( )
```

Resets all time variables.

6.16.3.3 Start()

```
void FATime::Time::Start ( )
```

Starts the timer.

6.16.3.4 Stop()

```
void FATime::Time::Stop ( )
```

Stops the timer.

6.16.3.5 Tick()

```
void FATime::Time::Tick ( )
```

Stores the difference between the current time and the previous time.

6.16.3.6 TotalTime()

```
float FATime::Time::TotalTime ( ) const
```

Returns how much time has passed since [Reset\(\)](#) was called. Does not count any pause time.

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

- [FATime.h](#)

6.17 FAWindow::Window Class Reference

The window class is used to make a [Window](#) using Windows API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

```
#include "FAWindow.h"
```

Public Member Functions

- **Window** (const [Window](#) &)=delete
- **Window & operator=** (const [Window](#) &)=delete
- **Window** ()
Creates a [Window](#) object. No window is created. Call [CreateParentWindow\(\)](#) or [CreateChildWindow\(\)](#) or [CreateControlWindow\(\)](#) to create a window.
- **Window** (const HINSTANCE &hInstance, WNDPROC windowProcedure, const [FAColor::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.
- **Window** (const HINSTANCE &hInstance, HWND parent, unsigned int identifier, WNDPROC windowProcedure, const [FAColor::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.
- **Window** (const HINSTANCE &hInstance, HWND parent, unsigned 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.
- void **CreateParentWindow** (const HINSTANCE &hInstance, WNDPROC windowProcedure, const [FAColor::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** (const HINSTANCE &hInstance, HWND parent, unsigned int identifier, WNDPROC windowProcedure, const [FAColor::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** (const HINSTANCE &hInstance, HWND parent, unsigned 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.
- HWND **GetWindowHandle** () const
Returns the window handle.
- unsigned int **GetWidth** () const
Returns the width of the window.
- unsigned int **GetHeight** () const
Returns the height of the window.
- unsigned int **GetX** () const
Returns the x position of the top left corner of the window.
- unsigned int **GetY** () const
Returns the y position of the top left corner of the window.
- void **SetWidth** (unsigned int width)

- Sets the width of the window to the specified width.*
 • void [SetHeight](#) (unsigned int height)
- Sets the height of the window o the specified height.*
 • void [SetX](#) (unsigned int x)
- Sets the x position of the top left corner of the window.*
 • void [SetY](#) (unsigned int y)
- Sets the y position of the top left corner of the window.*

6.17.1 Detailed Description

The window class is used to make a [Window](#) using Windows API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

6.17.2 Constructor & Destructor Documentation

6.17.2.1 Window() [1/4]

```
FAWindow::Window::Window ( )
```

Creates a [Window](#) object. No window is created. Call [CreateParentWindow\(\)](#) or [CreateChildWindow\(\)](#) or [CreateControlWindow\(\)](#) to create a window.

6.17.2.2 Window() [2/4]

```
FAWindow::Window::Window (
    const HINSTANCE & hInstance,
    WNDPROC windowProcedure,
    const FAColor::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>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.

Parameters

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>	parent a handle to a parent. Set to nullptr if it is not a child window.
	<i>[in,optional]</i>	additionalData A pointer to data to access in the window procedure.

6.17.2.3 Window() [3/4]

```
FAWindow::Window::Window (
    const HINSTANCE & hInstance,
    HWND parent,
    unsigned int identifier,
    WNDPROC windowProcedure,
    const FAColor::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>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..

Parameters

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.

6.17.2.4 Window() [4/4]

```
FAWindow::Window::Window (
    const HINSTANCE & hInstance,
    HWND parent,
    unsigned 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>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.

6.17.3 Member Function Documentation**6.17.3.1 CreateChildWindow()**

```
void FAWindow::Window::CreateChildWindow (
    const HINSTANCE & hInstance,
```



```

    HWND parent,
    unsigned int identifier,
    WNDPROC windowProcedure,
    const FColor::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>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.

6.17.3.2 CreateControlWindow()

```

void FAWindow::Window::CreateControlWindow (
    const HINSTANCE & hInstance,
    HWND parent,
    unsigned 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>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.

6.17.3.3 CreateParentWindow()

```
void FAWindow::Window::CreateParentWindow (
    const HINSTANCE & hInstance,
    WNDPROC windowProcedure,
    const FAColor::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>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>	parent a handle to a parent. Set to nullptr if it is not a child window.
	<i>[in,optional]</i>	additionalData A pointer to data to access in the window procedure.

6.17.3.4 GetHeight()

```
unsigned int FAWindow::Window::GetHeight ( ) const
```

Returns the height of the window.

6.17.3.5 GetWidth()

```
unsigned int FAWindow::Window::GetWidth ( ) const
```

Returns the width of the window.

6.17.3.6 GetWindowHandle()

```
HWND FAWindow::Window::GetWindowHandle ( ) const
```

Returns the window handle.

6.17.3.7 GetX()

```
unsigned int FAWindow::Window::GetX ( ) const
```

Returns the x position of the top left corner of the window.

6.17.3.8 GetY()

```
unsigned int FAWindow::Window::GetY ( ) const
```

Returns the y position of the top left corner of the window.

6.17.3.9 SetHeight()

```
void FAWindow::Window::SetHeight (
    unsigned int height )
```

Sets the height of the window o the specified *height*.

6.17.3.10 SetWidth()

```
void FAWindow::Window::SetWidth (
    unsigned int width )
```

Sets the width of the window to the specified *width*.

6.17.3.11 SetX()

```
void FAWindow::Window::SetX (
    unsigned int x )
```

Sets the x position of the top left corner of the window.

6.17.3.12 SetY()

```
void FAWindow::Window::SetY (
    unsigned int y )
```

Sets the y position of the top left corner of the window.

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

- [FAWindow.h](#)

Chapter 7

File Documentation

7.1 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 }

```

7.2 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")

```

7.3 FABuffer.h File Reference

File has the classes `RenderTargetBuffer`, `DepthStencilBuffer`, `StaticBuffer` and `DynamicBuffer` under namespace `FARender`.

```

#include <wrl.h>
#include <d3d12.h>

```

Classes

- class `FARender::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 `FARender::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 `FARender::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 `FARender::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.

Namespaces

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

Enumerations

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

7.3.1 Detailed Description

File has the classes RenderTargetBuffer, DepthStencilBuffer, StaticBuffer and DynamicBuffer under namespace [FARender](#).

7.4 FABuffer.h

[Go to the documentation of this file.](#)

```

1 #pragma once
2
3 #include <wrl.h>
4 #include <d3d12.h>
5
6 namespace FAREnder
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         //No copying
15         RenderTargetBuffer(const RenderTargetBuffer&) = delete;
16         RenderTargetBuffer& operator=(const RenderTargetBuffer&) = delete;
17
18         RenderTargetBuffer();
19
20         RenderTargetBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
21             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int index, unsigned int
22             rtvSize,
23             unsigned int width, unsigned int height, DXGI_FORMAT format = DXGI_FORMAT_R8G8B8A8_UNORM,
24             unsigned int sampleCount = 1);
25
26         DXGI_FORMAT GetRenderTargetFormat() const;
27
28         Microsoft::WRL::ComPtr<ID3D12Resource>& GetRenderTargetBuffer();
29
30         const Microsoft::WRL::ComPtr<ID3D12Resource>& GetRenderTargetBuffer() const;
31
32         void CreateRenderTargetBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
33             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int index, unsigned int
34             rtvSize,
35             unsigned int width, unsigned int height, DXGI_FORMAT format = DXGI_FORMAT_R8G8B8A8_UNORM,
36             unsigned int sampleCount = 1);
37
38         void ReleaseBuffer();
39
40         void ClearRenderTargetBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
41             commandList,
42             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int index, unsigned int
43             rtvSize,
44             const float* clearValue);
45
46     private:
47         Microsoft::WRL::ComPtr<ID3D12Resource> mRenderTargetBuffer;
48     };
49
50     class DepthStencilBuffer
51     {
52     public:
53         //No copying
54         DepthStencilBuffer(const DepthStencilBuffer&) = delete;
55         DepthStencilBuffer& operator=(const DepthStencilBuffer&) = delete;
56
57         DepthStencilBuffer();
58     };
59 }
```

