

Farouq Adepetu's Rendering Engine

Generated by Doxygen 1.9.4



<b>1 Namespace Index</b>	<b>1</b>
1.1 Namespace List	1
<b>2 Class Index</b>	<b>3</b>
2.1 Class List	3
<b>3 File Index</b>	<b>5</b>
3.1 File List	5
<b>4 Namespace Documentation</b>	<b>7</b>
4.1 FACamera Namespace Reference	7
4.1.1 Detailed Description	7
4.2 FARender Namespace Reference	7
4.2.1 Detailed Description	8
4.3 FAWindow Namespace Reference	8
4.3.1 Detailed Description	8
<b>5 Class Documentation</b>	<b>9</b>
5.1 FACamera::Camera Class Reference	9
5.1.1 Detailed Description	11
5.1.2 Constructor & Destructor Documentation	11
5.1.2.1 Camera()	11
5.1.3 Member Function Documentation	11
5.1.3.1 Backward()	12
5.1.3.2 Down()	12
5.1.3.3 Foward()	12
5.1.3.4 GetAngularVelocity()	12
5.1.3.5 GetAspectRatio()	12
5.1.3.6 GetCameraPosition()	12
5.1.3.7 GetCameraVelocity()	13
5.1.3.8 GetPerspectiveProjectionMatrix()	13
5.1.3.9 GetVerticalFov()	13
5.1.3.10 GetViewPerspectiveProjectionMatrix()	13
5.1.3.11 GetViewTransformationMatrix()	13
5.1.3.12 GetX()	13
5.1.3.13 GetY()	14
5.1.3.14 GetZ()	14
5.1.3.15 GetZFar()	14
5.1.3.16 GetZNear()	14
5.1.3.17 KeyboardInput()	14
5.1.3.18 Left()	14
5.1.3.19 LookAt()	15
5.1.3.20 MouseInput()	15
5.1.3.21 Right()	15

5.1.3.22 RotateCameraLeftRight()	15
5.1.3.23 RotateCameraUpDown()	15
5.1.3.24 SetAngularVelocity()	16
5.1.3.25 SetAspectRatio()	16
5.1.3.26 SetCameraPosition()	16
5.1.3.27 SetCameraVelocity()	16
5.1.3.28 SetVerticalFov()	16
5.1.3.29 SetX()	16
5.1.3.30 SetY()	17
5.1.3.31 SetZ()	17
5.1.3.32 SetZFar()	17
5.1.3.33 SetZNear()	17
5.1.3.34 Up()	17
5.1.3.35 UpdatePerspectiveProjectionMatrix()	17
5.1.3.36 UpdateViewMatrix()	18
5.1.3.37 UpdateViewPerspectiveProjectionMatrix()	18
5.2 FColor::Color Class Reference	18
5.2.1 Detailed Description	19
5.2.2 Constructor & Destructor Documentation	19
5.2.2.1 Color() [1/2]	19
5.2.2.2 Color() [2/2]	19
5.2.3 Member Function Documentation	20
5.2.3.1 GetAlpha()	20
5.2.3.2 GetBlue()	20
5.2.3.3 GetColor()	20
5.2.3.4 GetGreen()	20
5.2.3.5 GetRed()	20
5.2.3.6 operator*=( ) [1/2]	21
5.2.3.7 operator*=( ) [2/2]	21
5.2.3.8 operator+=( )	21
5.2.3.9 operator-=( )	21
5.2.3.10 SetAlpha()	21
5.2.3.11 SetBlue()	22
5.2.3.12 SetColor()	22
5.2.3.13 SetGreen()	22
5.2.3.14 SetRed()	22
5.3 FRender::ConstantBuffer Class Reference	22
5.3.1 Detailed Description	23
5.3.2 Constructor & Destructor Documentation	23
5.3.2.1 ~ConstantBuffer()	23
5.3.3 Member Function Documentation	23
5.3.3.1 CopyData()	23

5.3.3.2 CreateConstantBuffer()	24
5.3.3.3 CreateConstantBufferView()	24
5.4 DepthStencil Class Reference	24
5.4.1 Detailed Description	24
5.5 FARender::DepthStencilBuffer Class Reference	24
5.5.1 Detailed Description	25
5.5.2 Constructor & Destructor Documentation	25
5.5.2.1 DepthStencilBuffer() [1/2]	25
5.5.2.2 DepthStencilBuffer() [2/2]	26
5.5.3 Member Function Documentation	26
5.5.3.1 ClearDepthStencilBuffer() [1/2]	26
5.5.3.2 ClearDepthStencilBuffer() [2/2]	26
5.5.3.3 CreateDepthStencilBufferAndView() [1/2]	26
5.5.3.4 CreateDepthStencilBufferAndView() [2/2]	27
5.5.3.5 GetDepthStencilFormat() [1/2]	27
5.5.3.6 GetDepthStencilFormat() [2/2]	27
5.5.3.7 ResetBuffer() [1/2]	27
5.5.3.8 ResetBuffer() [2/2]	27
5.6 FARender::DeviceResources Class Reference	28
5.6.1 Detailed Description	29
5.6.2 Constructor & Destructor Documentation	29
5.6.2.1 ~DeviceResources()	29
5.6.3 Member Function Documentation	29
5.6.3.1 AfterTextDraw()	29
5.6.3.2 BeforeTextDraw()	30
5.6.3.3 DisableMSAA()	30
5.6.3.4 EnableMSAA()	30
5.6.3.5 Execute()	30
5.6.3.6 FlushCommandQueue()	30
5.6.3.7 GetBackBufferFormat()	31
5.6.3.8 GetCBVSize()	31
5.6.3.9 GetCommandList()	31
5.6.3.10 GetCurrentFrame()	31
5.6.3.11 GetDepthStencilFormat()	31
5.6.3.12 GetDevice()	31
5.6.3.13 GetInstance()	32
5.6.3.14 GetTextResources()	32
5.6.3.15 IsMSAAEnabled()	32
5.6.3.16 NextFrame()	32
5.6.3.17 Present()	32
5.6.3.18 Resize()	32
5.6.3.19 RTBufferTransition()	33

5.6.3.20 Signal()	33
5.6.3.21 UpdateCurrentFrameFenceValue()	33
5.6.3.22 WaitForGPU()	33
5.7 DirectXException Class Reference	33
5.8 FARender::DrawSettings Struct Reference	34
5.8.1 Detailed Description	34
5.9 FARender::IndexBuffer Class Reference	34
5.9.1 Detailed Description	34
5.9.2 Member Function Documentation	34
5.9.2.1 CreateIndexBuffer()	35
5.9.2.2 CreateIndexBufferView()	35
5.9.2.3 GetIndexBufferView()	35
5.10 FARender::MultiSampling Class Reference	35
5.10.1 Detailed Description	36
5.10.2 Constructor & Destructor Documentation	36
5.10.2.1 MultiSampling()	36
5.10.3 Member Function Documentation	37
5.10.3.1 ClearDepthStencilBuffer()	37
5.10.3.2 ClearRenderTargetBuffer()	37
5.10.3.3 CreateDepthStencilBufferAndView()	37
5.10.3.4 CreateRenderTargetBufferAndView()	37
5.10.3.5 GetRenderTargetBuffer()	38
5.10.3.6 ResetBuffers()	38
5.10.3.7 Transition()	38
5.11 FARender::RenderScene Class Reference	38
5.11.1 Detailed Description	41
5.11.2 Member Function Documentation	41
5.11.2.1 AddDrawArgument() [1/2]	41
5.11.2.2 AddDrawArgument() [2/2]	41
5.11.2.3 AddIndices() [1/2]	42
5.11.2.4 AddIndices() [2/2]	42
5.11.2.5 AddVertices() [1/2]	42
5.11.2.6 AddVertices() [2/2]	42
5.11.2.7 AfterDraw()	42
5.11.2.8 AfterDrawObjects()	43
5.11.2.9 AfterDrawText()	43
5.11.2.10 BeforeDrawObjects()	43
5.11.2.11 BeforeDrawText()	43
5.11.2.12 CopyData()	43
5.11.2.13 CreateCBVHeap()	44
5.11.2.14 CreateConstantBuffer()	44
5.11.2.15 CreateConstantBufferView()	44

5.11.2.16 CreateDrawSettings()	44
5.11.2.17 CreateIndexBuffer()	44
5.11.2.18 CreateText()	45
5.11.2.19 DisableMSAA()	45
5.11.2.20 DrawObjects()	45
5.11.2.21 EnableMSAA()	45
5.11.2.22 ExecuteAndFlush()	46
5.11.2.23 IsMSAAEnabled()	46
5.11.2.24 NextFrame()	46
5.11.2.25 RemoveDrawArgument()	46
5.11.2.26 RemoveDrawSettings()	46
5.11.2.27 RemoveText()	47
5.11.2.28 RenderText()	47
5.11.2.29 Resize()	47
5.11.2.30 SetPrimitive()	47
5.11.2.31 SetPSO()	48
5.11.2.32 SetRootSignature()	48
5.12 FARender::RenderTargetBuffer Class Reference	48
5.12.1 Detailed Description	49
5.12.2 Constructor & Destructor Documentation	49
5.12.2.1 RenderTargetBuffer()	49
5.12.3 Member Function Documentation	49
5.12.3.1 ClearRenderTargetBuffer()	49
5.12.3.2 CreateRenderTargetBufferAndView()	49
5.12.3.3 GetRenderTargetBuffer() [1/2]	50
5.12.3.4 GetRenderTargetBuffer() [2/2]	50
5.12.3.5 GetRenderTargetFormat()	50
5.12.3.6 ResetBuffer()	50
5.13 FARender::SwapChain Class Reference	50
5.13.1 Detailed Description	51
5.13.2 Constructor & Destructor Documentation	51
5.13.2.1 SwapChain()	52
5.13.3 Member Function Documentation	52
5.13.3.1 ClearCurrentBackBuffer()	52
5.13.3.2 ClearDepthStencilBuffer()	52
5.13.3.3 CreateDepthStencilBufferAndView()	52
5.13.3.4 CreateRenderTargetBuffersAndViews()	53
5.13.3.5 GetBackBufferFormat()	53
5.13.3.6 GetCurrentBackBuffer()	53
5.13.3.7 GetCurrentBackBufferIndex()	53
5.13.3.8 GetDepthStencilFormat()	53
5.13.3.9 GetNumRenderTargetBuffers()	53

5.13.3.10 GetRenderTargetBuffers()	54
5.13.3.11 Present()	54
5.13.3.12 ResetBuffers()	54
5.13.3.13 ResizeSwapChain()	54
5.13.3.14 Transition()	54
5.14 FARender::Text Class Reference	55
5.14.1 Detailed Description	55
5.14.2 Constructor & Destructor Documentation	55
5.14.2.1 Text()	55
5.14.3 Member Function Documentation	56
5.14.3.1 GetTextColor()	56
5.14.3.2 GetTextLocation()	56
5.14.3.3 GetTextSize()	56
5.14.3.4 GetTextString()	56
5.14.3.5 SetTextColor()	56
5.14.3.6 SetTextLocation()	57
5.14.3.7 SetTextSize()	57
5.14.3.8 SetTextString()	57
5.15 FARender::TextResources Class Reference	57
5.15.1 Detailed Description	58
5.15.2 Constructor & Destructor Documentation	58
5.15.2.1 TextResources()	58
5.15.3 Member Function Documentation	58
5.15.3.1 AfterRenderText()	58
5.15.3.2 BeforeRenderText()	58
5.15.3.3 GetDirect2DDeviceContext()	58
5.15.3.4 GetDirectWriteFactory()	59
5.15.3.5 ResetBuffers()	59
5.15.3.6 ResizeBuffers()	59
5.16 FATime::Time Class Reference	59
5.16.1 Constructor & Destructor Documentation	60
5.16.1.1 Time()	60
5.16.2 Member Function Documentation	60
5.16.2.1 DeltaTime()	60
5.16.2.2 Reset()	60
5.16.2.3 Start()	60
5.16.2.4 Stop()	60
5.16.2.5 Tick()	61
5.16.2.6 TotalTime()	61
5.17 Time Class Reference	61
5.17.1 Detailed Description	61
5.18 FARender::VertexBuffer Class Reference	61



5.18.1 Detailed Description	62
5.18.2 Member Function Documentation	62
5.18.2.1 CreateVertexBuffer()	62
5.18.2.2 CreateVertexBufferView()	62
5.18.2.3 GetVertexBufferView()	62
5.19 FAWindow::Window Class Reference	62
5.19.1 Detailed Description	63
5.19.2 Constructor & Destructor Documentation	63
5.19.2.1 Window() [1/2]	63
5.19.2.2 Window() [2/2]	64
5.19.3 Member Function Documentation	64
5.19.3.1 GetHeight()	64
5.19.3.2 GetWidth()	64
5.19.3.3 GetWindowHandle()	64
5.19.3.4 SetHeight()	64
5.19.3.5 SetWidth()	64
<b>6 File Documentation</b>	<b>65</b>
6.1 Direct3DLink.h	65
6.2 FABuffer.h File Reference	65
6.2.1 Detailed Description	66
6.3 FABuffer.h	66
6.4 FACamera.h File Reference	67
6.4.1 Detailed Description	68
6.4.2 Typedef Documentation	68
6.4.2.1 vec2	68
6.5 FACamera.h	68
6.6 FAColor.h File Reference	70
6.6.1 Detailed Description	70
6.6.2 Function Documentation	70
6.6.2.1 operator*() [1/3]	71
6.6.2.2 operator*() [2/3]	71
6.6.2.3 operator*() [3/3]	71
6.6.2.4 operator+()	71
6.6.2.5 operator-()	71
6.7 FAColor.h	72
6.8 FADepthStencil.h	72
6.9 FADeviceResources.h File Reference	73
6.9.1 Detailed Description	73
6.10 FADeviceResources.h	73
6.11 FADirectXException.h	75
6.12 FAMultiSampling.h	75

6.13 FARenderScene.h File Reference . . . . .	76
6.13.1 Detailed Description . . . . .	77
6.14 FARenderScene.h . . . . .	77
6.15 FASwapChain.h . . . . .	80
6.16 FAText.h File Reference . . . . .	81
6.16.1 Detailed Description . . . . .	81
6.17 FAText.h . . . . .	82
6.18 FATextResources.h . . . . .	82
6.19 FATime.h File Reference . . . . .	83
6.19.1 Detailed Description . . . . .	83
6.20 FATime.h . . . . .	83
6.21 FAWindow.h File Reference . . . . .	84
6.21.1 Detailed Description . . . . .	84
6.22 FAWindow.h . . . . .	84
<b>Index</b>	<b>85</b>

# Chapter 1

## Namespace Index

### 1.1 Namespace List

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

<a href="#">FACamera</a>		
Has <a href="#">Camera</a> class	.....	<a href="#">7</a>
<a href="#">FARender</a>		
Has classes that are used for rendering objects and text through the Direct3D 12 API	.....	<a href="#">7</a>
<a href="#">FAWindow</a>		
Has <a href="#">Window</a> class	.....	<a href="#">8</a>



## Chapter 2

# Class Index

### 2.1 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.

It keeps track of the viewing frustum of the camera so that the projection matrix can be obtained.