```

124     DepthStencilBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
125         const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int index, unsigned
int dsvSize,
126         unsigned int width, unsigned int height, DXGI_FORMAT format = DXGI_FORMAT_D24_UNORM_S8_UINT,
unsigned int sampleCount = 1);
127
130     DXGI_FORMAT GetDepthStencilFormat() const;
131
132     void CreateDepthStencilBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
133         const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int index, unsigned
int dsvSize,
134         unsigned int width, unsigned int height, DXGI_FORMAT format = DXGI_FORMAT_D24_UNORM_S8_UINT,
unsigned int sampleCount = 1);
135
138     void ReleaseBuffer();
139
140     void ClearDepthStencilBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
commandList,
141         const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int indexInView,
unsigned int dsvSize,
142         float clearValue);
143
144 private:
145     Microsoft::WRL::ComPtr<ID3D12Resource> mDepthStencilBuffer;
146 };
147
148 class StaticBuffer
149 {
150 public:
151
152     //No copying
153     StaticBuffer(const StaticBuffer&) = delete;
154     StaticBuffer& operator=(const StaticBuffer&) = delete;
155
156     StaticBuffer();
157
158     StaticBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
159         const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const void* data,
160         unsigned int numBytes, unsigned int stride);
161
162     StaticBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
163         const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const void* data,
164         unsigned int numBytes, DXGI_FORMAT format);
165
166     StaticBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
167         const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const wchar_t*
filename);
168
169     void CreateStaticBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
170         const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const void* data,
171         unsigned int numBytes, unsigned int stride);
172
173     void CreateStaticBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
174         const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const void* data,
175         unsigned int numBytes, DXGI_FORMAT format);
176
177     void CreateStaticBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
178         const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const wchar_t*
filename);
179
180     const D3D12_VERTEX_BUFFER_VIEW GetVertexBufferView() const;
181
182     const D3D12_INDEX_BUFFER_VIEW GetIndexBufferView() const;
183
184     void CreateTexture2DView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
185         const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& srvHeap, unsigned int srvSize, unsigned
int index);
186
187     void CreateTexture2DMSView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
188         const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& srvHeap, unsigned int srvSize, unsigned
int index);
189
190     void ReleaseBuffer();
191
192 private:
193     Microsoft::WRL::ComPtr<ID3D12Resource> mStaticDefaultBuffer;
194     Microsoft::WRL::ComPtr<ID3D12Resource> mStaticUploadBuffer;
195
196     union
197     {
198         unsigned int mStride;
199         DXGI_FORMAT mFormat;
200     };
201 };
202
203 class DynamicBuffer

```



```

334     {
335     public:
336
337         //No copying
338         DynamicBuffer(const DynamicBuffer&) = delete;
339         DynamicBuffer& operator=(const DynamicBuffer&) = delete;
340
341         DynamicBuffer();
342
343         DynamicBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device, unsigned int numBytes,
344             unsigned int stride);
345
346         DynamicBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device, unsigned int numBytes,
347             DXGI_FORMAT format);
348
349         ~DynamicBuffer();
350
351         void CreateDynamicBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device, unsigned int
352             numBytes, unsigned int stride);
353
354         void CreateDynamicBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device, unsigned int
355             numBytes, DXGI_FORMAT format);
356
357         const D3D12_GPU_VIRTUAL_ADDRESS GetGPUAddress(unsigned int index) const;
358
359         void CreateConstantBufferView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
360             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& cbvHeap, unsigned int cbvSize, unsigned
361             int cbvHeapIndex,
362             unsigned int cBufferIndex);
363
364         const D3D12_VERTEX_BUFFER_VIEW GetVertexBufferView();
365
366         const D3D12_INDEX_BUFFER_VIEW GetIndexBufferView();
367
368         void CopyData(unsigned int index, const void* data, unsigned long long numBytes);
369
370         void ReleaseBuffer();
371
372     private:
373         Microsoft::WRL::ComPtr<ID3D12Resource> mDynamicBuffer;
374         BYTE* mMappedData{ nullptr };
375
376         union
377         {
378             {
379                 UINT mStride;
380                 DXGI_FORMAT mFormat;
381             };
382         };
383     };
384 }

```

7.5 FACamera.h File Reference

File that has namespace [FACamera](#). Withn the namespace is the class Camera.

```

#include "FAMathEngine.h"
#include <Windows.h>

```

Classes

- class [FACamera::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.

Namespaces

- namespace [FACamera](#)
Has [Camera](#) class.

Typedefs

- typedef FAMath::Vector2D [vec2](#)
- typedef FAMath::Vector3D **vec3**
- typedef FAMath::Vector4D **vec4**
- typedef FAMath::Matrix4x4 **mat4**

7.5.1 Detailed Description

File that has namespace [FACamera](#). Withn the namespace is the class Camera.

7.5.2 Typedef Documentation

7.5.2.1 [vec2](#)

```
typedef FAMath::Vector2D vec2
```

FACAMERA_H FILE

7.6 FACamera.h

[Go to the documentation of this file.](#)

```
1 #pragma once
2
12 #include "FAMathEngine.h"
13 #include <Windows.h>
14
15 typedef FAMath::Vector2D vec2;
16 typedef FAMath::Vector3D vec3;
17 typedef FAMath::Vector4D vec4;
18 typedef FAMath::Matrix4x4 mat4;
19
23 namespace FACamera
24 {
25     class Camera
26     {
27     public:
28
29         Camera();
30
31         Camera(vec3 cameraPosition, vec3 x, vec3 y, vec3 z, float cameraVelocity, float angularVelocity);
32
33         void SetProperties(vec3 cameraPosition, vec3 x, vec3 y, vec3 z, float cameraVelocity, float
34             angularVelocity);
35
36         const vec3& GetCameraPosition() const;
37
38         const vec3& GetX() const;
39
40         const vec3& GetY() const;
41
42         const vec3& GetZ() const;
43
44         const mat4& GetViewMatrix() const;
45
46         float GetCameraVelocity() const;
47
48         float GetAngularVelocity() const;
49
50         void LookAt(vec3 cameraPosition, vec3 target, vec3 up);
51
52     };
53 }
```

```

97     void SetCameraPosition(const vec3& position);
98
101    void SetX(const vec3& x);
102
105    void SetY(const vec3& y);
106
109    void SetZ(const vec3& z);
110
113    void SetCameraVelocity(float velocity);
114
117    void SetAngularVelocity(float velocity);
118
121    void UpdateViewMatrix();
122
125    void Left(float dt);
126
131    void Right(float dt);
132
137    void Forward(float dt);
138
143    void Backward(float dt);
144
149    void Up(float dt);
150
155    void Down(float dt);
156
161    void RotateCameraLeftRight(float xDiff);
162
167    void RotateCameraUpDown(float yDiff);
168
177    void KeyboardInput(float dt);
178
187    void KeyboardInputWASD(float dt);
188
197    void KeyboardInputArrow(float dt);
198
201    void MouseInput();
202
203 private:
204     //camera position in world coordinates
205     vec3 mCameraPosition;
206
207     //z-axis of the camera coordinate system
208     vec3 mN;
209
210     //y-axis of the camera coordinate system
211     vec3 mV;
212
213     //x-axis of the camera coordinate system
214     vec3 mU;
215
216     //stores the world to camera transform
217     mat4 mViewMatrix;
218
219     float mCameraVelocity;
220     float mAngularVelocity;
221
222     vec2 mLastMousePosition;
223
224     bool mUpdateViewMatrix;
225 };
226 }

```

7.7 FAColor.h File Reference

File has class Color under namespace [FAColor](#).

```
#include "FAMathEngine.h"
```

Classes

- class [FAColor::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.

Namespaces

- namespace [FAColor](#)
Has the [Color](#) class.

Functions

- Color [FAColor::operator+](#) (const Color &c1, const Color &c2)
Returns the result of $c1 + c2$.
- Color [FAColor::operator-](#) (const Color &c1, const Color &c2)
Returns the result of $c1 - c2$.
- Color [FAColor::operator*](#) (const Color &c, float k)
Returns the result of $c * k$.
- Color [FAColor::operator*](#) (float k, const Color &c)
Returns the result of $k * c$.
- Color [FAColor::operator*](#) (const Color &c1, const Color &c2)
Returns the result of $c1 * c2$.

7.7.1 Detailed Description

File has class Color under namespace [FAColor](#).

7.8 FAColor.h

[Go to the documentation of this file.](#)

```

1 #pragma once
2
3 #include "FAMathEngine.h"
4
12 namespace FAColor
13 {
14     class Color
15     {
16     public:
17
25         Color(float r = 0.0f, float g = 0.0f, float b = 0.0f, float a = 1.0f);
26
29         Color(const FAMath::Vector4D& color);
30
33         const FAMath::Vector4D& GetColor() const;
34
37         float GetRed() const;
38
41         float GetGreen() const;
42
45         float GetBlue() const;
46
49         float GetAlpha() const;
50
53         void SetColor(const FAMath::Vector4D& color);
54
57         void SetRed(float r);
58
61         void SetGreen(float g);
62
65         void SetBlue(float b);
66
69         void SetAlpha(float a);
70
75         Color& operator+=(const Color& c);
76
81         Color& operator-=(const Color& c);
82
88         Color& operator*=(float k);

```

```

89
95         Color& operator*=(const Color& c);
96
97     private:
98         FMath::Vector4D mColor;
99     };
100
105     Color operator+(const Color& c1, const Color& c2);
106
111     Color operator-(const Color& c1, const Color& c2);
112
118     Color operator*(const Color& c, float k);
119
126     Color operator*(float k, const Color& c);
127
132     Color operator*(const Color& c1, const Color& c2);
133 }

```

7.9 FADeviceResources.h File Reference

File has class DeviceResources under namespace [FARender](#).

```

#include <wrl.h>
#include <d3d12.h>
#include <dxgil_4.h>
#include "FASwapChain.h"
#include "FAMultiSampling.h"
#include "FATextResources.h"

```

Classes

- class [FARender::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.

Namespaces

- namespace [FARender](#)

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

7.9.1 Detailed Description

File has class DeviceResources under namespace [FARender](#).

7.10 FADeviceResources.h

[Go to the documentation of this file.](#)

```

1  #pragma once
2
3  #include <wrl.h>
4  #include <d3d12.h>
5  #include <dxgi1_4.h>
6  #include "FASwapChain.h"
7  #include "FAMultiSampling.h"
8  #include "FATextResources.h"
9
10 namespace FASwapChain
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         const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& GetCommandList() const;
29
30         DXGI_FORMAT GetBackBufferFormat() const;
31         DXGI_FORMAT GetDepthStencilFormat() const;
32
33         unsigned int GetCBVSRVUAVSize() const;
34         unsigned int GetCurrentFrame() const;
35
36         const TextResources& GetTextResources() const;
37
38         void UpdateCurrentFrameFenceValue();
39
40         void FlushCommandQueue();
41
42         void WaitForGPU() const;
43
44         void Signal();
45
46         void Resize(int width, int height, const HWND& handle, bool isMSAAEnabled, bool isTextEnabled);
47         void RTBufferTransition(bool isMSAAEnabled, bool isTextEnabled);
48         void BeforeTextDraw();
49         void AfterTextDraw();
50         void Execute() const;
51         void Present();
52
53         /*@brief Calls the necessary functions to let the user draw their objects.
54          * @param[in] isMSAAEnabled Pass in true if MSAA enabled, false otherwise.
55          */
56         void Draw(bool isMSAAEnabled);
57         void NextFrame();
58
59     private:
60         DeviceResources(unsigned int width, unsigned int height, HWND windowHandle,
61             bool isMSAAEnabled, bool isTextEnabled);
62
63         unsigned int mCurrentFrameIndex;
64         Microsoft::WRL::ComPtr<ID3D12Device> mDirect3DDevice;
65         Microsoft::WRL::ComPtr<IDXGIFactory4> mDXGIFactory;
66         Microsoft::WRL::ComPtr<ID3D12Fence> mFence;
67         UINT64 mFenceValue;
68         UINT64 mCurrentFrameFenceValue[NUM_OF_FRAMES];

```

```

171
172     Microsoft::WRL::ComPtr<ID3D12CommandQueue> mCommandQueue;
173     Microsoft::WRL::ComPtr<ID3D12CommandAllocator> mCommandAllocators[NUM\_OF\_FRAMES];
174     Microsoft::WRL::ComPtr<ID3D12CommandAllocator> mDirectCommandAllocator;
175     Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList> mCommandList;
176
177     UINT mRTVSize;
178     UINT mDSVSize;
179     UINT mCBVSize;
180
181     Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mRTVHeap;
182     Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mDSVHeap;
183
184     SwapChain mSwapChain;
185
186     MultiSampling mMultiSampling;
187
188     D3D12_VIEWPORT mViewport{};
189     D3D12_RECT mScissor{};
190
191     TextResources mTextResources;
192
193     //Call all of these functions to initialize Direct3D
194     void mEnableDebugLayer();
195     void mCreateDirect3DDevice();
196     void mCreateDXGIFactory();
197     void mCreateFence();
198     void mQueryDescriptorSizes();
199     void mCreateRTVHeap();
200     void mCreateDSVHeap();
201     void mCreateCommandObjects();
202 };
203 }

```

7.11 FADirectXException.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
10 inline std::wstring AnsiToWString(const std::string& str)
11 {
12     WCHAR buffer[1024];
13     MultiByteToWideChar(CP_ACP, 0, str.c_str(), -1, buffer, 1024);
14     return std::wstring(buffer);
15 }
16
17 class DirectXException
18 {
19 public:
20     DirectXException(HRESULT hr, const std::wstring& functionName, const std::wstring& fileName, int
        lineNumber);
21
22     std::wstring ErrorMessage() const;
23
24 private:
25     HRESULT errorCode;
26     std::wstring functionName;
27     std::wstring fileName;
28     int lineNumber;
29     Microsoft::WRL::ComPtr<IDXGIInfoQueue> mInfoQueue;
30 };
31
32 #ifndef ThrowIfFailed
33 #define ThrowIfFailed(x)
34 {
35     HRESULT hr = (x);
36     std::wstring filename(AnsiToWString(__FILE__));
37     if (FAILED(hr)) { throw DirectXException(hr, L#x, filename, __LINE__); }
38 }
39 #endif

```

7.12 FAMultiSampling.h

```

1  #pragma once

```

```

2
3 #include <wrl.h>
4 #include "d3dx12.h"
5 #include "FABuffer.h"
6
7 namespace FAREnder
8 {
9     class MultiSampling
10     {
11     public:
12
13         MultiSampling(const MultiSampling&) = delete;
14         MultiSampling& operator=(const MultiSampling&) = delete;
15
16         MultiSampling();
17
18         MultiSampling(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
19             DXGI_FORMAT rtFormat, unsigned int sampleCount);
20
21         void CheckMultiSamplingSupport(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
22             DXGI_FORMAT rtFormat, unsigned int sampleCount);
23
24         const Microsoft::WRL::ComPtr<ID3D12Resource>& GetRenderTargetBuffer();
25
26         DXGI_FORMAT GetRenderTargetFormat();
27
28         DXGI_FORMAT GetDepthStencilFormat();
29
30         void ReleaseBuffers();
31
32         void CreateRenderTargetBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
33             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int
34             indexWhereToStoreView, unsigned int rtvSize,
35             unsigned int width, unsigned int height);
36
37         void CreateDepthStencilBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
38             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int
39             indexWhereToStoreView, unsigned int dsvSize,
40             unsigned int width, unsigned int height);
41
42         void Transition(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList,
43             D3D12_RESOURCE_STATES before, D3D12_RESOURCE_STATES after);
44
45         void ClearRenderTargetBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
46             commandList,
47             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int indexWhereToStoreView,
48             unsigned int rtvSize,
49             const float* clearValue);
50
51         void ClearDepthStencilBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
52             commandList,
53             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int indexWhereToStoreView,
54             unsigned int dsvSize,
55             float clearValue);
56
57     private:
58         RenderTargetBuffer mMSAARenderTargetBuffer;
59         DepthStencilBuffer mMSAADepthStencilBuffer;
60         unsigned int mSampleCount;
61     };
62 }

```

7.13 FAProjection.h File Reference

File that has namespace [FAProjection](#). Within the namespace is the interface [IProjection](#) and class [PerspectiveProjection](#).

```
#include "FAMathEngine.h"
```

Classes

- class [FAProjection::IProjection](#)
An interface for projections.
- class [FAProjection::PerspectiveProjection](#)
A class for doing perspective projection.