9

#### [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 . . . . .

18

#### [FARender::ConstantBuffer](#)

This class stores constant data in a Direct3D 12 upload buffer . . . . .

22

#### [DepthStencil](#)

A wrapper for depth stencil buffer resources. Uses DirectD 12 API . . . . .

24

#### [FARender::DepthStencilBuffer](#)

A wrapper for depth stencil buffer resources. Uses DirectD 12 API . . . . .

24

#### [FARender::DeviceResources](#)

A wrapper for resources that are needed to render objects and text using the Direct3D 12 API .

28

#### [DirectXException](#)

. . . . .

33

#### [FARender::DrawSettings](#)

Holds a array of objects that use the same PSO, root signature and primitive . . . . .

34

#### [FARender::IndexBuffer](#)

This class stores indices in a Direct3D 12 default buffer . . . . .

34

#### [FARender::MultiSampling](#)

A wrapper for multisampling resources. Uses DirectD 12 API . . . . .

35

#### [FARender::RenderScene](#)

This class is used to render a scene using Direct3D 12 API . . . . .

38

#### [FARender::RenderTargetBuffer](#)

A wrapper for render target buffer resources. Uses DirectD 12 API . . . . .

48

#### [FARender::SwapChain](#)

A wrapper for swap chain resources. Uses DirectD 12 API and DXGI API . . . . .

50

#### [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 . . . . .

55

[FARender::TextResources](#)

A wrapper for resources that are needed to render text using Direct3D 11on12, Direct2D and DirectWrite . . . . . 57

[FATime::Time](#) . . . . . 59[Time](#)

This class is used to get the time between each frame. You can stop start, reset and get the total time . . . . . 61

[FARender::VertexBuffer](#)

This class stores vertices in a Direct3D 12 default buffer . . . . . 61

[FAWindow::Window](#)

The window class is used to make a [Window](#) using Windows API . . . . . 62

## Chapter 3

# File Index

### 3.1 File List

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

<a href="#">Direct3DLink.h</a>	??
<a href="#">FABuffer.h</a>	
File has classes VertexBuffer, IndexBuffer and ConstantBuffer under namespace <a href="#">FARender</a>	65
<a href="#">FACamera.h</a>	
File that has namespace <a href="#">FACamera</a> . Withn the namespace is the class Camera	67
<a href="#">FAColor.h</a>	
File has class Color under namespace <a href="#">FAColor</a>	70
<a href="#">FADepthStencil.h</a>	??
<a href="#">FADeviceResources.h</a>	
File has class DeviceResources under namespace <a href="#">FARender</a>	73
<a href="#">FADirectXException.h</a>	??
<a href="#">FAMultiSampling.h</a>	??
<a href="#">FARenderScene.h</a>	
File has class RenderScene under namespace <a href="#">FARender</a>	76
<a href="#">FASwapChain.h</a>	??
<a href="#">FAText.h</a>	
File has class Text under namespace <a href="#">FARender</a>	81
<a href="#">FATextResources.h</a>	??
<a href="#">FATime.h</a>	
File that has namespace <a href="#">FATime</a> . Withn the namespace is the class <a href="#">Time</a>	83
<a href="#">FAWindow.h</a>	
File that has namespace <a href="#">FAWindow</a> . Withn the namespace is the class Window	84





## Chapter 4

# Namespace Documentation

### 4.1 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.*

*It keeps track of the viewing frustum of the camera so that the projection matrix can be obtained.*

.

#### 4.1.1 Detailed Description

Has [Camera](#) class.

### 4.2 FARender Namespace Reference

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

#### Classes

- class [ConstantBuffer](#)

*This class stores constant data in a Direct3D 12 upload buffer.*

- class [DepthStencilBuffer](#)

*A wrapper for depth stencil buffer resources. Uses DirectD 12 API.*

- class [DeviceResources](#)

*A wrapper for resources that are needed to render objects and text using the Direct3D 12 API.*

- struct [DrawSettings](#)

*Holds a array of objects that use the same PSO, root signature and primitive.*

- class [IndexBuffer](#)  
*This class stores indices in a Direct3D 12 default buffer.*
- class [MultiSampling](#)  
*A wrapper for multisampling resources. Uses DirectD 12 API.*
- class [RenderScene](#)  
*This class is used to render a scene using Direct3D 12 API.*
- class [RenderTargetBuffer](#)  
*A wrapper for render target buffer resources. Uses DirectD 12 API.*
- class [SwapChain](#)  
*A wrapper for swap chain resources. Uses DirectD 12 API and DXGI API.*
- 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.*
- class [VertexBuffer](#)  
*This class stores vertices in a Direct3D 12 default buffer.*

#### 4.2.1 Detailed Description

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

### 4.3 FAWindow Namespace Reference

Has [Window](#) class.

#### Classes

- class [Window](#)  
*The window class is used to make a [Window](#) using Windows API.*

#### 4.3.1 Detailed Description

Has [Window](#) class.

## Chapter 5

# Class Documentation

### 5.1 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.

It keeps track of the viewing frustum of the camera so that the projection matrix can be obtained.

```
#include "FACamera.h"
```

#### Public Member Functions

- [Camera](#) (vec3 cameraPosition=vec3(0.0f, 0.0f, 0.0f), vec3 x=vec3(1.0f, 0.0f, 0.0f), vec3 y=vec3(0.0f, 1.0f, 0.0f), vec3 z=vec3(0.0f, 0.0f, 1.0f), float znear=1.0f, float zfar=100.f, float aspectRatio=1.0f, float vFov=45.0f, float cameraVelocity=10.0f, float angularVelocity=0.25f)  
*Constructor.*
- 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 & [GetViewTransformationMatrix](#) () 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.*
- float [GetZNear](#) () const  
*Returns the near value of the frustum.*
- float [GetZFar](#) () 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 [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.*
- void [SetY](#) (const vec3 &y)

*Sets the camera's y-axis to the specified vector.*
- void [SetZ](#) (const vec3 &z)

*Sets the camera's z-axis to the specified vector.*
- 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 [SetZNear](#) (float znear)

*Sets the camera's near plane z value to the specified value.*
- void [SetZFar](#) (float zfar)

*Sets the camera's far plane z 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.*
- const mat4 & [GetPerspectiveProjectionMatrix](#) () const

*Returns a constant reference to the perspective projection transformation matrix of this camera.*
- const mat4 & [GetViewPerspectiveProjectionMatrix](#) () const

*Returns a constant reference to the view perspective projection transformation matrix of this camera.*
- void [UpdateViewMatrix](#) ()

*After modifying the camera position and/or orientation, call this to rebuild the view transformation matrix.*
- void [UpdatePerspectiveProjectionMatrix](#) ()

*After modifying any of the frustrum properties, call this to rebuild the perspective projection transformation matrix.*
- void [UpdateViewPerspectiveProjectionMatrix](#) ()

*After modifying view and/or perspective projection transformation matrix, call this to rebuild the view perspective projection 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. 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.*

- void [MouseInput](#) ()

*Rotates camera on mouse movement.*

### 5.1.1 Detailed Description

Simple first person style camera class that lets the viewer explore the 3D scene.

It keeps track of the camera coordinate system relative to the world space so that the view matrix can be constructed.

It keeps track of the viewing frustum of the camera so that the projection matrix can be obtained.

### 5.1.2 Constructor & Destructor Documentation

#### 5.1.2.1 Camera()

```
FACamera::Camera::Camera (
    vec3 cameraPosition = vec3(0.0f, 0.0f, 0.0f),
    vec3 x = vec3(1.0f, 0.0f, 0.0f),
    vec3 y = vec3(0.0f, 1.0f, 0.0f),
    vec3 z = vec3(0.0f, 0.0f, 1.0f),
    float znear = 1.0f,
    float zfar = 100.f,
    float aspectRatio = 1.0f,
    float vFov = 45.0f,
    float cameraVelocity = 10.0f,
    float angularVelocity = 0.25f )
```

Constructor.

Creates a new camera.

Sets the origin of the camera space to the given cameraPosition.

Sets the axis of the camera space to the given x, y and z vectors.

The origin and basis vectors of the camera space should be relative to world space.

Sets the frustum properties for perspective projection to the given znear, zar, aspectRatio and fov values.

vFov should be in degrees.

The constant velocity of the camera when moved is set to the given cameraVelocity; The angular velocity of the camera is set the to specified angularVelocity.

### 5.1.3 Member Function Documentation

#### 5.1.3.1 Backward()

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

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

#### 5.1.3.2 Down()

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

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

#### 5.1.3.3 Foward()

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

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

#### 5.1.3.4 GetAngularVelocity()

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

Returns the camera's angular velocity.

#### 5.1.3.5 GetAspectRatio()

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

Returns the aspect ratio of the frustum.

#### 5.1.3.6 GetCameraPosition()

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

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

#### 5.1.3.7 GetCameraVelocity()

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

Returns the camera's velocity.

#### 5.1.3.8 GetPerspectiveProjectionMatrix()

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

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

#### 5.1.3.9 GetVerticalFov()

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

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

#### 5.1.3.10 GetViewPerspectiveProjectionMatrix()

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

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

#### 5.1.3.11 GetViewTransformationMatrix()

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

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

#### 5.1.3.12 GetX()

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

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

#### 5.1.3.13 GetY()

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

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

#### 5.1.3.14 GetZ()

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

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

#### 5.1.3.15 GetZFar()

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

Returns the far value of the frustum.

#### 5.1.3.16 GetZNear()

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

Returns the near value of the frustum.

#### 5.1.3.17 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.

#### 5.1.3.18 Left()

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

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



#### 5.1.3.19 LookAt()

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

Defines the camera space using UVN.

#### 5.1.3.20 MouseInput()

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

Rotates camera on mouse movement.

#### 5.1.3.21 Right()

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

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

#### 5.1.3.22 RotateCameraLeftRight()

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

Rotates the camera to look left and right.

#### 5.1.3.23 RotateCameraUpDown()

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

Rotates the camera to look up and down.

#### 5.1.3.24 SetAngularVelocity()

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

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

#### 5.1.3.25 SetAspectRatio()

```
void FACamera::Camera::SetAspectRatio (
    float ar )
```

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

#### 5.1.3.26 SetCameraPosition()

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

Sets the camera's position to the specified position.

#### 5.1.3.27 SetCameraVelocity()

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

Sets the camera's velocity to the specified velocity.

#### 5.1.3.28 SetVerticalFov()

```
void FACamera::Camera::SetVerticalFov (
    float fov )
```

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

#### 5.1.3.29 SetX()

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

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

#### 5.1.3.30 SetY()

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

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

#### 5.1.3.31 SetZ()

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

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

#### 5.1.3.32 SetZFar()

```
void FACamera::Camera::SetZFar (
    float zfar )
```

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

#### 5.1.3.33 SetZNear()

```
void FACamera::Camera::SetZNear (
    float znear )
```

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

#### 5.1.3.34 Up()

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

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

#### 5.1.3.35 UpdatePerspectiveProjectionMatrix()

```
void FACamera::Camera::UpdatePerspectiveProjectionMatrix ( )
```

After modifying any of the frustrum properties, call this to rebuild the perspective projection transformation matrix.

### 5.1.3.36 UpdateViewMatrix()

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

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

### 5.1.3.37 UpdateViewPerspectiveProjectionMatrix()

```
void FACamera::Camera::UpdateViewPerspectiveProjectionMatrix ( )
```

After modifying view and/or perspective projection transformation matrix, call this to rebuild the view perspective projection transformation matrix.

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

- [FACamera.h](#)

## 5.2 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)  
*Default Constructor. Initializes the color to the specified RGBA values.*
- [Color](#) (const FAMath::Vector4D &color)  
*Overloaded Constructor. 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 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.*
  - **Color & operator-=** (const **Color** &c)

*Subtracts the specified color 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.*
  - **Color & operator\*=** (float k)

*Multiplies this objects color by the specified float 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.*
  - **Color & operator\*=** (const **Color** &c)

*Multiplies this objects color by the specified color c and stores the result in this object. If any of the resultant components are > 1.0f, they are set to 1.0f. Does component-wise multiplication.*

### 5.2.1 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.

### 5.2.2 Constructor & Destructor Documentation

#### 5.2.2.1 Color() [1/2]

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

Default Constructor. Initializes the color to the specified RGBA values.

#### 5.2.2.2 Color() [2/2]

```
FColor::Color::Color (
    const FMath::Vector4D & color )
```

Overloaded Constructor. Initializes the color to the specified color.

## 5.2.3 Member Function Documentation

### 5.2.3.1 GetAlpha()

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

Returns the value of the alpha component.

### 5.2.3.2 GetBlue()

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

Returns the value of the green component.

### 5.2.3.3 GetColor()

```
const FMath::Vector4D & FColor::Color::GetColor ( ) const
```

Returns the color.

### 5.2.3.4 GetGreen()

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

Returns the value of the blue component.

### 5.2.3.5 GetRed()

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

Returns the value of the red component.

### 5.2.3.6 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.

### 5.2.3.7 operator\*=( ) [2/2]

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

Multiplies this objects color by the specified float value k and stores the result in this object. If  $k < 0.0f$ , no multiplication happens and this objects color does not get modified. If any of the resultant components are  $> 1.0f$ , they are set to  $1.0f$ .

.

### 5.2.3.8 operator+=( )

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

Adds this objects color to the specified color and stores the result in this object. Does component-wise addition. If any of the resultant components are  $> 1.0f$ , they are set to  $1.0f$ .

### 5.2.3.9 operator-=( )

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

Subtracts the specified color from this objects color and stores the result in this object. Does component-wise subtraction. If any of the resultant components are  $< 0.0f$ , they are set to  $0.0f$ .

### 5.2.3.10 SetAlpha()

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

Sets the alpha component to the specified float value.

#### 5.2.3.11 SetBlue()

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

Sets the blue component to the specified float value.

#### 5.2.3.12 SetColor()

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

Sets the color to the specified color.

#### 5.2.3.13 SetGreen()

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

Sets the green component to the specified float value.

#### 5.2.3.14 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](#)

## 5.3 FRender::ConstantBuffer Class Reference

This class stores constant data in a Direct3D 12 upload buffer.

```
#include "FABuffer.h"
```



## Public Member Functions

- **ConstantBuffer** (const [ConstantBuffer](#) &)=delete
- **ConstantBuffer & operator=** (const [ConstantBuffer](#) &)=delete
- **~ConstantBuffer** ()  
*Unmaps the pointer to the constant buffer.*
- void **CreateConstantBuffer** (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const UINT &numOfBytes)  
*Creates and maps the constant buffer. The number of bytes allocated should be a multiple of 256 bytes.*
- void **CreateConstantBufferView** (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &cbvHeap, UINT cbvSize, UINT cBufferIndex, UINT cbvHeapIndex, UINT numBytes)  
*Creates and maps the constant buffer view and stores it in the specified descriptor heap.*
- void **CopyData** (UINT index, UINT byteSize, const void \*data, UINT64 numOfBytes)  
*Copies data from the given data into the constant buffer. Uses 0-indexing.*

### 5.3.1 Detailed Description

This class stores constant data in a Direct3D 12 upload buffer.