Namespaces

- namespace [FAProjection](#)

Within the namespace is the interface [IProjection](#) and class [PerspectiveProjection](#).

7.13.1 Detailed Description

File that has namespace [FAProjection](#). Within the namespace is the interface [IProjection](#) and class [PerspectiveProjection](#).

7.14 FAProjection.h

[Go to the documentation of this file.](#)

```

1  #pragma once
2
7  #include "FAMathEngine.h"
8
12 namespace FAProjection
13 {
17     class IProjection
18     {
19     public:
20
23         IProjection();
24
27         const FAMath::Matrix4x4& GetProjectionMatrix() const;
28
29         virtual void UpdateProjectionMatrix() = 0;
30
31     protected:
32         FAMath::Matrix4x4 mProjectionMatrix;
33         bool mUpdateProjectionMatrix;
34     };
35
39     class PerspectiveProjection : public IProjection
40     {
41     public:
42
45         PerspectiveProjection();
46
54         PerspectiveProjection(float znear, float zfar, float vFov, float aspectRatio);
55
63         void SetProperties(float znear, float zfar, float vFov, float aspectRatio);
64
67         float GetNear() const;
68
71         float GetFar() const;
72
75         float GetVerticalFov() const;
76
79         float GetAspectRatio() const;
80
83         void SetNear(float near);
84
87         void SetFar(float far);
88
91         void SetVerticalFov(float fov);
92
95         void SetAspectRatio(float ar);
96
99         void UpdateProjectionMatrix() override final;
100
101     private:
102         //frustrum properties
103         float mNear;
104         float mFar;
105         float mVerticalFov;
106         float mAspectRatio;
107     };
108 }
```

7.15 FARenderScene.h File Reference

File has class `RenderScene` under namespace `FARender`.

```
#include <d3dcompiler.h>
#include <unordered_map>
#include "FADeviceResources.h"
#include "FABuffer.h"
#include "FAColor.h"
```

Classes

- class `FARender::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.

Namespaces

- namespace `FARender`

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

7.15.1 Detailed Description

File has class `RenderScene` under namespace `FARender`.

7.16 FARenderScene.h

[Go to the documentation of this file.](#)

```
1 #pragma once
2
3 #include <d3dcompiler.h>
4 #include <unordered_map>
5 #include "FADeviceResources.h"
6 #include "FABuffer.h"
7 #include "FAColor.h"
8
9 namespace FARender
10 {
11     class RenderScene
12     {
13     public:
14
15         //No copying
16         RenderScene(const RenderScene&) = delete;
17         RenderScene operator=(const RenderScene&) = delete;
18
19         /**@brief Creates a RenderScene object. Does not create the necessary resources to render a
20          scene.
21         * Call the function CreateDeviceResources() to initialize all necessary resources.
22         */
23         RenderScene();
24
25         /**@brief Initializes all necessary resources.
26         *
27         * @param[in] width The width of a window.
28         * @param[in] height The height of a window.
29         * @param[in] windowHandle A handle to a window.
30         * @param[in, optional] isMSAAEnabled Pass in true if you want to have MSAA enabled for the initial frame,
31           false otherwise.
```

```

38 * @param[in, optional] isTextEnabled Pass in true if you want to have text enabled for the initial frame,
   false otherwise.
39 */
40     RenderScene(unsigned int width, unsigned int height, HWND windowHandle, bool isMSAAEnabled =
   false, bool isTextEnabled = false);
41
42     void CreateDeviceResources(unsigned int width, unsigned int height, HWND windowHandle, bool
   isMSAAEnabled = false, bool isTextEnabled = false);
43
44     void LoadShader(unsigned int shaderKey, const std::wstring& filename);
45
46     void CompileShader(unsigned int shaderKey, const std::wstring& filename,
47         const std::string& entryPointName, const std::string& target);
48
49     void RemoveShader(unsigned int shaderKey);
50
51     void CreateStaticBuffer(unsigned int staticBufferKey, const void* data, unsigned numBytes,
52         unsigned int stride);
53
54     void CreateStaticBuffer(unsigned int staticBufferKey, const void* data, unsigned numBytes,
55         DXGI_FORMAT format);
56
57     void CreateStaticBuffer(unsigned int staticBufferKey, const wchar_t* filename, unsigned int
58         texType, unsigned int index);
59
60     void CreateDynamicBuffer(unsigned int dynamicBufferKey, unsigned numBytes, const void* data,
61         unsigned int stride);
62
63     void CreateDynamicBuffer(unsigned int dynamicBufferKey, unsigned numBytes, const void* data,
64         DXGI_FORMAT format);
65
66     void CreateInputElementDescription(unsigned int key, const char* semanticName, unsigned int
67         semanticIndex,
68         DXGI_FORMAT format, unsigned int inputSlot, unsigned int byteOffset,
69         D3D12_INPUT_CLASSIFICATION inputSlotClass = D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA,
70         unsigned int instanceStepRate = 0);
71
72     void CreateRootDescriptor(unsigned int rootParameterKey, unsigned int shaderRegister);
73
74     void CreateDescriptorRange(unsigned int descriptorRangeKey,
75         D3D12_DESCRIPTOR_RANGE_TYPE type, unsigned int numDescriptors, unsigned int shaderRegister,
76         unsigned int registerSpace);
77
78     void CreateDescriptorTable(unsigned int rootParameterKey, unsigned int descriptorRangeKey);
79
80     void CreateRootConstants(unsigned int rootParameterKey, unsigned int shaderRegister, unsigned
81         int numValues);
82
83     void CreateRootSignature(unsigned int rootSigKey, unsigned int rootParametersKey);
84
85     void CreateRootSignature(unsigned int rootSigKey, unsigned int rootParametersKey,
86         unsigned int staticsSamplerKey);
87
88     void CreatePSO(unsigned int psKey, D3D12_FILL_MODE fillMode, BOOL enableMultisample,
89         unsigned int vsKey, unsigned int psKey, unsigned int inputElementDescriptionsKey,
90         unsigned int rootSigKey,
91         const D3D12_PRIMITIVE_TOPOLOGY_TYPE& primitiveType, UINT sampleCount = 1);
92
93     void SetPSOAndRootSignature(unsigned int psKey, unsigned int rootSigKey);
94
95     void SetStaticBuffer(unsigned int bufferType, unsigned int staticBufferKey);
96
97     void SetDynamicBuffer(unsigned int bufferType, unsigned int dynamicBufferKey, unsigned int
98         indexConstantData = 0,
99         unsigned int rootParameterIndex = 0);
100
101     void SetTextureViewHeap();
102
103     void SetTexture(unsigned int rootParameterIndex, unsigned int textureViewIndex);
104
105     void SetConstants(unsigned int rootParameterIndex, unsigned int numValues, void* data, unsigned
106         int index);
107
108     void BeforeRenderObjects(bool isMSAAEnabled = false);
109
110     void RenderObject(unsigned int indexCount, unsigned int locationFirstIndex, int
111         indexOffFirstVertex,
112         D3D_PRIMITIVE_TOPOLOGY primitive);
113
114     void AfterRenderObjects(bool isMSAAEnabled = false, bool isTextEnabled = false);
115
116     void BeforeRenderText();
117
118     void RenderText(const FAMath::Vector4D& textLocation, const FColor::Color& textColor, float
119         textSize,
120         const std::wstring& textString, DWRITE_PARAGRAPH_ALIGNMENT alignment =
121         DWRITE_PARAGRAPH_ALIGNMENT_CENTER);