### 5.3.2 Constructor & Destructor Documentation

#### 5.3.2.1 ~ConstantBuffer()

```
FARender::ConstantBuffer::~ConstantBuffer ( )
```

Unmaps the pointer to the constant buffer.

### 5.3.3 Member Function Documentation

#### 5.3.3.1 CopyData()

```
void FAREnder::ConstantBuffer::CopyData (
    UINT index,
    UINT byteSize,
    const void * data,
    UINT64 numOfBytes )
```

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

### 5.3.3.2 CreateConstantBuffer()

```
void FARender::ConstantBuffer::CreateConstantBuffer (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
    const UINT & numOfBytes )
```

Creates and maps the constant buffer. The number of bytes allocated should be a multiple of 256 bytes.

### 5.3.3.3 CreateConstantBufferView()

```
void FARender::ConstantBuffer::CreateConstantBufferView (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
    const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & cbvHeap,
    UINT cbvSize,
    UINT cBufferIndex,
    UINT cbvHeapIndex,
    UINT numBytes )
```

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

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

- [FABuffer.h](#)

## 5.4 DepthStencil Class Reference

A wrapper for depth stencil buffer resources. Uses DirectD 12 API.

```
#include "FADepthStencil.h"
```

### 5.4.1 Detailed Description

A wrapper for depth stencil buffer resources. Uses DirectD 12 API.

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

- FADepthStencil.h

## 5.5 FARender::DepthStencilBuffer Class Reference

A wrapper for depth stencil buffer resources. Uses DirectD 12 API.

```
#include "FABuffer.h"
```

## Public Member Functions

- [DepthStencilBuffer](#) (DXGI\_FORMAT format=DXGI\_FORMAT\_D24\_UNORM\_S8\_UINT)  
*Default Constructor.*
- 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 indexWhereToStoreView, unsigned int dsvSize, unsigned int width, unsigned int height, unsigned int sampleCount=1)  
*Creates the depth stencil buffer and view.*
- void [ResetBuffer](#) ()  
*Resest the depth stencil 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.*
- [DepthStencilBuffer](#) (DXGI\_FORMAT format=DXGI\_FORMAT\_D24\_UNORM\_S8\_UINT)  
*Default Constructor.*
- 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 indexWhereToStoreView, unsigned int dsvSize, unsigned int width, unsigned int height)  
*Creates the depth stencil buffer and view.*
- void [ResetBuffer](#) ()  
*Resest the depth stencil 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.*

### 5.5.1 Detailed Description

A wrapper for depth stencil buffer resources. Uses DirectD 12 API.

### 5.5.2 Constructor & Destructor Documentation

#### 5.5.2.1 DepthStencilBuffer() [1/2]

```
FARender::DepthStencilBuffer::DepthStencilBuffer (
    DXGI_FORMAT format = DXGI_FORMAT_D24_UNORM_S8_UINT )
```

Default Constructor.

### 5.5.2.2 DepthStencilBuffer() [2/2]

```
FARender::DepthStencilBuffer::DepthStencilBuffer (
    DXGI_FORMAT format = DXGI_FORMAT_D24_UNORM_S8_UINT )
```

Default Constructor.

## 5.5.3 Member Function Documentation

### 5.5.3.1 ClearDepthStencilBuffer() [1/2]

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

Clears the depth stencil buffer with the specified clear value.

### 5.5.3.2 ClearDepthStencilBuffer() [2/2]

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

Clears the depth stencil buffer with the specified clear value.

### 5.5.3.3 CreateDepthStencilBufferAndView() [1/2]

```
void FARender::DepthStencilBuffer::CreateDepthStencilBufferAndView (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
    const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & dsvHeap,
    unsigned int indexOfWhereToStoreView,
    unsigned int dsvSize,
    unsigned int width,
    unsigned int height )
```

Creates the depth stencil buffer and view.

#### 5.5.3.4 CreateDepthStencilBufferAndView() [2/2]

```
void FAREnder::DepthStencilBuffer::CreateDepthStencilBufferAndView (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
    const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & dsvHeap,
    unsigned int indexOfWhereToStoreView,
    unsigned int dsvSize,
    unsigned int width,
    unsigned int height,
    unsigned int sampleCount = 1 )
```

Creates the depth stencil buffer and view.

#### 5.5.3.5 GetDepthStencilFormat() [1/2]

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

Returns the format of the depth stencil buffer.

#### 5.5.3.6 GetDepthStencilFormat() [2/2]

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

Returns the format of the depth stencil buffer.

#### 5.5.3.7 ResetBuffer() [1/2]

```
void FAREnder::DepthStencilBuffer::ResetBuffer ( )
```

Resest the depth stencil buffer.

#### 5.5.3.8 ResetBuffer() [2/2]

```
void FAREnder::DepthStencilBuffer::ResetBuffer ( )
```

Resest the depth stencil buffer.

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

- [FABuffer.h](#)
- [FADepthStencil.h](#)

## 5.6 FAREnder::DeviceResources Class Reference

A wrapper for resources that are needed to render objects and text using the Direct3D 12 API.

```
#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.*
- UINT **GetCBVSize** () const  
*The size of a constant buffer view.*
- unsigned int **GetCurrentFrame** () const  
*Returns the current frame.*
- const [TextResources](#) & **GetTextResources** () const  
*Returns a constant reference to the [TextResources](#) object.*
- bool **IsMSAAEnabled** () const  
*Returns true if MSAA is enabled, false otherwise.*
- void **DisableMSAA** (unsigned int width, unsigned int height, HWND windowHandle)  
*Disables MSAA.*
- void **EnableMSAA** (unsigned int width, unsigned int height, HWND windowHandle)  
*Enables MSAA.*
- void **UpdateCurrentFrameFenceValue** ()  
*Updates the current frames fence value.*
- void **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.*
- void **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.*
- 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)  
*Call when the window gets resized. Call when you initialize your program.*
- void **RTBufferTransition** (bool renderText)  
*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** ()
- 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)  
*Call to make an object of [DeviceResources](#). This only allows one instance to exist.*

## Static Public Attributes

- static const unsigned int **NUM\_OF\_FRAMES** { 3 }

### 5.6.1 Detailed Description

A wrapper for resources that are needed to render objects and text using the Direct3D 12 API.

### 5.6.2 Constructor & Destructor Documentation

#### 5.6.2.1 ~DeviceResources()

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

Flushes the command queue.

### 5.6.3 Member Function Documentation

#### 5.6.3.1 AfterTextDraw()

```
void FArender::DeviceResources::AfterTextDraw ( )
```

Executes the text commands.

#### 5.6.3.2 BeforeTextDraw()

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

Prepares to render text.

#### 5.6.3.3 DisableMSAA()

```
void FARender::DeviceResources::DisableMSAA (
    unsigned int width,
    unsigned int height,
    HWND windowHandle )
```

Disables MSAA.

#### 5.6.3.4 EnableMSAA()

```
void FARender::DeviceResources::EnableMSAA (
    unsigned int width,
    unsigned int height,
    HWND windowHandle )
```

Enables MSAA.

#### 5.6.3.5 Execute()

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

Executes the command list.

#### 5.6.3.6 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.



#### 5.6.3.7 GetBackBufferFormat()

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

Returns a constant reference to the back buffer format.

#### 5.6.3.8 GetCBVSize()

```
UINT FARender::DeviceResources::GetCBVSize ( ) const
```

The size of a constant buffer view.

#### 5.6.3.9 GetCommandList()

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

Returns a constant reference to the ID3D12GraphicsCommandList object.

#### 5.6.3.10 GetCurrentFrame()

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

Returns the current frame.

#### 5.6.3.11 GetDepthStencilFormat()

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

Returns a constant reference to the depth stencil format.

#### 5.6.3.12 GetDevice()

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

Returns a constant reference to the ID3D12Device object.

#### 5.6.3.13 GetInstance()

```
static DeviceResources & FARender::DeviceResources::GetInstance (
    unsigned int width,
    unsigned int height,
    HWND windowHandle ) [static]
```

Call to make an object of [DeviceResources](#). This only allows one instance to exist.

#### 5.6.3.14 GetTextResources()

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

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

#### 5.6.3.15 IsMSAAEnabled()

```
bool FARender::DeviceResources::IsMSAAEnabled ( ) const
```

Returns true if MSAA is enabled, false otherwise.

#### 5.6.3.16 NextFrame()

```
void FARender::DeviceResources::NextFrame ( )
```

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

#### 5.6.3.17 Present()

```
void FARender::DeviceResources::Present ( )
```

Swaps the front and back buffers.

#### 5.6.3.18 Resize()

```
void FARender::DeviceResources::Resize (
    int width,
    int height,
    const HWND & handle )
```

Call when the window gets resized. Call when you initialize your program.

#### 5.6.3.19 RTBufferTransition()

```
void FARender::DeviceResources::RTBufferTransition (
    bool renderText )
```

Transitions the render target buffer.

#### 5.6.3.20 Signal()

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

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

#### 5.6.3.21 UpdateCurrentFrameFenceValue()

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

Updates the current frames fence value.

#### 5.6.3.22 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.

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

- [FADeviceResources.h](#)

## 5.7 DirectXException Class Reference

### Public Member Functions

- **DirectXException** (HRESULT hr, const std::wstring &functionName, const std::wstring &fileName, int line↵  
Number)
- std::wstring **ErrorMsg** () const

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

- FADirectXException.h

## 5.8 FARender::DrawSettings Struct Reference

Holds a array of objects that use the same PSO, root signature and primitive.

```
#include "FARenderScene.h"
```

### Public Attributes

- Microsoft::WRL::ComPtr< ID3D12PipelineState > **pipelineState**
- Microsoft::WRL::ComPtr< ID3D12RootSignature > **rootSig**
- D3D\_PRIMITIVE\_TOPOLOGY **prim** = D3D\_PRIMITIVE\_TOPOLOGY\_TRIANGLELIST
- std::vector< FAShapes::DrawArguments > **drawArgs**

### 5.8.1 Detailed Description

Holds a array of objects that use the same PSO, root signature and primitive.

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

- [FARenderScene.h](#)

## 5.9 FARender::IndexBuffer Class Reference

This class stores indices in a Direct3D 12 default buffer.

```
#include "FABuffer.h"
```

### Public Member Functions

- **IndexBuffer** (const [IndexBuffer](#) &)=delete
- **IndexBuffer** & **operator=** (const [IndexBuffer](#) &)=delete
- const D3D12\_INDEX\_BUFFER\_VIEW & [GetIndexBufferView](#) ()  
*Returns a constant reference to the vertex buffer view.*
- void [CreateIndexBuffer](#) (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const void \*data, UINT numBytes)  
*Creates the vertex buffer and stores all of the specified vertices in the vertex buffer.*
- void [CreateIndexBufferView](#) (UINT numBytes, DXGI\_FORMAT format)  
*Creates the vertex buffer view and stores it.*

### 5.9.1 Detailed Description

This class stores indices in a Direct3D 12 default buffer.

### 5.9.2 Member Function Documentation

### 5.9.2.1 CreateIndexBuffer()

```
void FARender::IndexBuffer::CreateIndexBuffer (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
    const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & commandList,
    const void * data,
    UINT numBytes )
```

Creates the vertex buffer and stores all of the specified vertices in the vertex buffer.

### 5.9.2.2 CreateIndexBufferView()

```
void FARender::IndexBuffer::CreateIndexBufferView (
    UINT numBytes,
    DXGI_FORMAT format )
```

Creates the vertex buffer view and stores it.

### 5.9.2.3 GetIndexBufferView()

```
const D3D12_INDEX_BUFFER_VIEW & FARender::IndexBuffer::GetIndexBufferView ( )
```

Returns a constant reference to the vertex buffer view.

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

- [FABuffer.h](#)

## 5.10 FARender::MultiSampling Class Reference

A wrapper for multisampling resources. Uses DirectD 12 API.

```
#include "FAMultiSampling.h"
```

## Public Member Functions

- [MultiSampling](#) (const Microsoft::WRL::ComPtr< ID3D12Device > &device, DXGI\_FORMAT rtFormat, DXGI\_FORMAT dsFormat, unsigned int sampleCount)  
*Constructor. 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.*
- const Microsoft::WRL::ComPtr< ID3D12Resource > & [GetRenderTargetBuffer](#) ()  
*Returns the MSAA render target buffer.*
- DXGI\_FORMAT [GetRenderTargetFormat](#) ()
- DXGI\_FORMAT [GetDepthStencilFormat](#) ()
- void [ResetBuffers](#) ()  
*Resets the MSAA render target buffer and MSAA depth stencil buffer.*
- void [CreateRenderTargetBufferAndView](#) (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &rtvHeap, unsigned int indexWhereToStoreView, unsigned int rtvSize, unsigned int width, unsigned int height)  
*Creates the MSAA render target buffer and a view to it.*
- void [CreateDepthStencilBufferAndView](#) (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &dsvHeap, unsigned int indexWhereToStoreView, unsigned int dsvSize, unsigned int width, unsigned int height)  
*Creates the MSAA depth stencil buffer and a view to it.*
- void [Transition](#) (const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, D3D12\_RESOURCE\_STATES before, D3D12\_RESOURCE\_STATES after)  
*Transitions the MSAA render target buffer from the specified before state to the specified after state.*
- void [ClearRenderTargetBuffer](#) (const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &rtvHeap, unsigned int indexOfView, unsigned int rtvSize, const float \*clearValue)  
*Clears the MSAA render target buffer with the specified clear value.*
- void [ClearDepthStencilBuffer](#) (const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &dsvHeap, unsigned int indexOfView, unsigned int dsvSize, float clearValue)  
*Clears the MSAA depth stencil buffer with the specified clear value.*

### 5.10.1 Detailed Description

A wrapper for multisampling resources. Uses DirectD 12 API.

### 5.10.2 Constructor & Destructor Documentation

#### 5.10.2.1 MultiSampling()

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

Constructor. 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.

### 5.10.3 Member Function Documentation

#### 5.10.3.1 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.

#### 5.10.3.2 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.

#### 5.10.3.3 CreateDepthStencilBufferAndView()

```
void FAREnder::MultiSampling::CreateDepthStencilBufferAndView (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
    const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & dsvHeap,
    unsigned int indexOfWhereToStoreView,
    unsigned int dsvSize,
    unsigned int width,
    unsigned int height )
```

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

#### 5.10.3.4 CreateRenderTargetBufferAndView()

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

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

#### 5.10.3.5 GetRenderTargetBuffer()

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

Returns the MSAA render target buffer.

#### 5.10.3.6 ResetBuffers()

```
void FARender::MultiSampling::ResetBuffers ( )
```

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

#### 5.10.3.7 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.

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

- FAMultiSampling.h

## 5.11 FARender::RenderScene Class Reference

This class is used to render a scene using Direct3D 12 API.

```
#include "FARenderScene.h"
```



## Public Member Functions

- **RenderScene** (unsigned int width, unsigned int height, HWND windowHandle)
- **RenderScene** (const [RenderScene](#) &)=delete
- **RenderScene & operator=** (const [RenderScene](#) &)=delete
- const [DeviceResources](#) & **GetDeviceResources** () const
- const Microsoft::WRL::ComPtr< ID3DBlob > & **GetShader** (const std::wstring &name) const
- const std::vector< D3D12\_INPUT\_ELEMENT\_DESC > & **GetInputElementLayout** (const std::wstring &name) const
- const D3D12\_RASTERIZER\_DESC & **GetRasterizationState** (const std::wstring &name) const
- const Microsoft::WRL::ComPtr< ID3D12PipelineState > & **GetPSO** (const std::wstring &drawSettingsName) const
- const Microsoft::WRL::ComPtr< ID3D12RootSignature > & **GetRootSignature** (const std::wstring &drawSettingsName) const
- const D3D\_PRIMITIVE\_TOPOLOGY & **GetPrimitive** (const std::wstring &drawSettingsName) const
- FAShapes::DrawArguments & **GetDrawArguments** (const std::wstring &drawSettingsName, unsigned int index)
- const FAShapes::DrawArguments & **GetDrawArguments** (const std::wstring &drawSettingsName, unsigned int index) const
- [FACamera::Camera](#) & **GetCamera** ()
- const [FACamera::Camera](#) & **GetCamera** () const
- [FARender::Text](#) & **GetText** (std::wstring textName)
- const [FARender::Text](#) & **GetText** (std::wstring textName) const
- void **LoadShader** (const std::wstring &filename, const std::wstring &name)
- void **RemoveShader** (const std::wstring &shaderName)
- void **StoreInputElementDescriptions** (const std::wstring &name, const std::vector< D3D12\_INPUT\_ELEMENT\_DESC > &inputElementLayout)
- void **StoreInputElementDescriptions** (const std::wstring &name, const D3D12\_INPUT\_ELEMENT\_DESC \*inputElementLayout, UINT numElements)
- void **RemoveInputElementDescription** (const std::wstring &name)
- void **CreateRasterizationState** (D3D12\_FILL\_MODE fillMode, BOOL enableMultisample, const std::wstring &name)
- void **RemoveRasterizationState** (const std::wstring &name)
- void **CreatePSO** (const std::wstring &drawSettingsName, const std::wstring &rStateName, const std::wstring &vsName, const std::wstring &psName, const std::wstring &inputLayoutName, const D3D12\_PRIMITIVE\_TOPOLOGY\_TYPE &primitiveType, UINT sampleCount)
- void **CreateRootSignature** (const std::wstring &drawSettingsName)
- void **CreateVertexBuffer** ()
- void **CreateIndexBuffer** ()
 

*Creates an index buffer with the specified name and stores all of the added indices. Also creates a view to the index buffer.*

*Execute commands and flush the command queue after calling createVertexBuffer() and createIndexBuffer().*
- void **CreateCBVHeap** (UINT numDescriptors, UINT shaderRegister)
 

*Creates the CBV heap.*
- void **CreateConstantBuffer** (UINT numBytes)
 

*Creates a constant buffer for each frame.*
- void **CreateConstantBufferView** (UINT index, UINT numBytes)
 

*Creates a constant buffer view for each frame and stores it in the CBV heap.*
- void **SetPSO** (const std::wstring &drawSettingsName, const Microsoft::WRL::ComPtr< ID3D12PipelineState > &pso)
 

*Sets the PSO in the specified [DrawSettings](#) structure to the specified pso. If the specified [DrawSettings](#) structure does not exist an out\_of\_range exception is thrown.*
- void **SetRootSignature** (const std::wstring &drawSettingsName, const Microsoft::WRL::ComPtr< ID3D12RootSignature > &rootSignature)

- Sets the root signature in the specified [DrawSettings](#) structure to the specified root signature. If the specified [DrawSettings](#) structure does not exist an `out_of_range` exception is thrown.
- void [SetPrimitive](#) (const std::wstring &drawSettingsName, const D3D\_PRIMITIVE\_TOPOLOGY &primitive)  
*Sets the Primitive in the specified [DrawSettings](#) structure to the specified primitive. If the specified [DrawSettings](#) structure does not exist an `out_of_range` exception is thrown.*
  - void [AddDrawArgument](#) (const std::wstring &drawSettingsName, const FAShapes::DrawArguments &draw←Arg)  
*Adds the specified draw argument structure to the DrawArguments vector of the specified [DrawSettings](#) structure. If the specified [DrawSettings](#) structure does not exist an `out_of_range` exception is thrown.*
  - void [AddDrawArgument](#) (const std::wstring &drawSettingsName, unsigned int indexCount, unsigned int locationOfFirstIndex, int indexOfFirstVertex, int indexOfConstantData)  
*Adds the specified draw arguments to the DrawArguments vector of the specified [DrawSettings](#) structure. If the specified [DrawSettings](#) structure does not exist an `out_of_range` exception is thrown.*
  - void [RemoveDrawArgument](#) (const std::wstring &drawSettingsName, unsigned int index)  
*Removes the draw argument in the specified [DrawSettings](#) structure at the specified index. If the [DrawSettings](#) does not exist or if the index is out of bounds an `out_of_range` exception is thrown.*
  - void [CreateDrawSettings](#) (const std::wstring &drawSettingsName)  
*Creates a [DrawSettings](#) structure with the specified name.*
  - void [RemoveDrawSettings](#) (const std::wstring &drawSettingsName)  
*Removes the specified [DrawSettings](#) structure. If the [DrawSettings](#) structure does not exist an `out_of_range` exception is thrown.*
  - void [CreateText](#) (const std::wstring &textName, FAMath::Vector4D textLocation, const std::wstring &text←String, float textSize, const [FAColor::Color](#) textColor)  
*Creates a [Text](#) object with the specified properties and stores it with the specified name. 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.*
  - void [RemoveText](#) (const std::wstring &textName)  
*Removes the specified text object with the specified name. If the [Text](#) object does not exist an `out_of_range` exception is thrown.*
  - void [AddVertices](#) (const std::vector< FAShapes::Vertex > &vertices)  
*Adds the specified vertices to the vertex list.*
  - void [AddVertices](#) (const FAShapes::Vertex \*vertices, unsigned int numVertices)  
*Adds the specified vertices to the vertex list.*
  - void [AddIndices](#) (const std::vector< unsigned int > &indices)  
*Adds the specified vertices to the index list.*
  - void [AddIndices](#) (const unsigned int \*indices, unsigned int numIndices)  
*Adds the specified vertices to the index list.*
  - void [BeforeDrawObjects](#) ()  
*Puts all of the commands needed in the command list before drawing the objects of the scene. Call before calling the first drawObjects function.*
  - void [DrawObjects](#) (const std::wstring &drawSettingsName)  
*Draws all of the objects that use the same PSO, root signature and primitive. Call in between a beforeDrawObjects function and a afterDrawObjects function.*
  - void [AfterDrawObjects](#) (bool renderText)  
*Transitions the render target buffer to the correct state and excutes all the beforeDrawObjects and drawObjects commands. Pass in true if you are going to render text, false otherwise. Call after calling all the DrawObjects functions.*
  - void [BeforeDrawText](#) ()  
*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 std::wstring &textName)  
*Draws the specified [Text](#) object. Call in between a BeforeDrawText function and a AfterDrawText function.*

- void [AfterDrawText](#) ()  
*Transitions the render target buffer and executes all of the text drawing commands. Call after calling all the RenderText functions.*
- void [AfterDraw](#) ()  
*Presents and signals (puts a fence command in the command queue). Call after drawing all your objects and text.*
- void [ExecuteAndFlush](#) ()  
*Executes the commands to fill the vertex and index buffer with data and flushes the queue.*
- void [NextFrame](#) ()  
*Moves to next frame and waits for the GPU to finish executing the next frame's commands.*
- void [Resize](#) (unsigned int width, unsigned int height, HWND windowHandle)  
*Resizes the [DeviceResources](#) resources when the window gets resized.*
- void [CopyData](#) (UINT index, UINT byteSize, const void \*data, UINT64 numOfBytes)  
*Copies the specified data into the constant buffer.*
- bool [IsMSAAEnabled](#) () const  
*Returns true if MSAA is enabled, false otherwise.*
- void [DisableMSAA](#) (unsigned int width, unsigned int height, HWND windowHandle)  
*Disables MSAA.*
- void [EnableMSAA](#) (unsigned int width, unsigned int height, HWND windowHandle)  
*Enables MSAA.*

### 5.11.1 Detailed Description

This class is used to render a scene using Direct3D 12 API.

### 5.11.2 Member Function Documentation

#### 5.11.2.1 AddDrawArgument() [1/2]

```
void FARender::RenderScene::AddDrawArgument (
    const std::wstring & drawSettingsName,
    const FAShapes::DrawArguments & drawArg )
```

Adds the specified draw argument structure to the DrawArguments vector of the specified [DrawSettings](#) structure. If the specified [DrawSettings](#) structure does not exist an out\_of\_range exception is thrown.

#### 5.11.2.2 AddDrawArgument() [2/2]

```
void FARender::RenderScene::AddDrawArgument (
    const std::wstring & drawSettingsName,
    unsigned int indexCount,
    unsigned int locationOfFirstIndex,
    int indexOfFirstVertex,
    int indexOfConstantData )
```

Adds the specified draw arguments to the DrawArguments vector of the specified [DrawSettings](#) structure. If the specified [DrawSettings](#) structure does not exist an out\_of\_range exception is thrown.

#### 5.11.2.3 AddIndices() [1/2]

```
void FARender::RenderScene::AddIndices (
    const std::vector< unsigned int > & indices )
```

Adds the specified vertices to the index list.

#### 5.11.2.4 AddIndices() [2/2]

```
void FARender::RenderScene::AddIndices (
    const unsigned int * indices,
    unsigned int numIndices )
```

Adds the specified vertices to the index list.

#### 5.11.2.5 AddVertices() [1/2]

```
void FARender::RenderScene::AddVertices (
    const FAShapes::Vertex * vertices,
    unsigned int numVertices )
```

Adds the specified vertices to the vertex list.

#### 5.11.2.6 AddVertices() [2/2]

```
void FARender::RenderScene::AddVertices (
    const std::vector< FAShapes::Vertex > & vertices )
```

Adds the specified vertices to the vertex list.

#### 5.11.2.7 AfterDraw()

```
void FARender::RenderScene::AfterDraw ( )
```

Presents and signals (puts a fence command in the command queue). Call after drawing all your objects and text.

### 5.11.2.8 AfterDrawObjects()

```
void FARender::RenderScene::AfterDrawObjects (
    bool renderText )
```

Transitions the render target buffer to the correct state and excutes all the beforeDrawObjects and drawObjects commands. Pass in true if you are going to render text, false otherwise. Call after calling all the DrawObjects functions.

### 5.11.2.9 AfterDrawText()

```
void FARender::RenderScene::AfterDrawText ( )
```

Transitions the render target buffer and executes all of the text drawing commands. Call after calling all the Render↵Text functions.

### 5.11.2.10 BeforeDrawObjects()

```
void FARender::RenderScene::BeforeDrawObjects ( )
```

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

### 5.11.2.11 BeforeDrawText()

```
void FARender::RenderScene::BeforeDrawText ( )
```

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

### 5.11.2.12 CopyData()

```
void FARender::RenderScene::CopyData (
    UINT index,
    UINT byteSize,
    const void * data,
    UINT64 numOfBytes )
```

Copies the specified data into the constant buffer.

#### 5.11.2.13 CreateCBVHeap()

```
void FARender::RenderScene::CreateCBVHeap (
    UINT numDescriptors,
    UINT shaderRegister )
```

Creates the CBV heap.

#### 5.11.2.14 CreateConstantBuffer()

```
void FARender::RenderScene::CreateConstantBuffer (
    UINT numBytes )
```

Creates a constant buffer for each frame.

#### 5.11.2.15 CreateConstantBufferView()

```
void FARender::RenderScene::CreateConstantBufferView (
    UINT index,
    UINT numBytes )
```

Creates a constant buffer view for each frame and stores it in the CBV heap.

#### 5.11.2.16 CreateDrawSettings()

```
void FARender::RenderScene::CreateDrawSettings (
    const std::wstring & drawSettingsName )
```

Creates a [DrawSettings](#) structure with the specified name.

#### 5.11.2.17 CreateIndexBuffer()

```
void FARender::RenderScene::CreateIndexBuffer ( )
```

Creates an index buffer with the specified name and stores all of the added indices. Also creates a view to the index buffer.

Execute commands and flush the command queue after calling `createVertexBuffer()` and `createIndexBuffer()`.

### 5.11.2.18 CreateText()

```
void FARender::RenderScene::CreateText (
    const std::wstring & textName,
    FAMath::Vector4D textLocation,
    const std::wstring & textString,
    float textSize,
    const FColor::Color textColor )
```

Creates a [Text](#) object with the specified properties and stores it with the specified name. 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.

### 5.11.2.19 DisableMSAA()

```
void FARender::RenderScene::DisableMSAA (
    unsigned int width,
    unsigned int height,
    HWND windowHandle )
```

Disables MSAA.

### 5.11.2.20 DrawObjects()

```
void FARender::RenderScene::DrawObjects (
    const std::wstring & drawSettingsName )
```

Draws all of the objects that use the same PSO, root signature and primitive. Call in between a `beforeDrawObjects` function and a `afterDrawObjects` function.

.

Ex.

```
beforeDrawObjects()
drawObjects()
drawObjects()
afterDrawObjects()
```

Throws an `out_of_range` exception if the specified [DrawSettings](#) structure does not exist.

### 5.11.2.21 EnableMSAA()

```
void FARender::RenderScene::EnableMSAA (
    unsigned int width,
    unsigned int height,
    HWND windowHandle )
```

Enables MSAA.

#### 5.11.2.22 ExecuteAndFlush()

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

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

#### 5.11.2.23 IsMSAAEnabled()

```
bool FARender::RenderScene::IsMSAAEnabled ( ) const
```

Returns true if MSAA is enabled, false otherwise.

#### 5.11.2.24 NextFrame()

```
void FARender::RenderScene::NextFrame ( )
```

Moves to next frame and waits for the GPU to finish executing the next frame's commands.

#### 5.11.2.25 RemoveDrawArgument()

```
void FARender::RenderScene::RemoveDrawArgument (
    const std::wstring & drawSettingsName,
    unsigned int index )
```

Removes the draw argument in the specified [DrawSettings](#) structure at the specified index. If the [DrawSettings](#) does not exist or if the index is out of bounds an `out_of_range` exception is thrown.

#### 5.11.2.26 RemoveDrawSettings()

```
void FARender::RenderScene::RemoveDrawSettings (
    const std::wstring & drawSettingsName )
```

Removes the specified [DrawSettings](#) structure. If the [DrawSettings](#) structure does not exist an `out_of_range` exception is thrown.



### 5.11.2.27 RemoveText()

```
void FARender::RenderScene::RemoveText (
    const std::wstring & textName )
```

Removes the specified text object with the specified name. If the [Text](#) object does not exist an `out_of_range` exception is thrown.

### 5.11.2.28 RenderText()

```
void FARender::RenderScene::RenderText (
    const std::wstring & textName )
```

Draws the specified [Text](#) object. Call in between a `BeforeDrawText` function and a `AfterDrawText` function.

.

Ex.