```



```

28
38     SwapChain(const Microsoft::WRL::ComPtr<IDXGIFactory4>& dxgiFactory,
39         const Microsoft::WRL::ComPtr<ID3D12CommandQueue>& commandQueue, HWND windowHandle,
40         DXGI_FORMAT rtFormat = DXGI_FORMAT_R8G8B8A8_UNORM, DXGI_FORMAT dsFormat =
DXGI_FORMAT_D24_UNORM_S8_UINT,
41         unsigned int numRenderTargetBuffers = 2);
42
43     void CreateSwapChain(const Microsoft::WRL::ComPtr<IDXGIFactory4>& dxgiFactory,
44         const Microsoft::WRL::ComPtr<ID3D12CommandQueue>& commandQueue, HWND windowHandle,
45         DXGI_FORMAT rtFormat = DXGI_FORMAT_R8G8B8A8_UNORM, DXGI_FORMAT dsFormat =
DXGI_FORMAT_D24_UNORM_S8_UINT,
46         unsigned int numRenderTargetBuffers = 2);
47
48     const Microsoft::WRL::ComPtr<ID3D12Resource>& GetCurrentBackBuffer() const;
49
50     unsigned int GetNumRenderTargetBuffers() const;
51
52     unsigned int GetCurrentBackBufferIndex() const;
53
54     DXGI_FORMAT GetBackBufferFormat() const;
55
56     DXGI_FORMAT GetDepthStencilFormat() const;
57
58     const std::vector<std::unique_ptr<RenderTargetBuffer>>& GetRenderTargetBuffers();
59
60     void ReleaseBuffers();
61
62     void CreateRenderTargetBuffersAndViews(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
63         const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int index,
64         unsigned int rtvSize, unsigned width, unsigned height);
65
66     void CreateDepthStencilBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
67         const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int index, unsigned
int dsvSize,
68         unsigned int width, unsigned int height);
69
70     void ClearCurrentBackBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
commandList,
71         const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int index,
72         unsigned int rtvSize,
73         const float* backBufferClearValue);
74
75     void ClearDepthStencilBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
commandList,
76         const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int index,
77         unsigned int dsvSize,
78         float clearValue);
79
80     void Transition(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList,
81         D3D12_RESOURCE_STATES before, D3D12_RESOURCE_STATES after);
82
83     void Present();
84
85 private:
86     unsigned int mNumRenderTargetBuffers;
87     unsigned int mCurrentBackBufferIndex;
88
89     Microsoft::WRL::ComPtr<IDXGISwapChain1> mSwapChain;
90     std::vector<std::unique_ptr<RenderTargetBuffer>> mRenderTargetBuffers;
91
92     DepthStencilBuffer mDepthStencilBuffer;
93 };
94 }

```

7.18 FAtext.h File Reference

File has class `Text` under namespace `FARender`.

```

#include <string>
#include "FAColor.h"

```

Classes

- class `FARender::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.

Namespaces

- namespace [FARender](#)

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

7.18.1 Detailed Description

File has class Text under namespace [FARender](#).

7.19 FAText.h

[Go to the documentation of this file.](#)

```

1 #pragma once
2
3 #include <string>
4 #include "FAColor.h"
5
6 namespace FArender
7 {
8     class Text
9     {
10     public:
11
12         Text() = default;
13
14         Text(const FAMath::Vector4D& textLocation, const std::wstring& textString, float textSize, const
FAColor::Color& textColor);
15
16         const FAMath::Vector4D& GetTextLocation() const;
17
18         const std::wstring& GetTextString() const;
19
20         float GetTextSize() const;
21
22         const FAColor::Color& GetTextColor() const;
23
24         void SetTextSize(float textSize);
25
26         void SetTextColor(const FAColor::Color& textColor);
27
28         void SetTextString(const std::wstring& textString);
29
30         void SetTextLocation(const FAMath::Vector4D& textLocation);
31
32     private:
33
34         FAMath::Vector4D mTextLocation;
35         std::wstring mText;
36         float mTextSize{ 0.0f };
37         FAColor::Color mTextColor;
38     };
39 }
```

7.20 FATextResources.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 "FABuffer.h"
11
12 namespace FArender
13 {
14     class TextResources
15     {
16     {
```

```

19     public:
20         TextResources() = default;
21
22         TextResources(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
23             const Microsoft::WRL::ComPtr<ID3D12CommandQueue>& commandQueue, unsigned int
24             numSwapChainBuffers);
25
26         const Microsoft::WRL::ComPtr<ID2D1DeviceContext>& GetDirect2DDeviceContext() const;
27
28         const Microsoft::WRL::ComPtr<IDWriteFactory>& GetDirectWriteFactory() const;
29
30         void ResetBuffers();
31
32         void ResizeBuffers(const std::vector<std::unique_ptr<RenderTargetBuffer>& renderTargetBuffers,
33             HWND windowHandle);
34
35         void BeforeRenderText(unsigned int currentBackBuffer);
36
37         void AfterRenderText(unsigned int currentBackBuffer);
38
39     private:
40         Microsoft::WRL::ComPtr<ID3D11Device> mDevice11;
41         Microsoft::WRL::ComPtr<ID3D11DeviceContext> mDevice11Context;
42         Microsoft::WRL::ComPtr<ID3D11On12Device> mDevice11on12;
43
44         Microsoft::WRL::ComPtr<ID2D1Device2> mDirect2DDevice;
45         Microsoft::WRL::ComPtr<ID2D1Factory3> mDirect2DFactory;
46         Microsoft::WRL::ComPtr<ID2D1DeviceContext> mDirect2DDeviceContext;
47
48         Microsoft::WRL::ComPtr<IDWriteFactory> mDirectWriteFactory;
49
50         std::vector<Microsoft::WRL::ComPtr<ID3D11Resource>> mWrappedBuffers;
51         std::vector<Microsoft::WRL::ComPtr<ID2D1Bitmap1> mDirect2DBuffers;
52         std::vector<Microsoft::WRL::ComPtr<IDXGISurface> mSurfaces;
53
54     };
55 }

```

7.21 FATime.h File Reference

File that has namespace [FATime](#). Withn the namespace is the class Time.

```
#include <Windows.h>
```

Classes

- class [FATime::Time](#)

This class is used to get the time between each frame. You can stop start, reset and get the total time.

Namespaces

- namespace [FATime](#)

Has [Time](#) class.

7.21.1 Detailed Description

File that has namespace [FATime](#). Withn the namespace is the class Time.

7.22 FATime.h

[Go to the documentation of this file.](#)

```

1 #pragma once
2
3 #include <Windows.h>
4
5 namespace FATime
6 {
7     class Time
8     {
9     public:
10         Time();
11
12         void Tick();
13
14         float DeltaTime() const;
15
16         void Reset();
17
18         void Stop();
19
20         void Start();
21
22         float TotalTime() const;
23
24     private:
25         __int64 mCurrTime; //holds current time stamp ti
26         __int64 mPrevTime; //holds previous time stamp ti-1
27         __int64 mStopTime; //holds the time we stopped the game/animation
28         __int64 mPausedTime; //holds how long the game/animation was paused for
29         __int64 mBaseTime; //holds the time we started / resetted
30
31         double mSecondsPerCount;
32         double mDeltaTime; //time elapsed btw frames change in t = ti - ti-1
33
34         bool mStopped; //flag to indicate if the game/animation is paused or not
35     };
36 }

```

7.23 FAWindow.h File Reference

File that has namespace [FAWindow](#). Withn the namespace is the class Window.