```
beforeDrawText()
```

```
drawText()
```

```
drawText()
```

```
afterDrawText()
```

Throws an `out_of_range` exception if the specified [Text](#) object does not exist.

### 5.11.2.29 Resize()

```
void FARender::RenderScene::Resize (
    unsigned int width,
    unsigned int height,
    HWND windowHandle )
```

Resizes the [DeviceResources](#) resources when the window gets resized.

### 5.11.2.30 SetPrimitive()

```
void FARender::RenderScene::SetPrimitive (
    const std::wstring & drawSettingsName,
    const D3D_PRIMITIVE_TOPOLOGY & primitive )
```

Sets the Primitive in the specified [DrawSettings](#) structure to the specified primitive. If the specified [DrawSettings](#) structure does not exist an `out_of_range` exception is thrown.

### 5.11.2.31 SetPSO()

```
void FAREnder::RenderScene::SetPSO (
    const std::wstring & drawSettingsName,
    const Microsoft::WRL::ComPtr< ID3D12PipelineState > & pso )
```

Sets the PSO in the specified [DrawSettings](#) structure to the specified pso. If the specified [DrawSettings](#) structure does not exist an out\_of\_range exception is thrown.

### 5.11.2.32 SetRootSignature()

```
void FAREnder::RenderScene::SetRootSignature (
    const std::wstring & drawSettingsName,
    const Microsoft::WRL::ComPtr< ID3D12RootSignature > & rootSignature )
```

Sets the root signature in the specified [DrawSettings](#) structure to the specified root signature. If the specified [DrawSettings](#) structure does not exist an out\_of\_range exception is thrown.

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

- [FARenderScene.h](#)

## 5.12 FAREnder::RenderTargetBuffer Class Reference

A wrapper for render target buffer resources. Uses DirectD 12 API.

```
#include "FABuffer.h"
```

### Public Member Functions

- [RenderTargetBuffer](#) (DXGI\_FORMAT format=DXGI\_FORMAT\_R8G8B8A8\_UNORM)  
*Default Constructor.*
- 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 indexWhereToStoreView, unsigned int rtvSize, unsigned int width, unsigned int height, unsigned int sampleCount=1)  
*Creates the render target buffer and view.*
- void [ResetBuffer](#) ()  
*Resest the render target buffer.*
- void [ClearRenderTargetBuffer](#) (const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &rtvHeap, unsigned int indexWhereToStoreView, unsigned int rtvSize, const float \*clearValue)  
*Clears the render target buffer with the specified clear value.*

### 5.12.1 Detailed Description

A wrapper for render target buffer resources. Uses DirectD 12 API.

### 5.12.2 Constructor & Destructor Documentation

#### 5.12.2.1 RenderTargetBuffer()

```
FARender::RenderTargetBuffer::RenderTargetBuffer (
    DXGI_FORMAT format = DXGI_FORMAT_R8G8B8A8_UNORM )
```

Default Constructor.

### 5.12.3 Member Function Documentation

#### 5.12.3.1 ClearRenderTargetBuffer()

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

Clears the render target buffer with the specified clear value.

#### 5.12.3.2 CreateRenderTargetBufferAndView()

```
void FARender::RenderTargetBuffer::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,
    unsigned int sampleCount = 1 )
```

Creates the render target buffer and view.

### 5.12.3.3 GetRenderTargetBuffer() [1/2]

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

Returns a reference to the render target buffer.

### 5.12.3.4 GetRenderTargetBuffer() [2/2]

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

Returns a constant reference to the render target buffer.

### 5.12.3.5 GetRenderTargetFormat()

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

Returns the format of the render target buffer.

### 5.12.3.6 ResetBuffer()

```
void FARender::RenderTargetBuffer::ResetBuffer ( )
```

Resest the render target buffer.

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

- [FABuffer.h](#)

## 5.13 FARender::SwapChain Class Reference

A wrapper for swap chain resources. Uses DirectD 12 API and DXGI API.

```
#include "FASwapChain.h"
```

## Public Member Functions

- [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)  
*Constructor. Creates a swap chain.*
- const [RenderTargetBuffer](#) \* [GetRenderTargetBuffers](#) () const  
*Returns a constant pointer to the render target buffers.*
- 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.*
- void [ResetBuffers](#) ()  
*The render target buffers no longer reference the swap chain buffers after this function is executed.*
- void [ResizeSwapChain](#) (unsigned width, unsigned height)  
*Resizes the swap chain.*
- void [CreateRenderTargetBuffersAndViews](#) (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &rtvHeap, unsigned int indexWhereToStoreFirstView, unsigned int rtvSize)  
*Creates the render target buffers and views to them.*
- void [CreateDepthStencilBufferAndView](#) (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &dsvHeap, unsigned int index, unsigned int dsvSize, unsigned int width, unsigned int height)  
*Creates the swap chains depth stencil buffer and view to it.*
- void [ClearCurrentBackBuffer](#) (const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &rtvHeap, unsigned int indexOfFirstView, unsigned int rtvSize, const float \*backBufferClearValue)  
*Clears the current render target buffer.*
- void [ClearDepthStencilBuffer](#) (const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &dsvHeap, unsigned int indexOfView, unsigned int dsvSize, float clearValue)  
*Clears the swap chains depth stencil buffer with the specified clear value.*
- void [Transition](#) (const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, D3D12\_RESOURCE\_STATES before, D3D12\_RESOURCE\_STATES after)  
*Transitions the current render target buffer from the specified before state to the specified after state.*
- void [Present](#) ()  
*Swaps the front and back buffers.*

### 5.13.1 Detailed Description

A wrapper for swap chain resources. Uses DirectD 12 API and DXGI API.

### 5.13.2 Constructor & Destructor Documentation

### 5.13.2.1 SwapChain()

```
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 )
```

Constructor. Creates a swap chain.

## 5.13.3 Member Function Documentation

### 5.13.3.1 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.

### 5.13.3.2 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.

### 5.13.3.3 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.

#### 5.13.3.4 CreateRenderTargetBuffersAndViews()

```
void FARender::SwapChain::CreateRenderTargetBuffersAndViews (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
    const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & rtvHeap,
    unsigned int indexOfWhereToStoreFirstView,
    unsigned int rtvSize )
```

Creates the render target buffers and views to them.

#### 5.13.3.5 GetBackBufferFormat()

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

Returns the format of the swap chain.

#### 5.13.3.6 GetCurrentBackBuffer()

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

Returns a constant reference to the current render target buffer.

#### 5.13.3.7 GetCurrentBackBufferIndex()

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

Returns the current back buffer index.

#### 5.13.3.8 GetDepthStencilFormat()

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

Returns the format of the depth stencil buffer.

#### 5.13.3.9 GetNumRenderTargetBuffers()

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

Returns the number of swap chain buffers.

#### 5.13.3.10 GetRenderTargetBuffers()

```
const RenderTargetBuffer * FARender::SwapChain::GetRenderTargetBuffers ( ) const
```

Returns a constant pointer to the render target buffers.

#### 5.13.3.11 Present()

```
void FARender::SwapChain::Present ( )
```

Swaps the front and back buffers.

#### 5.13.3.12 ResetBuffers()

```
void FARender::SwapChain::ResetBuffers ( )
```

The render target buffers no longer reference the swap chain buffers after this function is executed.

#### 5.13.3.13 ResizeSwapChain()

```
void FARender::SwapChain::ResizeSwapChain (
    unsigned width,
    unsigned height )
```

Resizes the swap chain.

#### 5.13.3.14 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.

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

- FASwapChain.h



## 5.14 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)  
*Overloaded Constructor. Initializes the format of the text.  
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.*
- 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 size.*
- void [SetTextColor](#) (const [FAColor::Color](#) &textColor)  
*Changes the text color to the specified color.*
- void [SetTextString](#) (const std::wstring &textString)  
*Changes the text string to the specified string.*
- void [SetTextLocation](#) (const FAMath::Vector4D &textLocation)  
*Changes the text location to the specified location.*

### 5.14.1 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.

### 5.14.2 Constructor & Destructor Documentation

#### 5.14.2.1 Text()

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

Overloaded Constructor. Initializes the format of the text.

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.

### 5.14.3 Member Function Documentation

#### 5.14.3.1 GetTextColor()

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

Returns a constant reference to the text color.

#### 5.14.3.2 GetTextLocation()

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

Returns a constant reference to the text location.

#### 5.14.3.3 GetTextSize()

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

Returns the text size.

#### 5.14.3.4 GetTextString()

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

Returns a constant reference to the text string.

#### 5.14.3.5 SetTextColor()

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

Changes the text color to the specified color.

#### 5.14.3.6 SetTextLocation()

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

Changes the text location to the specified location.

#### 5.14.3.7 SetTextSize()

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

Changes the text size to the specified size.

#### 5.14.3.8 SetTextString()

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

Changes the text string to the specified string.

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

- [FAText.h](#)

## 5.15 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)  
*Constructor. 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 [RenderTargetBuffer](#) \*renderTargetBuffers, HWND windowHandle)  
*Resizes the buffers.*
- void [BeforeRenderText](#) (unsigned int currentBackBuffer)  
*Prepares to render text.*
- void [AfterRenderText](#) (unsigned int currentBackBuffer)  
*Executes text commands.*

### 5.15.1 Detailed Description

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

### 5.15.2 Constructor & Destructor Documentation

#### 5.15.2.1 TextResources()

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

Constructor. Initializes the text resources.

### 5.15.3 Member Function Documentation

#### 5.15.3.1 AfterRenderText()

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

Executes text commands.

#### 5.15.3.2 BeforeRenderText()

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

Prepares to render text.

#### 5.15.3.3 GetDirect2DDeviceContext()

```
const Microsoft::WRL::ComPtr< ID2D1DeviceContext > & FAREnder::TextResources::GetDirect2↔
DDeviceContext ( ) const
```

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

#### 5.15.3.4 GetDirectWriteFactory()

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

Returns a constant reference to the direct direct write factory.

#### 5.15.3.5 ResetBuffers()

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

Resets the text buffers.

#### 5.15.3.6 ResizeBuffers()

```
void FARender::TextResources::ResizeBuffers (
    const RenderTargetBuffer * renderTargetBuffers,
    HWND windowHandle )
```

Resizes the buffers.

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

- FATextResources.h

## 5.16 FATime::Time Class Reference

### Public Member Functions

- [Time](#) ()  
*Default Constructor. 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.*

## 5.16.1 Constructor & Destructor Documentation

### 5.16.1.1 Time()

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

Default Constructor. Gets and stores the seconds per count.

## 5.16.2 Member Function Documentation

### 5.16.2.1 DeltaTime()

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

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

### 5.16.2.2 Reset()

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

Resets all time variables.

### 5.16.2.3 Start()

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

Starts the timer.

### 5.16.2.4 Stop()

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

Stops the timer.

### 5.16.2.5 Tick()

```
void FTime::Time::Tick ( )
```

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

### 5.16.2.6 TotalTime()

```
float FTime::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:

- [FTime.h](#)

## 5.17 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 "FTime.h"
```

### 5.17.1 Detailed Description

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

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

- [FTime.h](#)

## 5.18 FRender::VertexBuffer Class Reference

This class stores vertices in a Direct3D 12 default buffer.

```
#include "FABuffer.h"
```

### Public Member Functions

- **VertexBuffer** (const [VertexBuffer](#) &)=delete
- **VertexBuffer & operator=** (const [VertexBuffer](#) &)=delete
- void [CreateVertexBuffer](#) (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const void \*data, UINT numBytes)  
*Creates the vertex buffer and stores all of the specified vertices in the vertex buffer.*
- void [CreateVertexBufferView](#) (UINT numBytes, UINT stride)  
*Creates the vertex buffer view and stores it.*
- const D3D12\_VERTEX\_BUFFER\_VIEW & [GetVertexBufferView](#) ()  
*Returns a constant reference to the vertex buffer view.*

### 5.18.1 Detailed Description

This class stores vertices in a Direct3D 12 default buffer.

### 5.18.2 Member Function Documentation

#### 5.18.2.1 CreateVertexBuffer()

```
void FARender::VertexBuffer::CreateVertexBuffer (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
    const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & commandList,
    const void * data,
    UINT numBytes )
```

Creates the vertex buffer and stores all of the specified vertices in the vertex buffer.

#### 5.18.2.2 CreateVertexBufferView()

```
void FARender::VertexBuffer::CreateVertexBufferView (
    UINT numBytes,
    UINT stride )
```

Creates the vertex buffer view and stores it.

#### 5.18.2.3 GetVertexBufferView()

```
const D3D12_VERTEX_BUFFER_VIEW & FARender::VertexBuffer::GetVertexBufferView ( )
```

Returns a constant reference to the vertex buffer view.

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

- [FABuffer.h](#)

## 5.19 FAWindow::Window Class Reference

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

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



## Public Member Functions

- [Window](#) (const HINSTANCE &hInstance, const std::wstring &windowClassName, const std::wstring &windowName, WNDPROC winProcFunction, unsigned int width, unsigned int height, void \*additionalData=nullptr)  
*Creates and displays a window. Registers a default window class with the OS with the specified instance, class name and window procedure.*
- [Window](#) (const HINSTANCE &hInstance, const WNDCLASSEX &windowClass, const std::wstring &windowName, unsigned int width, unsigned int height, void \*additionalData=nullptr)  
*Creates and displays a window. Registers the specified window class with the OS.*
- 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.*
- 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 to the specified height.*

### 5.19.1 Detailed Description

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

### 5.19.2 Constructor & Destructor Documentation

#### 5.19.2.1 Window() [1/2]

```
FAWindow::Window::Window (
    const HINSTANCE & hInstance,
    const std::wstring & windowClassName,
    const std::wstring & windowName,
    WNDPROC winProcFunction,
    unsigned int width,
    unsigned int height,
    void * additionalData = nullptr )
```

Creates and displays a window. Registers a default window class with the OS with the specified instance, class name and window procedure.

### 5.19.2.2 Window() [2/2]

```
FAWindow::Window::Window (
    const HINSTANCE & hInstance,
    const WNDCLASSEX & windowClass,
    const std::wstring & windowName,
    unsigned int width,
    unsigned int height,
    void * additionalData = nullptr )
```

Creates and displays a window. Registers the specified window class with the OS.

## 5.19.3 Member Function Documentation

### 5.19.3.1 GetHeight()

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

Returns the height of the window.

### 5.19.3.2 GetWidth()

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

Returns the width of the window.

### 5.19.3.3 GetWindowHandle()

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

Returns the window handle.

### 5.19.3.4 SetHeight()

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

Sets the height of the window o the specified height.

### 5.19.3.5 SetWidth()

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

Sets the width of the window to the specified width.

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

- [FAWindow.h](#)

## Chapter 6

# File Documentation

### 6.1 Direct3DLink.h

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

### 6.2 FABuffer.h File Reference

File has classes `VertexBuffer`, `IndexBuffer` and `ConstantBuffer` under namespace `FARender`.

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

#### Classes

- class `FARender::RenderTargetBuffer`  
*A wrapper for render target buffer resources. Uses DirectD 12 API.*
- class `FARender::DepthStencilBuffer`  
*A wrapper for depth stencil buffer resources. Uses DirectD 12 API.*
- class `FARender::VertexBuffer`  
*This class stores vertices in a Direct3D 12 default buffer.*
- class `FARender::IndexBuffer`  
*This class stores indices in a Direct3D 12 default buffer.*
- class `FARender::ConstantBuffer`  
*This class stores constant data in a Direct3D 12 upload buffer.*

#### Namespaces

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

## 6.2.1 Detailed Description

File has classes VertexBuffer, IndexBuffer and ConstantBuffer under namespace [FARender](#).

## 6.3 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      class RenderTargetBuffer
9      {
10     public:
11         RenderTargetBuffer(DXGI_FORMAT format = DXGI_FORMAT_R8G8B8A8_UNORM);
12
13         DXGI_FORMAT GetRenderTargetFormat() const;
14
15         Microsoft::WRL::ComPtr<ID3D12Resource>& GetRenderTargetBuffer();
16
17         const Microsoft::WRL::ComPtr<ID3D12Resource>& GetRenderTargetBuffer() const;
18
19         void CreateRenderTargetBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
20             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int
21             indexWhereToStoreView, unsigned int rtvSize,
22             unsigned int width, unsigned int height, unsigned int sampleCount = 1);
23
24         void ResetBuffer();
25
26         void ClearRenderTargetBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
27             commandList,
28             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int indexWhereToStoreView,
29             unsigned int rtvSize,
30             const float* clearValue);
31
32     private:
33         Microsoft::WRL::ComPtr<ID3D12Resource> mRenderTargetBuffer;
34         DXGI_FORMAT mRenderTargetFormat;
35     };
36
37     class DepthStencilBuffer
38     {
39     public:
40         DepthStencilBuffer(DXGI_FORMAT format = DXGI_FORMAT_D24_UNORM_S8_UINT);
41
42         DXGI_FORMAT GetDepthStencilFormat() const;
43
44         void CreateDepthStencilBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
45             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int
46             indexWhereToStoreView, unsigned int dsvSize,
47             unsigned int width, unsigned int height, unsigned int sampleCount = 1);
48
49         void ResetBuffer();
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         Microsoft::WRL::ComPtr<ID3D12Resource> mDepthStencilBuffer;
59         DXGI_FORMAT mDepthStencilFormat;
60     };
61
62     class VertexBuffer
63     {
64     public:
65         VertexBuffer() = default;
66         VertexBuffer(const VertexBuffer&) = delete;
67         VertexBuffer& operator=(const VertexBuffer&) = delete;
68
69         void CreateVertexBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,

```

```

110         const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const void* data, UINT
numBytes);
111
112     void CreateVertexBufferView(UINT numBytes, UINT stride);
113
114     const D3D12_VERTEX_BUFFER_VIEW& GetVertexBufferView();
115
116 private:
117     Microsoft::WRL::ComPtr<ID3D12Resource> mVertexDefaultBuffer;
118     Microsoft::WRL::ComPtr<ID3D12Resource> mVertexUploadBuffer;
119     D3D12_VERTEX_BUFFER_VIEW mVertexBufferView{};
120 };
121
122 class IndexBuffer
123 {
124 public:
125     IndexBuffer() = default;
126     IndexBuffer(const IndexBuffer&) = delete;
127     IndexBuffer& operator=(const IndexBuffer&) = delete;
128
129     const D3D12_INDEX_BUFFER_VIEW& GetIndexBufferView();
130
131     void CreateIndexBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const void* data, UINT
numBytes);
132
133     void CreateIndexBufferView(UINT numBytes, DXGI_FORMAT format);
134
135 private:
136     Microsoft::WRL::ComPtr<ID3D12Resource> mIndexDefaultBuffer;
137     Microsoft::WRL::ComPtr<ID3D12Resource> mIndexUploadBuffer;
138     D3D12_INDEX_BUFFER_VIEW mIndexBufferView{};
139 };
140
141 class ConstantBuffer
142 {
143 public:
144     ConstantBuffer() = default;
145
146     ConstantBuffer(const ConstantBuffer&) = delete;
147     ConstantBuffer& operator=(const ConstantBuffer&) = delete;
148
149     ~ConstantBuffer();
150
151     void CreateConstantBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device, const UINT&
numOfBytes);
152
153     void CreateConstantBufferView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& cbvHeap, UINT cbvSize, UINT
cBufferIndex,
154     UINT cbvHeapIndex, UINT numBytes);
155
156     void CopyData(UINT index, UINT byteSize, const void* data, UINT64 numOfBytes);
157
158 private:
159     Microsoft::WRL::ComPtr<ID3D12Resource> mConstantBuffer;
160     BYTE* mMappedData{ nullptr };
161 };
162 }

```

## 6.4 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.*

*It keeps track of the viewing frustum of the camera so that the projection matrix can be obtained.*

.

## Namespaces

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

## Typedefs

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

### 6.4.1 Detailed Description

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

### 6.4.2 Typedef Documentation

#### 6.4.2.1 vec2

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

FACAMERA\_H FILE

## 6.5 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 {
30     class Camera
31     {
32     public:
44         Camera(vec3 cameraPosition = vec3(0.0f, 0.0f, 0.0f),
45             vec3 x = vec3(1.0f, 0.0f, 0.0f), vec3 y = vec3(0.0f, 1.0f, 0.0f), vec3 z = vec3(0.0f, 0.0f,
1.0f),
46             float znear = 1.0f, float zfar = 100.f, float aspectRatio = 1.0f, float vFov = 45.0f,
47             float cameraVelocity = 10.0f, float angularVelocity = 0.25f);
48
51         const vec3& GetCameraPosition() const;
52
55         const vec3& GetX() const;
56
59         const vec3& GetY() const;
60
63         const vec3& GetZ() const;
64
```

```

67     const mat4& GetViewTransformationMatrix() const;
68
71     float GetCameraVelocity() const;
72
75     float GetAngularVelocity() const;
76
79     void LookAt(vec3 cameraPosition, vec3 target, vec3 up);
80
83     float GetZNear() const;
84
87     float GetZFar() const;
88
91     float GetVerticalFov() const;
92
95     float GetAspectRatio() const;
96
99     void SetCameraPosition(const vec3& position);
100
103     void SetX(const vec3& x);
104
107     void SetY(const vec3& y);
108
111     void SetZ(const vec3& z);
112
115     void SetCameraVelocity(float velocity);
116
119     void SetAngularVelocity(float velocity);
120
123     void SetZNear(float znear);
124
127     void SetZFar(float zfar);
128
131     void SetVerticalFov(float fov);
132
135     void SetAspectRatio(float ar);
136
139     const mat4& GetPerspectiveProjectionMatrix() const;
140
143     const mat4& GetViewPerspectiveProjectionMatrix() const;
144
147     void UpdateViewMatrix();
148
151     void UpdatePerspectiveProjectionMatrix();
152
156     void UpdateViewPerspectiveProjectionMatrix();
157
160     void Left(float dt);
161
164     void Right(float dt);
165
168     void Forward(float dt);
169
172     void Backward(float dt);
173
176     void Up(float dt);
177
180     void Down(float dt);
181
184     void RotateCameraLeftRight(float xDiff);
185
188     void RotateCameraUpDown(float yDiff);
189
195     void KeyboardInput(float dt);
196
199     void MouseInput();
200
201 private:
202     //camera position in world coordinates
203     vec3 mCameraPosition;
204
205     //z-axis of the camera coordinate system
206     vec3 mN;
207
208     //y-axis of the camera coordinate system
209     vec3 mV;
210
211     //x-axis of the camera coordinate system
212     vec3 mU;
213
214     //stores the world to camera transform
215     mat4 mViewMatrix;
216
217     //frustrum properties
218     float mNear;
219     float mFar;
220     float mVerticalFov;
221     float mAspectRatio;

```

```

222         mat4 mPerspectiveProjectionMatrix;
223
224         mat4 mViewPerspectiveProjectionMatrix;
225
226         float mCameraVelocity;
227         float mAngularVelocity;
228
229         vec2 mLastMousePosition;
230     };
231 }

```

## 6.6 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.*

### Functions

- Color [FAColor::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  $1.0f$ .*
- Color [FAColor::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  $0.0f$ .*
- Color [FAColor::operator\\*](#) (const Color &c, float k)  
*Returns the result of  $c * k$ . If  $k < 0.0f$ , no multiplication happens and [Color](#)  $c$  is returned. If any of the resultant components are  $> 1.0f$ , they are set to  $1.0f$ .*
- Color [FAColor::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  $1.0f$ .*
- Color [FAColor::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  $1.0f$ .*

#### 6.6.1 Detailed Description

File has class Color under namespace FAColor.

#### 6.6.2 Function Documentation



### 6.6.2.1 operator\*() [1/3]

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

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

.

### 6.6.2.2 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  $1.0f$ .

.

### 6.6.2.3 operator\*() [3/3]

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

.

### 6.6.2.4 operator+()

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

### 6.6.2.5 operator-()

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

## 6.7 FIColor.h

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

```

1  #pragma once
2
3  #include "FAMathEngine.h"
4
5  namespace FIColor
6  {
7      class Color
8      {
9      public:
10
11          Color(float r = 0.0f, float g = 0.0f, float b = 0.0f, float a = 1.0f);
12
13          Color(const FAMath::Vector4D& color);
14
15          const FAMath::Vector4D& GetColor() const;
16
17          float GetRed() const;
18
19          float GetGreen() const;
20
21          float GetBlue() const;
22
23          float GetAlpha() const;
24
25          void SetColor(const FAMath::Vector4D& color);
26
27          void SetRed(float r);
28
29          void SetGreen(float g);
30
31          void SetBlue(float b);
32
33          void SetAlpha(float a);
34
35          Color& operator+=(const Color& c);
36
37          Color& operator-=(const Color& c);
38
39          Color& operator*=(float k);
40
41          Color& operator*=(const Color& c);
42
43      private:
44          FAMath::Vector4D mColor;
45      };
46
47      Color operator+(const Color& c1, const Color& c2);
48
49      Color operator-(const Color& c1, const Color& c2);
50
51      Color operator*(const Color& c, float k);
52
53      Color operator*(float k, const Color& c);
54
55      Color operator*(const Color& c1, const Color& c2);
56  }

```

## 6.8 FADepthStencil.h

```

1  #pragma once
2
3  #include <wrl.h>
4  #include "d3dx12.h"
5
6  namespace FARender
7  {
8      class DepthStencilBuffer
9      {
10      public:
11
12          DepthStencilBuffer(DXGI_FORMAT format = DXGI_FORMAT_D24_UNORM_S8_UINT);
13
14          DXGI_FORMAT GetDepthStencilFormat() const;
15
16          void CreateDepthStencilBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
17              const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int
18              indexWhereToStoreView, unsigned int dsvSize,
19              unsigned int width, unsigned int height);
20
21      };
22  }

```

```

28
31     void ResetBuffer();
32
33     void ClearDepthStencilBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
commandList,
34     const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int indexOfView,
35     unsigned int dsvSize,
36     float clearValue);
37
38 private:
39     Microsoft::WRL::ComPtr<ID3D12Resource> mDepthStencilBuffer;
40     DXGI_FORMAT mDepthStencilFormat;
41 };
42
43 }

```

## 6.9 FADeviceResources.h File Reference

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

```

#include <wrl.h>
#include <d3d12.h>
#include <dxgi1_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.*

### Namespaces

- namespace [FARender](#)  
*Has classes that are used for rendering objects and text through the Direct3D 12 API.*

#### 6.9.1 Detailed Description

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

## 6.10 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 FARender
11 {
12     class DeviceResources
13     {

```

```

21     public:
22         static const unsigned int NUM_OF_FRAMES{ 3 };
23
24         static DeviceResources& GetInstance(unsigned int width, unsigned int height, HWND windowHandle);
25
26         DeviceResources(const DeviceResources&) = delete;
27         DeviceResources& operator=(const DeviceResources&) = delete;
28
29         ~DeviceResources();
30
31         const Microsoft::WRL::ComPtr<ID3D12Device>& GetDevice() const;
32
33         const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& GetCommandList() const;
34
35         DXGI_FORMAT GetBackBufferFormat() const;
36
37         DXGI_FORMAT GetDepthStencilFormat() const;
38
39         UINT GetCBVSize() const;
40
41         unsigned int GetCurrentFrame() const;
42
43         const TextResources& GetTextResources() const;
44
45         bool IsMSAAEnabled() const;
46
47         void DisableMSAA(unsigned int width, unsigned int height, HWND windowHandle);
48
49         void EnableMSAA(unsigned int width, unsigned int height, HWND windowHandle);
50
51         void UpdateCurrentFrameFenceValue();
52
53         void FlushCommandQueue();
54
55         void WaitForGPU() const;
56
57         void Signal();
58
59         void Resize(int width, int height, const HWND& handle);
60
61         void RTBufferTransition(bool renderText);
62
63         void BeforeTextDraw();
64
65         void AfterTextDraw();
66
67         void Execute() const;
68
69         void Present();
70
71         /*@brief Calls the necessary functions to let the user draw their objects.
72         */
73         void Draw();
74
75         void NextFrame();
76
77     private:
78
79         DeviceResources(unsigned int width, unsigned int height, HWND windowHandle);
80
81         unsigned int mCurrentFrameIndex{ 0 };
82
83         Microsoft::WRL::ComPtr<ID3D12Device> mDirect3DDevice;
84
85         Microsoft::WRL::ComPtr<IDXGIFactory4> mDXGIFactory;
86
87         Microsoft::WRL::ComPtr<ID3D12Fence> mFence;
88         UINT64 mFenceValue{ 0 };
89         UINT64 mCurrentFrameFenceValue[NUM_OF_FRAMES];
90
91         Microsoft::WRL::ComPtr<ID3D12CommandQueue> mCommandQueue;
92         Microsoft::WRL::ComPtr<ID3D12CommandAllocator> mCommandAllocator[NUM_OF_FRAMES];
93         Microsoft::WRL::ComPtr<ID3D12CommandAllocator> mDirectCommandAllocator;
94         Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList> mCommandList;
95
96         UINT mRTVSize;
97         UINT mDSVSize;
98         UINT mCBVSize;
99
100        Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mRTVHeap;
101        Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mDSVHeap;
102
103        SwapChain mSwapChain;
104
105        bool mIsMSAAEnabled{ false };
106        MultiSampling mMultiSampling;
107

```

```

170         D3D12_VIEWPORT mViewport{};
171         D3D12_RECT mScissor{};
172
173         TextResources mTextResources;
174
175         //Call all of these functions to initialize Direct3D
176         void mEnableDebugLayer();
177         void mCreateDirect3DDevice();
178         void mCreateDXGIFactory();
179         void mCreateFence();
180         void mQueryDescriptorSizes();
181         void mCreateRTVHeap();
182         void mCreateDSVHeap();
183         void mCreateCommandObjects();
184     };
185 }

```

## 6.11 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 ErrorMsg() 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 //use when calling Direct3D or DXGI function to check if the function failed or not.
33 #ifndef ThrowIfFailed
34 #define ThrowIfFailed(x)
35 {
36     HRESULT hr = (x);
37     std::wstring filename(AnsiToWString(__FILE__));
38     if (FAILED(hr)) { throw DirectXException(hr, L"x", filename, __LINE__); }
39 }
40 #endif

```

## 6.12 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         MultiSampling() = default;
13
14         MultiSampling(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
            DXGI_FORMAT rtFormat, DXGI_FORMAT dsFormat, unsigned int sampleCount);
15     };
16 }

```