```

#include <Windows.h>
#include <string>
#include <stdexcept>
#include "FAColor.h"

```

Classes

- class [FAWindow::Window](#)

The window class is used to make a [Window](#) using Windows API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

Namespaces

- namespace [FAWindow](#)

Has [Window](#) class.

7.23.1 Detailed Description

File that has namespace [FAWindow](#). Withn the namespace is the class [Window](#).

7.24 FAWindow.h

[Go to the documentation of this file.](#)

```

1 #pragma once
2
3 #include <Windows.h>
4 #include <string>
5 #include <stdexcept>
6 #include "FAColor.h"
7
8 namespace FAWindow
9 {
10     class Window
11     {
12     public:
13
14         Window(const Window&) = delete;
15         Window& operator=(const Window&) = delete;
16
17         Window();
18
19         Window(const HINSTANCE& hInstance, WNDPROC windowProcedure, const FAColor::Color&
backgroundColor,
20             const std::wstring& windowClassName, const std::wstring& windowName, unsigned int styles,
21             unsigned int x, unsigned int y, unsigned int width, unsigned int height, void* additionalData
22             = nullptr);
23
24         Window(const HINSTANCE& hInstance, HWND parent, unsigned int identifier,
25             WNDPROC windowProcedure, const FAColor::Color& backgroundColor,
26             const std::wstring& windowClassName, const std::wstring& windowName, unsigned int styles,
27             unsigned int x, unsigned int y, unsigned int width, unsigned int height, void*
28             additionalData = nullptr);
29
30         Window(const HINSTANCE& hInstance, HWND parent, unsigned int identifier,
31             const std::wstring& windowClassName,
32             const std::wstring& windowName, unsigned int styles,
33             unsigned int x, unsigned int y, unsigned int width, unsigned int height, void*
34             additionalData = nullptr);
35
36         void CreateParentWindow(const HINSTANCE& hInstance, WNDPROC windowProcedure, const
FAColor::Color& backgroundColor,
37             const std::wstring& windowClassName, const std::wstring& windowName, unsigned int styles,
38             unsigned int x, unsigned int y, unsigned int width, unsigned int height, void*
39             additionalData = nullptr);
40
41         void CreateChildWindow(const HINSTANCE& hInstance, HWND parent, unsigned int identifier,
42             WNDPROC windowProcedure, const FAColor::Color& backgroundColor,
43             const std::wstring& windowClassName, const std::wstring& windowName, unsigned int styles,
44             unsigned int x, unsigned int y, unsigned int width, unsigned int height, void*
45             additionalData = nullptr);
46
47         void CreateControlWindow(const HINSTANCE& hInstance, HWND parent, unsigned int identifier,
48             const std::wstring& windowClassName,
49             const std::wstring& windowName, unsigned int styles,
50             unsigned int x, unsigned int y, unsigned int width, unsigned int height, void*
51             additionalData = nullptr);
52
53         HWND GetWindowHandle() const;
54
55         unsigned int GetWidth() const ;
56
57         unsigned int GetHeight() const;
58
59         unsigned int GetX() const;
60
61         unsigned int GetY() const;
62
63         void SetWidth(unsigned int width);
64
65         void SetHeight(unsigned int height);
66
67         void SetX(unsigned int x);
68
69         void SetY(unsigned int y);
70
71     };
72
73 }
```