```

24
27     const Microsoft::WRL::ComPtr<ID3D12Resource>& GetRenderTargetBuffer();
28
29     DXGI_FORMAT GetRenderTargetFormat();
30
31     DXGI_FORMAT GetDepthStencilFormat();
32
33     void ResetBuffers();
34
35     void CreateRenderTargetBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
36     const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int
37     indexWhereToStoreView, unsigned int rtvSize,
38     unsigned int width, unsigned int height);
39
40     void CreateDepthStencilBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
41     const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int
42     indexWhereToStoreView, unsigned int dsvSize,
43     unsigned int width, unsigned int height);
44
45     void Transition(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList,
46     D3D12_RESOURCE_STATES before, D3D12_RESOURCE_STATES after);
47
48     void ClearRenderTargetBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
49     commandList,
50     const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int indexWhereToStoreView,
51     unsigned int rtvSize,
52     const float* clearValue);
53
54     void ClearDepthStencilBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
55     commandList,
56     const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int indexWhereToStoreView,
57     unsigned int dsvSize,
58     float clearValue);
59
60     private:
61     RenderTargetBuffer mMSAARenderTargetBuffer;
62     DepthStencilBuffer mMSAADepthStencilBuffer;
63     unsigned int mSampleCount{ 0 };
64
65     /*Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mMSAARTVDescriptorHeap;
66     Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mMSAADSVDDescriptorHeap;
67     Microsoft::WRL::ComPtr<ID3D12Resource> mMSAARenderTargetBuffer;
68     Microsoft::WRL::ComPtr<ID3D12Resource> mMSAADepthStencilBuffer;*/
69
70     };
71 }

```

## 6.13 FARenderScene.h File Reference

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

```

#include <d3dcompiler.h>
#include <unordered_map>
#include <string>
#include "FADeviceResources.h"
#include "FABuffer.h"
#include "FACamera.h"
#include "FAText.h"
#include "FAShapesUtility.h"

```

### Classes

- struct `FARender::DrawSettings`  
*Holds a array of objects that use the same PSO, root signature and primitive.*
- class `FARender::RenderScene`  
*This class is used to render a scene using Direct3D 12 API.*

## Namespaces

- namespace [FARender](#)

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

### 6.13.1 Detailed Description

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

## 6.14 FARenderScene.h

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

```

1 #pragma once
2
3 #include <d3dcompiler.h>
4 #include <unordered_map>
5 #include <string>
6 #include "FADeviceResources.h"
7 #include "FABuffer.h"
8 #include "FACamera.h"
9 #include "FAText.h"
10 #include "FAShapesUtility.h"
11
12 namespace FARender
13 {
14     struct DrawSettings
15     {
16         Microsoft::WRL::ComPtr<ID3D12PipelineState> pipelineState;
17         Microsoft::WRL::ComPtr<ID3D12RootSignature> rootSig;
18         D3D_PRIMITIVE_TOPOLOGY prim = D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST;
19         std::vector<FAShapes::DrawArguments> drawArgs;
20     };
21
22     class RenderScene
23     {
24     public:
25         RenderScene(unsigned int width, unsigned int height, HWND windowHandle);
26         RenderScene(const RenderScene&) = delete;
27         RenderScene& operator=(const RenderScene&) = delete;
28
29         /*@brief Returns a constant reference to the device resources object.
30 */
31         const DeviceResources& GetDeviceResources() const;
32
33         /*@brief Returns a constant reference to the shader with the specified name.
34 * Throws an out_of_range exception if the shader does not exist.
35 */
36         const Microsoft::WRL::ComPtr<ID3DBlob>& GetShader(const std::wstring& name) const;
37
38         /*@brief Returns a constant reference to the specified array of input element layout
39 descriptions.
40 * Throws an out_of_range exception if the array of input element layout descriptions does not exist.
41 */
42         const std::vector<D3D12_INPUT_ELEMENT_DESC>& GetInputElementLayout(const std::wstring& name)
43             const;
44
45         /*@brief Returns a constant reference to the specified rasterization description.
46 * Throws an out_of_range exception if the rasterization description does not exist.
47 */
48         const D3D12_RASTERIZER_DESC& GetRasterizationState(const std::wstring& name) const;
49
50         /*@brief Returns a constant reference to the PSO in the specified DrawSettings.
51 * Throws an out_of_range exception if the DrawSettings does not exist.
52 */
53         const Microsoft::WRL::ComPtr<ID3D12PipelineState>& GetPSO(const std::wstring& drawSettingsName)
54             const;
55
56         /*@brief Returns a constant reference to the root signature in the specified DrawSettings
57 structure.
58 * Throws an out_of_range exception if the DrawSettings does not exist.
59 */
60
61
62
63
64
65
66
67
68

```

```

69         const Microsoft::WRL::ComPtr<ID3D12RootSignature>& GetRootSignature(const std::wstring&
drawSettingsName) const;
70
71         /*@brief Returns a constant reference to the primitive in the specified DrawSettings structure.
72 * Throws an out_of_range exception if the DrawSettings does not exist.
73 */
74         const D3D_PRIMITIVE_TOPOLOGY& GetPrimitive(const std::wstring& drawSettingsName) const;
75
76         /*@brief Returns a reference to the specified DrawArguments object in the specified DrawSettings
structure.
77 * Throws an out_of_range exception if the DrawSettings does not exist or if the index is out of range.
78 */
79         FAShapes::DrawArguments& GetDrawArguments(const std::wstring& drawSettingsName, unsigned int
index);
80
81         /*@brief Returns a constant reference to the specified DrawArguments object in the specified
DrawSettings structure.
82 * Throws an out_of_range exception if the DrawSettings does not exist or if the index is out of range.
83 */
84         const FAShapes::DrawArguments& GetDrawArguments(const std::wstring& drawSettingsName, unsigned
int index) const;
85
86         /*@brief Returns a reference to the this scene's camera;
87 */
88         FACamera::Camera& GetCamera();
89
90         /*@brief Returns a constant reference to the this scene's camera;
91 */
92         const FACamera::Camera& GetCamera() const;
93
94         /*@brief Returns a reference to the specified Text object.
95 * If the Text object does not exist an out_of_range exception is thrown.
96 */
97         FARender::Text& GetText(std::wstring textName);
98
99         /*@brief Returns a constant reference to the specified Text object.
100 * If the Text object does not exist an out_of_range exception is thrown.
101 */
102         const FARender::Text& GetText(std::wstring textName) const;
103
104         /*@brief Loads a shader's bytecode and stores it with the specified name.
105 */
106         void LoadShader(const std::wstring& filename, const std::wstring& name);
107
108         /*@brief Removes the specified shader.
109 * If the specified shader does not exist an out_of_range exception is thrown.
110 */
111         void RemoveShader(const std::wstring& shaderName);
112
113         /*@brief Stores an array of input element descriptions with the specified name.
114 */
115         void StoreInputElementDescriptions(const std::wstring& name, const
std::vector<D3D12_INPUT_ELEMENT_DESC>& inputElementLayout);
116
117         /*@brief Stores an array of input element descriptions with the specified name.
118 */
119         void StoreInputElementDescriptions(const std::wstring& name, const D3D12_INPUT_ELEMENT_DESC*
inputElementLayout,
UINT numElements);
120
121         /*@brief Removes the specified input element description.
122 * If the specified input element description does not exist an out_of_range exception is thrown.
123 */
124         void RemoveInputElementDescription(const std::wstring& name);
125
126         /*@brief Creates a rasterization state description and stores it with the specified name.
127 */
128         void CreateRasterizationState(D3D12_FILL_MODE fillMode, BOOL enableMultisample, const
std::wstring& name);
129
130         /*@brief Removes the specified rasterization state.
131 * If the specified rasterization state does not exist an out_of_range exception is thrown.
132 */
133         void RemoveRasterizationState(const std::wstring& name);
134
135         /*@brief Creates a PSO and stores it in the specified DrawSettings structure.
136 * If the specified DrawSettings structure, Rasterization State, Vertex Shader, Pixel Shader or Input
Layout
137 * does not exist an out_of_range exception is thrown.
138 */
139         void CreatePSO(const std::wstring& drawSettingsName, const std::wstring& rStateName,
const std::wstring& vsName, const std::wstring& psName, const std::wstring& inputLayoutName,
const D3D12_PRIMITIVE_TOPOLOGY_TYPE& primitiveType, UINT sampleCount);
140
141         /*@brief Creates a root signature and stores it with the specified name.
142 * If the specified DrawSettings structure does not exist an out_of_range exception is thrown.
143 */
144

```



```

147     void CreateRootSignature(const std::wstring& drawSettingsName);
148
149     /*@brief Creates a vertex buffer with the specified name and stores all of the added vertices.
150 * Also creates a view to the vertex buffer.\n
151 * Execute commands and the flush command queue after calling createVertexBuffer() and
152   createIndexBuffer().
153 */
154     void CreateVertexBuffer();
155
156     void CreateIndexBuffer();
157
158     void CreateCBVHeap(UINT numDescriptors, UINT shaderRegister);
159
160     void CreateConstantBuffer(UINT numBytes);
161
162     void CreateConstantBufferView(UINT index, UINT numBytes);
163
164     void SetPSO(const std::wstring& drawSettingsName, const
165 Microsoft::WRL::ComPtr<ID3D12PipelineState>& pso);
166
167     void SetRootSignature(const std::wstring& drawSettingsName, const
168 Microsoft::WRL::ComPtr<ID3D12RootSignature>& rootSignature);
169
170     void SetPrimitive(const std::wstring& drawSettingsName, const D3D_PRIMITIVE_TOPOLOGY&
171 primitive);
172
173     void AddDrawArgument(const std::wstring& drawSettingsName, const FAShapes::DrawArguments&
174 drawArg);
175
176     void AddDrawArgument(const std::wstring& drawSettingsName,
177 unsigned int indexCount, unsigned int locationOfFirstIndex, int indexOfFirstVertex, int
178 indexOfConstantData);
179
180     void RemoveDrawArgument(const std::wstring& drawSettingsName, unsigned int index);
181
182     void CreateDrawSettings(const std::wstring& drawSettingsName);
183
184     void RemoveDrawSettings(const std::wstring& drawSettingsName);
185
186     void CreateText(const std::wstring& textName, FAMath::Vector4D textLocation, const std::wstring&
187 textString,
188 float textSize, const FAColor::Color textColor);
189
190     void RemoveText(const std::wstring& textName);
191
192     void AddVertices(const std::vector<FAShapes::Vertex>& vertices);
193
194     void AddVertices(const FAShapes::Vertex* vertices, unsigned int numVertices);
195
196     void AddIndices(const std::vector<unsigned int>& indices);
197
198     void AddIndices(const unsigned int* indices, unsigned int numIndices);
199
200     void BeforeDrawObjects();
201
202     void DrawObjects(const std::wstring& drawSettingsName);
203
204     void AfterDrawObjects(bool renderText);
205
206     void BeforeDrawText();
207
208     void RenderText(const std::wstring& textName);
209
210     void AfterDrawText();
211
212     void AfterDraw();
213
214     void ExecuteAndFlush();
215
216     void NextFrame();
217
218     void Resize(unsigned int width, unsigned int height, HWND windowHandle);
219
220     void CopyData(UINT index, UINT byteSize, const void* data, UINT64 numBytes);
221
222     bool IsMSAAEnabled() const;
223
224     void DisableMSAA(unsigned int width, unsigned int height, HWND windowHandle);
225
226     void EnableMSAA(unsigned int width, unsigned int height, HWND windowHandle);
227
228 private:
229
230     //The device resources object that all RenderScene objects share.
231     DeviceResources& mDeviceResources;
232
233     //Stores all of the shaders and input element descriptions for this scene.

```

```

327         std::unordered_map<std::wstring, Microsoft::WRL::ComPtr<ID3DBlob>> mShaders;
328         std::unordered_map< std::wstring, std::vector<D3D12_INPUT_ELEMENT_DESC>
mInputElementDescriptions;
329
330         //Stores all of the rasterization states.
331         std::unordered_map<std::wstring, D3D12_RASTERIZER_DESC> mRasterizationStates;
332
333         //Stores all of the possible draw settings that the scene uses.
334         std::unordered_map<std::wstring, DrawSettings> mSceneObjects;
335
336         //Each scene gets a CBV heap.
337         Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mCBVHeap;
338         D3D12_DESCRIPTOR_RANGE mCBVHeapDescription{};
339         D3D12_ROOT_PARAMETER mCBVHeapRootParameter;
340
341         //Stores all of the constant buffers this scene uses. We can't update a constant buffer until
the GPU
342         //is done executing all the commands that reference it, so each frame needs its own constant
buffer.
343         ConstantBuffer mConstantBuffer[DeviceResources::NUM_OF_FRAMES];
344
345         //The vertices and indices for the scene.
346         std::vector<FAShapes::Vertex> mVertexList;
347         std::vector<unsigned int> mIndexList;
348
349         //The vertex and index buffer for the scene.
350         VertexBuffer mVertexBuffer;
351         IndexBuffer mIndexBuffer;
352
353         //All of the text that is rendered with the scene.
354         std::unordered_map<std::wstring, Text> mTexts;
355
356         //The camera for the scene.
357         FACamera::Camera mCamera;
358     };
359 }

```

## 6.15 FASwapChain.h

```

1 #pragma once
2
3 #include <wrl.h>
4 #include "d3dx12.h"
5 #include <dxgil_4.h>
6 #include <vector>
7 #include "FABuffer.h"
8
9 namespace FASwapChain
10 {
11     class SwapChain
12     {
13     public:
14
15         SwapChain() = default;
16
17         SwapChain(const Microsoft::WRL::ComPtr<IDXGIFactory4>& dxgiFactory,
18             const Microsoft::WRL::ComPtr<ID3D12CommandQueue>& commandQueue, HWND windowHandle,
19             DXGI_FORMAT rtFormat = DXGI_FORMAT_R8G8B8A8_UNORM, DXGI_FORMAT dsFormat =
20             DXGI_FORMAT_D24_UNORM_S8_UINT,
21             unsigned int numRenderTargetBuffers = 2);
22
23         const RenderTargetBuffer* GetRenderTargetBuffers() const;
24
25         const Microsoft::WRL::ComPtr<ID3D12Resource>& GetCurrentBackBuffer() const;
26
27         unsigned int GetNumRenderTargetBuffers() const;
28
29         unsigned int GetCurrentBackBufferIndex() const;
30
31         DXGI_FORMAT GetBackBufferFormat() const;
32
33         DXGI_FORMAT GetDepthStencilFormat() const;
34
35         void ResetBuffers();
36
37         void ResizeSwapChain(unsigned width, unsigned height);
38
39         void CreateRenderTargetBuffersAndViews(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
40             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int
41             indexWhereToStoreFirstView,
42             unsigned int rtvSize);
43
44         void CreateDepthStencilBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,

```