```
274     private:
275         HWND mWindowHandle;
276
277         WNDCLASSEX mWindowClass;
278
279         unsigned int mX;
280         unsigned int mY;
281         unsigned int mWidth;
282         unsigned int mHeight;
283     };
284 }
```

Index

- ~DeviceResources
 - FARender::DeviceResources, [32](#)
- ~DynamicBuffer
 - FARender::DynamicBuffer, [39](#)
- AfterRender
 - FARender::RenderScene, [55](#)
- AfterRenderObjects
 - FARender::RenderScene, [55](#)
- AfterRenderText
 - FARender::RenderScene, [55](#)
 - FARender::TextResources, [85](#)
- AfterTextDraw
 - FARender::DeviceResources, [32](#)
- Backward
 - FACamera::Camera, [17](#)
- BeforeRenderObjects
 - FARender::RenderScene, [55](#)
- BeforeRenderText
 - FARender::RenderScene, [56](#)
 - FARender::TextResources, [85](#)
- BeforeTextDraw
 - FARender::DeviceResources, [32](#)
- Camera
 - FACamera::Camera, [16](#), [17](#)
- CheckMultiSamplingSupport
 - FARender::MultiSampling, [45](#)
- ClearCurrentBackBuffer
 - FARender::SwapChain, [78](#)
- ClearDepthStencilBuffer
 - FARender::DepthStencilBuffer, [29](#)
 - FARender::MultiSampling, [46](#)
 - FARender::SwapChain, [78](#)
- ClearRenderTargetBuffer
 - FARender::MultiSampling, [46](#)
 - FARender::RenderTargetBuffer, [69](#)
- Color
 - FAColor::Color, [25](#)
- CompileShader
 - FARender::RenderScene, [56](#)
- CopyData
 - FARender::DynamicBuffer, [40](#)
- CopyDataIntoDynamicBuffer
 - FARender::RenderScene, [56](#)
- CreateChildWindow
 - FAWindow::Window, [92](#)
- CreateConstantBufferView
 - FARender::DynamicBuffer, [40](#)
- CreateControlWindow
 - FAWindow::Window, [93](#)
- CreateDepthStencilBufferAndView
 - FARender::DepthStencilBuffer, [30](#)
 - FARender::MultiSampling, [46](#)
 - FARender::SwapChain, [78](#)
- CreateDescriptorRange
 - FARender::RenderScene, [56](#)
- CreateDescriptorTable
 - FARender::RenderScene, [57](#)
- CreateDynamicBuffer
 - FARender::DynamicBuffer, [40](#), [41](#)
 - FARender::RenderScene, [57](#), [58](#)
- CreateInputElementDescription
 - FARender::RenderScene, [58](#)
- CreateParentWindow
 - FAWindow::Window, [94](#)
- CreatePSO
 - FARender::RenderScene, [59](#)
- CreateRenderTargetBufferAndView
 - FARender::MultiSampling, [47](#)
 - FARender::RenderTargetBuffer, [69](#)
- CreateRenderTargetBuffersAndViews
 - FARender::SwapChain, [79](#)
- CreateRootConstants
 - FARender::RenderScene, [59](#)
- CreateRootDescriptor
 - FARender::RenderScene, [60](#)
- CreateRootSignature
 - FARender::RenderScene, [60](#)
- CreateStaticBuffer
 - FARender::RenderScene, [61](#), [62](#)
 - FARender::StaticBuffer, [73](#), [74](#)
- CreateStaticSampler
 - FARender::RenderScene, [62](#)
- CreateSwapChain
 - FARender::SwapChain, [79](#)
- CreateTexture2DMSView
 - FARender::StaticBuffer, [74](#)
- CreateTexture2DView
 - FARender::StaticBuffer, [75](#)
- CreateTextureViewHeap
 - FARender::RenderScene, [63](#)
- DeltaTime
 - FATime::Time, [87](#)
- DepthStencilBuffer
 - FARender::DepthStencilBuffer, [29](#)
- DirectXException, [36](#)
 - DirectXException, [37](#)

- ErrorMsg, [37](#)
- Down
 - FACamera::Camera, [17](#)
- DynamicBuffer
 - FARender::DynamicBuffer, [39](#)
- ErrorMsg
 - DirectXException, [37](#)
- Execute
 - FARender::DeviceResources, [32](#)
- ExecuteAndFlush
 - FARender::RenderScene, [63](#)
- FABuffer.h, [98](#)
- FACamera, [9](#)
- FACamera.h, [101](#)
 - vec2, [102](#)
- FACamera::Camera, [15](#)
 - Backward, [17](#)
 - Camera, [16, 17](#)
 - Down, [17](#)
 - Foward, [18](#)
 - GetAngularVelocity, [18](#)
 - GetCameraPosition, [18](#)
 - GetCameraVelocity, [18](#)
 - GetViewMatrix, [18](#)
 - GetX, [19](#)
 - GetY, [19](#)
 - GetZ, [19](#)
 - KeyboardInput, [19](#)
 - KeyboardInputArrow, [19](#)
 - KeyboardInputWASD, [20](#)
 - Left, [20](#)
 - LookAt, [20](#)
 - MouseInput, [21](#)
 - Right, [21](#)
 - RotateCameraLeftRight, [21](#)
 - RotateCameraUpDown, [21](#)
 - SetAngularVelocity, [22](#)
 - SetCameraPosition, [22](#)
 - SetCameraVelocity, [22](#)
 - SetProperties, [22](#)
 - SetX, [23](#)
 - SetY, [23](#)
 - SetZ, [23](#)
 - Up, [23](#)
 - UpdateViewMatrix, [23](#)
- FAColor, [9](#)
 - operator*, [10](#)
 - operator+, [11](#)
 - operator-, [11](#)
- FAColor.h, [103](#)
- FAColor::Color, [24](#)
 - Color, [25](#)
 - GetAlpha, [25](#)
 - GetBlue, [25](#)
 - GetColor, [25](#)
 - GetGreen, [25](#)
 - GetRed, [26](#)
- operator*=, [26](#)
- operator+=, [26](#)
- operator-=, [26](#)
- SetAlpha, [27](#)
- SetBlue, [27](#)
- SetColor, [27](#)
- SetGreen, [27](#)
- SetRed, [27](#)
- FADeviceResources.h, [105](#)
- FAProjection, [11](#)
- FAProjection.h, [108](#)
- FAProjection::IProjection, [42](#)
 - GetProjectionMatrix, [43](#)
 - IProjection, [43](#)
 - UpdateProjectionMatrix, [43](#)
- FAProjection::PerspectiveProjection, [48](#)
 - GetAspectRatio, [50](#)
 - GetFar, [50](#)
 - GetNear, [50](#)
 - GetVerticalFov, [51](#)
 - PerspectiveProjection, [49, 50](#)
 - SetAspectRatio, [51](#)
 - SetFar, [51](#)
 - SetNear, [51](#)
 - SetProperties, [51](#)
 - SetVerticalFov, [52](#)
 - UpdateProjectionMatrix, [52](#)
- FARender, [12](#)
- FARender::DepthStencilBuffer, [28](#)
 - ClearDepthStencilBuffer, [29](#)
 - CreateDepthStencilBufferAndView, [30](#)
 - DepthStencilBuffer, [29](#)
 - GetDepthStencilFormat, [30](#)
 - ReleaseBuffer, [30](#)
- FARender::DeviceResources, [31](#)
 - ~DeviceResources, [32](#)
 - AfterTextDraw, [32](#)
 - BeforeTextDraw, [32](#)
 - Execute, [32](#)
 - FlushCommandQueue, [33](#)
 - GetBackBufferFormat, [33](#)
 - GetCBVSRVUAVSize, [33](#)
 - GetCommandList, [33](#)
 - GetCurrentFrame, [33](#)
 - GetDepthStencilFormat, [33](#)
 - GetDevice, [34](#)
 - GetInstance, [34](#)
 - GetTextResources, [34](#)
 - NextFrame, [34](#)
 - NUM_OF_FRAMES, [36](#)
 - Present, [35](#)
 - Resize, [35](#)
 - RTBufferTransition, [35](#)
 - Signal, [36](#)
 - UpdateCurrentFrameFenceValue, [36](#)
 - WaitForGPU, [36](#)
- FARender::DynamicBuffer, [38](#)
 - ~DynamicBuffer, [39](#)

- CopyData, 40
- CreateConstantBufferView, 40
- CreateDynamicBuffer, 40, 41
- DynamicBuffer, 39
- GetGPUAddress, 41
- GetIndexBufferView, 41
- GetVertexBufferView, 41
- ReleaseBuffer, 42
- FARender::MultiSampling, 43
 - CheckMultiSamplingSupport, 45
 - ClearDepthStencilBuffer, 46
 - ClearRenderTargetBuffer, 46
 - CreateDepthStencilBufferAndView, 46
 - CreateRenderTargetBufferAndView, 47
 - GetDepthStencilFormat, 47
 - GetRenderTargetBuffer, 47
 - GetRenderTargetFormat, 48
 - MultiSampling, 44, 45
 - ReleaseBuffers, 48
 - Transition, 48
- FARender::RenderScene, 52
 - AfterRender, 55
 - AfterRenderObjects, 55
 - AfterRenderText, 55
 - BeforeRenderObjects, 55
 - BeforeRenderText, 56
 - CompileShader, 56
 - CopyDataIntoDynamicBuffer, 56
 - CreateDescriptorRange, 56
 - CreateDescriptorTable, 57
 - CreateDynamicBuffer, 57, 58
 - CreateInputElementDescription, 58
 - CreatePSO, 59
 - CreateRootConstants, 59
 - CreateRootDescriptor, 60
 - CreateRootSignature, 60
 - CreateStaticBuffer, 61, 62
 - CreateStaticSampler, 62
 - CreateTextureViewHeap, 63
 - ExecuteAndFlush, 63
 - LoadShader, 63
 - RemoveShader, 63
 - RenderObject, 63
 - RenderText, 64
 - Resize, 65
 - SetConstants, 65
 - SetDynamicBuffer, 65
 - SetPSOAndRootSignature, 66
 - SetStaticBuffer, 66
 - SetTexture, 67
- FARender::RenderTargetBuffer, 67
 - ClearRenderTargetBuffer, 69
 - CreateRenderTargetBufferAndView, 69
 - GetRenderTargetBuffer, 70
 - GetRenderTargetFormat, 70
 - ReleaseBuffer, 70
 - RenderTargetBuffer, 68
- FARender::StaticBuffer, 71
 - CreateStaticBuffer, 73, 74
 - CreateTexture2DMSView, 74
 - CreateTexture2DView, 75
 - GetIndexBufferView, 75
 - GetVertexBufferView, 75
 - ReleaseBuffer, 75
 - StaticBuffer, 72, 73
- FARender::SwapChain, 76
 - ClearCurrentBackBuffer, 78
 - ClearDepthStencilBuffer, 78
 - CreateDepthStencilBufferAndView, 78
 - CreateRenderTargetBuffersAndViews, 79
 - CreateSwapChain, 79
 - GetBackBufferFormat, 80
 - GetCurrentBackBuffer, 80
 - GetCurrentBackBufferIndex, 80
 - GetDepthStencilFormat, 80
 - GetNumRenderTargetBuffers, 80
 - Present, 81
 - ReleaseBuffers, 81
 - SwapChain, 77
 - Transition, 81
- FARender::Text, 81
 - GetTextColor, 83
 - GetTextLocation, 83
 - GetTextSize, 83
 - GetTextString, 83
 - SetTextColor, 83
 - SetTextLocation, 83
 - SetTextSize, 84
 - SetTextString, 84
 - Text, 82
- FARender::TextResources, 84
 - AfterRenderText, 85
 - BeforeRenderText, 85
 - GetDirect2DDeviceContext, 86
 - GetDirectWriteFactory, 86
 - ResetBuffers, 86
 - ResizeBuffers, 86
 - TextResources, 85
- FARenderScene.h, 110
- FAText.h, 113
- FATime, 12
- FATime.h, 115
- FATime::Time, 87
 - DeltaTime, 87
 - Reset, 88
 - Start, 88
 - Stop, 88
 - Tick, 88
 - Time, 87
 - TotalTime, 88
- FAWindow, 13
- FAWindow.h, 116
- FAWindow::Window, 89
 - CreateChildWindow, 92
 - CreateControlWindow, 93
 - CreateParentWindow, 94

- GetHeight, 95
- GetWidth, 95
- GetWindowHandle, 95
- GetX, 95
- GetY, 95
- SetHeight, 95
- SetWidth, 95
- SetX, 96
- SetY, 96
- Window, 90–92
- FlushCommandQueue
 - FARender::DeviceResources, 33
- Foward
 - FACamera::Camera, 18
- GetAlpha
 - FAColor::Color, 25
- GetAngularVelocity
 - FACamera::Camera, 18
- GetAspectRatio
 - FAProjection::PerspectiveProjection, 50
- GetBackBufferFormat
 - FARender::DeviceResources, 33
 - FARender::SwapChain, 80
- GetBlue
 - FAColor::Color, 25
- GetCameraPosition
 - FACamera::Camera, 18
- GetCameraVelocity
 - FACamera::Camera, 18
- GetCBVSRVUAVSize
 - FARender::DeviceResources, 33
- GetColor
 - FAColor::Color, 25
- GetCommandList
 - FARender::DeviceResources, 33
- GetCurrentBackBuffer
 - FARender::SwapChain, 80
- GetCurrentBackBufferIndex
 - FARender::SwapChain, 80
- GetCurrentFrame
 - FARender::DeviceResources, 33
- GetDepthStencilFormat
 - FARender::DepthStencilBuffer, 30
 - FARender::DeviceResources, 33
 - FARender::MultiSampling, 47
 - FARender::SwapChain, 80
- GetDevice
 - FARender::DeviceResources, 34
- GetDirect2DDeviceContext
 - FARender::TextResources, 86
- GetDirectWriteFactory
 - FARender::TextResources, 86
- GetFar
 - FAProjection::PerspectiveProjection, 50
- GetGPUAddress
 - FARender::DynamicBuffer, 41
- GetGreen
 - FAColor::Color, 25
- GetHeight
 - FAWindow::Window, 95
- GetIndexBufferView
 - FARender::DynamicBuffer, 41
 - FARender::StaticBuffer, 75
- GetInstance
 - FARender::DeviceResources, 34
- GetNear
 - FAProjection::PerspectiveProjection, 50
- GetNumRenderTargetBuffers
 - FARender::SwapChain, 80
- GetProjectionMatrix
 - FAProjection::IProjection, 43
- GetRed
 - FAColor::Color, 26
- GetRenderTargetBuffer
 - FARender::MultiSampling, 47
 - FARender::RenderTargetBuffer, 70
- GetRenderTargetFormat
 - FARender::MultiSampling, 48
 - FARender::RenderTargetBuffer, 70
- GetTextColor
 - FARender::Text, 83
- GetTextLocation
 - FARender::Text, 83
- GetTextResources
 - FARender::DeviceResources, 34
- GetTextSize
 - FARender::Text, 83
- GetTextString
 - FARender::Text, 83
- GetVertexBufferView
 - FARender::DynamicBuffer, 41
 - FARender::StaticBuffer, 75
- GetVerticalFov
 - FAProjection::PerspectiveProjection, 51
- GetViewMatrix
 - FACamera::Camera, 18
- GetWidth
 - FAWindow::Window, 95
- GetWindowHandle
 - FAWindow::Window, 95
- GetX
 - FACamera::Camera, 19
 - FAWindow::Window, 95
- GetY
 - FACamera::Camera, 19
 - FAWindow::Window, 95
- GetZ
 - FACamera::Camera, 19
- IProjection
 - FAProjection::IProjection, 43
- KeyboardInput
 - FACamera::Camera, 19
- KeyboardInputArrow
 - FACamera::Camera, 19
- KeyboardInputWASD

- FACamera::Camera, 20
- Left
 - FACamera::Camera, 20
- LoadShader
 - FARender::RenderScene, 63
- LookAt
 - FACamera::Camera, 20
- MouseInput
 - FACamera::Camera, 21
- MultiSampling
 - FARender::MultiSampling, 44, 45
- NextFrame
 - FARender::DeviceResources, 34
- NUM_OF_FRAMES
 - FARender::DeviceResources, 36
- operator*
 - FAColor, 10
- operator*=
 - FAColor::Color, 26
- operator+
 - FAColor, 11
- operator+=
 - FAColor::Color, 26
- operator-
 - FAColor, 11
- operator-=
 - FAColor::Color, 26
- PerspectiveProjection
 - FAProjection::PerspectiveProjection, 49, 50
- Present
 - FARender::DeviceResources, 35
 - FARender::SwapChain, 81
- ReleaseBuffer
 - FARender::DepthStencilBuffer, 30
 - FARender::DynamicBuffer, 42
 - FARender::RenderTargetBuffer, 70
 - FARender::StaticBuffer, 75
- ReleaseBuffers
 - FARender::MultiSampling, 48
 - FARender::SwapChain, 81
- RemoveShader
 - FARender::RenderScene, 63
- RenderObject
 - FARender::RenderScene, 63
- RenderTargetBuffer
 - FARender::RenderTargetBuffer, 68
- RenderText
 - FARender::RenderScene, 64
- Reset
 - FATime::Time, 88
- ResetBuffers
 - FARender::TextResources, 86
- Resize
 - FARender::DeviceResources, 35
- FARender::RenderScene, 65
- ResizeBuffers
 - FARender::TextResources, 86
- Right
 - FACamera::Camera, 21
- RotateCameraLeftRight
 - FACamera::Camera, 21
- RotateCameraUpDown
 - FACamera::Camera, 21
- RTBufferTransition
 - FARender::DeviceResources, 35
- SetAlpha
 - FAColor::Color, 27
- SetAngularVelocity
 - FACamera::Camera, 22
- SetAspectRatio
 - FAProjection::PerspectiveProjection, 51
- SetBlue
 - FAColor::Color, 27
- SetCameraPosition
 - FACamera::Camera, 22
- SetCameraVelocity
 - FACamera::Camera, 22
- SetColor
 - FAColor::Color, 27
- SetConstants
 - FARender::RenderScene, 65
- SetDynamicBuffer
 - FARender::RenderScene, 65
- SetFar
 - FAProjection::PerspectiveProjection, 51
- SetGreen
 - FAColor::Color, 27
- SetHeight
 - FAWindow::Window, 95
- SetNear
 - FAProjection::PerspectiveProjection, 51
- SetProperties
 - FACamera::Camera, 22
 - FAProjection::PerspectiveProjection, 51
- SetPSOAndRootSignature
 - FARender::RenderScene, 66
- SetRed
 - FAColor::Color, 27
- SetStaticBuffer
 - FARender::RenderScene, 66
- SetTextColor
 - FARender::Text, 83
- SetTextLocation
 - FARender::Text, 83
- SetTextSize
 - FARender::Text, 84
- SetTextString
 - FARender::Text, 84
- SetTexture
 - FARender::RenderScene, 67
- SetVerticalFov
 - FAProjection::PerspectiveProjection, 52

- SetWidth
 - FAWindow::Window, [95](#)
- SetX
 - FACamera::Camera, [23](#)
 - FAWindow::Window, [96](#)
- SetY
 - FACamera::Camera, [23](#)
 - FAWindow::Window, [96](#)
- SetZ
 - FACamera::Camera, [23](#)
- Signal
 - FARender::DeviceResources, [36](#)
- Start
 - FATime::Time, [88](#)
- StaticBuffer
 - FARender::StaticBuffer, [72](#), [73](#)
- Stop
 - FATime::Time, [88](#)
- SwapChain
 - FARender::SwapChain, [77](#)
- Text
 - FARender::Text, [82](#)
- TextResources
 - FARender::TextResources, [85](#)
- Tick
 - FATime::Time, [88](#)
- Time
 - FATime::Time, [87](#)
- TotalTime
 - FATime::Time, [88](#)
- Transition
 - FARender::MultiSampling, [48](#)
 - FARender::SwapChain, [81](#)
- Up
 - FACamera::Camera, [23](#)
- UpdateCurrentFrameFenceValue
 - FARender::DeviceResources, [36](#)
- UpdateProjectionMatrix
 - FAProjection::IProjection, [43](#)
 - FAProjection::PerspectiveProjection, [52](#)
- UpdateViewMatrix
 - FACamera::Camera, [23](#)
- vec2
 - FACamera.h, [102](#)
- WaitForGPU
 - FARender::DeviceResources, [36](#)
- Window
 - FAWindow::Window, [90–92](#)