```

69         const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int index, unsigned int
dsvSize,
70         unsigned int width, unsigned int height);
71
74         void ClearCurrentBackBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList,
75         const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int indexOfFirstView,
unsigned int rtvSize,
76         const float* backBufferClearValue);
77
80         void ClearDepthStencilBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
commandList,
81         const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int indexOfView,
unsigned int dsvSize,
82         float clearValue);
83
86         void Transition(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList,
87         D3D12_RESOURCE_STATES before, D3D12_RESOURCE_STATES after);
88
91         void Present();
92
93     private:
94         unsigned int mNumRenderTargetBuffers = 0;
95         unsigned int mCurrentBackBufferIndex = 0;
96
97         Microsoft::WRL::ComPtr<IDXGISwapChain1> mSwapChain;
98         std::vector<RenderTargetBuffer> mRenderTargetBuffers;
99
100         DepthStencilBuffer mDepthStencilBuffer;
101     };
102 }

```

## 6.16 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.*

#### 6.16.1 Detailed Description

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

## 6.17 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         Text() = default;
12
13         Text(const FAMath::Vector4D& textLocation, const std::wstring& textString, float textSize, const
14             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         FAMath::Vector4D mTextLocation;
34         std::wstring mText;
35         float mTextSize{ 0.0f };
36         FAColor::Color mTextColor;
37     };
38 }

```

## 6.18 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 "FABuffer.h"
10
11 namespace FARender
12 {
13     class TextResources
14     {
15     public:
16         TextResources() = default;
17
18         TextResources(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
19             const Microsoft::WRL::ComPtr<ID3D12CommandQueue>& commandQueue, unsigned int
20             numSwapChainBuffers);
21
22         const Microsoft::WRL::ComPtr<ID2D1DeviceContext>& GetDirect2DDeviceContext() const;
23
24         const Microsoft::WRL::ComPtr<IDWriteFactory>& GetDirectWriteFactory() const;
25
26         void ResetBuffers();
27
28         void ResizeBuffers(const RenderTargetBuffer* renderTargetBuffers, HWND windowHandle);
29
30         void BeforeRenderText(unsigned int currentBackBuffer);
31
32         void AfterRenderText(unsigned int currentBackBuffer);
33
34     private:
35         Microsoft::WRL::ComPtr<ID3D11Device> mDevice11;
36     };
37 }

```

```

53     Microsoft::WRL::ComPtr<ID3D11DeviceContext> mDevice11Context;
54     Microsoft::WRL::ComPtr<ID3D11On12Device> mDevice11on12;
55
56     Microsoft::WRL::ComPtr<ID2D1Device2> mDirect2DDevice;
57     Microsoft::WRL::ComPtr<ID2D1Factory3> mDirect2DFactory;
58     Microsoft::WRL::ComPtr<ID2D1DeviceContext> mDirect2DDeviceContext;
59
60     Microsoft::WRL::ComPtr<IDWriteFactory> mDirectWriteFactory;
61
62     std::vector<Microsoft::WRL::ComPtr<ID3D11Resource>> mWrappedBuffers;
63     std::vector<Microsoft::WRL::ComPtr<ID2D1Bitmap1>> mDirect2DBuffers;
64     std::vector<Microsoft::WRL::ComPtr<IDXGISurface>> mSurfaces;
65 };
66 }

```

## 6.19 FATime.h File Reference

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

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

### Classes

- class [FATime::Time](#)

### 6.19.1 Detailed Description

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

## 6.20 FATime.h

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

```

1  #pragma once
2
7  #include <Windows.h>
8
12 namespace FATime
13 {
14     class Time
15     {
16     public:
20         Time();
21
24         void Tick();
25
28         float DeltaTime() const;
29
32         void Reset();
33
36         void Stop();
37
40         void Start();
41
44         float TotalTime() const;
45
46     private:
47         __int64 mCurrTime; //holds current time stamp ti
48         __int64 mPrevTime; //holds previous time stamp ti-1
49         __int64 mStopTime; //holds the time we stopped the game/animation
50         __int64 mPausedTime; //holds how long the game/animation was paused for
51         __int64 mBaseTime; //holds the time we started / reseted
52
53         double mSecondsPerCount;
54         double mDeltaTime; //time elapsed btw frames change in t = ti - ti-1
55
56         bool mStopped; //flag to indicate if the game/animation is paused or not
57     };
58 };
59 }

```

## 6.21 FAWindow.h File Reference

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

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

### Classes

- class [FAWindow::Window](#)

*The window class is used to make a [Window](#) using Windows API.*

### Namespaces

- namespace [FAWindow](#)

*Has [Window](#) class.*

#### 6.21.1 Detailed Description

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

## 6.22 FAWindow.h

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

```
1 #pragma once
2
3
4
5
6
7 #include <Windows.h>
8 #include <string>
9 #include <stdexcept>
10
11
12
13
14 namespace FAWindow
15 {
16
17     class Window
18     {
19     public:
20         //Window();
21
22         Window(const HINSTANCE& hInstance, const std::wstring& windowClassName, const std::wstring&
windowName,
23             WNDPROC winProcFunction, unsigned int width, unsigned int height, void* additionalData =
nullptr);
24
25         Window(const HINSTANCE& hInstance, const WNDCLASSEX& windowClass, const std::wstring& windowName,
26             unsigned int width, unsigned int height, void* additionalData = nullptr);
27
28         HWND GetWindowHandle() const;
29
30         unsigned int GetWidth() const ;
31
32         unsigned int GetHeight() const;
33
34         void SetWidth(unsigned int width);
35
36         void SetHeight(unsigned int height);
37
38     private:
39         HWND mWindowHandle;
40
41         WNDCLASSEX mWindowClass;
42         std::wstring mWindowClassName;
43
44         unsigned int mWidth;
45         unsigned int mHeight;
46     };
47 }
```

# Index

- ~ConstantBuffer
  - FARender::ConstantBuffer, [23](#)
- ~DeviceResources
  - FARender::DeviceResources, [29](#)
- AddDrawArgument
  - FARender::RenderScene, [41](#)
- AddIndices
  - FARender::RenderScene, [41](#), [42](#)
- AddVertices
  - FARender::RenderScene, [42](#)
- AfterDraw
  - FARender::RenderScene, [42](#)
- AfterDrawObjects
  - FARender::RenderScene, [42](#)
- AfterDrawText
  - FARender::RenderScene, [43](#)
- AfterRenderText
  - FARender::TextResources, [58](#)
- AfterTextDraw
  - FARender::DeviceResources, [29](#)
- Backward
  - FACamera::Camera, [11](#)
- BeforeDrawObjects
  - FARender::RenderScene, [43](#)
- BeforeDrawText
  - FARender::RenderScene, [43](#)
- BeforeRenderText
  - FARender::TextResources, [58](#)
- BeforeTextDraw
  - FARender::DeviceResources, [29](#)
- Camera
  - FACamera::Camera, [11](#)
- ClearCurrentBackBuffer
  - FARender::SwapChain, [52](#)
- ClearDepthStencilBuffer
  - FARender::DepthStencilBuffer, [26](#)
  - FARender::MultiSampling, [37](#)
  - FARender::SwapChain, [52](#)
- ClearRenderTargetBuffer
  - FARender::MultiSampling, [37](#)
  - FARender::RenderTargetBuffer, [49](#)
- Color
  - FAColor::Color, [19](#)
- CopyData
  - FARender::ConstantBuffer, [23](#)
  - FARender::RenderScene, [43](#)
- CreateCBVHeap
  - FARender::RenderScene, [43](#)
- CreateConstantBuffer
  - FARender::ConstantBuffer, [23](#)
  - FARender::RenderScene, [44](#)
- CreateConstantBufferView
  - FARender::ConstantBuffer, [24](#)
  - FARender::RenderScene, [44](#)
- CreateDepthStencilBufferAndView
  - FARender::DepthStencilBuffer, [26](#)
  - FARender::MultiSampling, [37](#)
  - FARender::SwapChain, [52](#)
- CreateDrawSettings
  - FARender::RenderScene, [44](#)
- CreateIndexBuffer
  - FARender::IndexBuffer, [34](#)
  - FARender::RenderScene, [44](#)
- CreateIndexBufferView
  - FARender::IndexBuffer, [35](#)
- CreateRenderTargetBufferAndView
  - FARender::MultiSampling, [37](#)
  - FARender::RenderTargetBuffer, [49](#)
- CreateRenderTargetBuffersAndViews
  - FARender::SwapChain, [52](#)
- CreateText
  - FARender::RenderScene, [44](#)
- CreateVertexBuffer
  - FARender::VertexBuffer, [62](#)
- CreateVertexBufferView
  - FARender::VertexBuffer, [62](#)
- DeltaTime
  - FATime::Time, [60](#)
- DepthStencil, [24](#)
- DepthStencilBuffer
  - FARender::DepthStencilBuffer, [25](#)
- DirectXException, [33](#)
- DisableMSAA
  - FARender::DeviceResources, [30](#)
  - FARender::RenderScene, [45](#)
- Down
  - FACamera::Camera, [12](#)
- DrawObjects
  - FARender::RenderScene, [45](#)
- EnableMSAA
  - FARender::DeviceResources, [30](#)
  - FARender::RenderScene, [45](#)
- Execute
  - FARender::DeviceResources, [30](#)
- ExecuteAndFlush

- FARender::RenderScene, 45
- FABuffer.h, 65
- FACamera, 7
- FACamera.h, 67
  - vec2, 68
- FACamera::Camera, 9
  - Backward, 11
  - Camera, 11
  - Down, 12
  - Foward, 12
  - GetAngularVelocity, 12
  - GetAspectRatio, 12
  - GetCameraPosition, 12
  - GetCameraVelocity, 12
  - GetPerspectiveProjectionMatrix, 13
  - GetVerticalFov, 13
  - GetViewPerspectiveProjectionMatrix, 13
  - GetViewTransformationMatrix, 13
  - GetX, 13
  - GetY, 13
  - GetZ, 14
  - GetZFar, 14
  - GetZNear, 14
  - KeyboardInput, 14
  - Left, 14
  - LookAt, 14
  - MouseInput, 15
  - Right, 15
  - RotateCameraLeftRight, 15
  - RotateCameraUpDown, 15
  - SetAngularVelocity, 15
  - SetAspectRatio, 16
  - SetCameraPosition, 16
  - SetCameraVelocity, 16
  - SetVerticalFov, 16
  - SetX, 16
  - SetY, 16
  - SetZ, 17
  - SetZFar, 17
  - SetZNear, 17
  - Up, 17
  - UpdatePerspectiveProjectionMatrix, 17
  - UpdateViewMatrix, 17
  - UpdateViewPerspectiveProjectionMatrix, 18
- FAColor.h, 70
  - operator\*, 70, 71
  - operator+, 71
  - operator-, 71
- FAColor::Color, 18
  - Color, 19
  - GetAlpha, 20
  - GetBlue, 20
  - GetColor, 20
  - GetGreen, 20
  - GetRed, 20
  - operator==, 20, 21
  - operator+=, 21
  - operator-=, 21
  - SetAlpha, 21
  - SetBlue, 21
  - SetColor, 22
  - SetGreen, 22
  - SetRed, 22
- FADeviceResources.h, 73
- FARender, 7
- FARender::ConstantBuffer, 22
  - ~ConstantBuffer, 23
  - CopyData, 23
  - CreateConstantBuffer, 23
  - CreateConstantBufferView, 24
- FARender::DepthStencilBuffer, 24
  - ClearDepthStencilBuffer, 26
  - CreateDepthStencilBufferAndView, 26
  - DepthStencilBuffer, 25
  - GetDepthStencilFormat, 27
  - ResetBuffer, 27
- FARender::DeviceResources, 28
  - ~DeviceResources, 29
  - AfterTextDraw, 29
  - BeforeTextDraw, 29
  - DisableMSAA, 30
  - EnableMSAA, 30
  - Execute, 30
  - FlushCommandQueue, 30
  - GetBackBufferFormat, 30
  - GetCBVSize, 31
  - GetCommandList, 31
  - GetCurrentFrame, 31
  - GetDepthStencilFormat, 31
  - GetDevice, 31
  - GetInstance, 31
  - GetTextResources, 32
  - IsMSAAEnabled, 32
  - NextFrame, 32
  - Present, 32
  - Resize, 32
  - RTBufferTransition, 32
  - Signal, 33
  - UpdateCurrentFrameFenceValue, 33
  - WaitForGPU, 33
- FARender::DrawSettings, 34
- FARender::IndexBuffer, 34
  - CreateIndexBuffer, 34
  - CreateIndexBufferView, 35
  - GetIndexBufferView, 35
- FARender::MultiSampling, 35
  - ClearDepthStencilBuffer, 37
  - ClearRenderTargetBuffer, 37
  - CreateDepthStencilBufferAndView, 37
  - CreateRenderTargetBufferAndView, 37
  - GetRenderTargetBuffer, 37
  - MultiSampling, 36
  - ResetBuffers, 38
  - Transition, 38
- FARender::RenderScene, 38
  - AddDrawArgument, 41



- AddIndices, [41](#), [42](#)
- AddVertices, [42](#)
- AfterDraw, [42](#)
- AfterDrawObjects, [42](#)
- AfterDrawText, [43](#)
- BeforeDrawObjects, [43](#)
- BeforeDrawText, [43](#)
- CopyData, [43](#)
- CreateCBVHeap, [43](#)
- CreateConstantBuffer, [44](#)
- CreateConstantBufferView, [44](#)
- CreateDrawSettings, [44](#)
- CreateIndexBuffer, [44](#)
- CreateText, [44](#)
- DisableMSAA, [45](#)
- DrawObjects, [45](#)
- EnableMSAA, [45](#)
- ExecuteAndFlush, [45](#)
- IsMSAAEnabled, [46](#)
- NextFrame, [46](#)
- RemoveDrawArgument, [46](#)
- RemoveDrawSettings, [46](#)
- RemoveText, [46](#)
- RenderText, [47](#)
- Resize, [47](#)
- SetPrimitive, [47](#)
- SetPSO, [47](#)
- SetRootSignature, [48](#)
- FARender::RenderTargetBuffer, [48](#)
  - ClearRenderTargetBuffer, [49](#)
  - CreateRenderTargetBufferAndView, [49](#)
  - GetRenderTargetBuffer, [49](#), [50](#)
  - GetRenderTargetFormat, [50](#)
  - RenderTargetBuffer, [49](#)
  - ResetBuffer, [50](#)
- FARender::SwapChain, [50](#)
  - ClearCurrentBackBuffer, [52](#)
  - ClearDepthStencilBuffer, [52](#)
  - CreateDepthStencilBufferAndView, [52](#)
  - CreateRenderTargetBuffersAndViews, [52](#)
  - GetBackBufferFormat, [53](#)
  - GetCurrentBackBuffer, [53](#)
  - GetCurrentBackBufferIndex, [53](#)
  - GetDepthStencilFormat, [53](#)
  - GetNumRenderTargetBuffers, [53](#)
  - GetRenderTargetBuffers, [53](#)
  - Present, [54](#)
  - ResetBuffers, [54](#)
  - ResizeSwapChain, [54](#)
  - SwapChain, [51](#)
  - Transition, [54](#)
- FARender::Text, [55](#)
  - GetTextColor, [56](#)
  - GetTextLocation, [56](#)
  - GetTextSize, [56](#)
  - GetTextString, [56](#)
  - SetTextColor, [56](#)
  - SetTextLocation, [56](#)
  - SetTextSize, [57](#)
  - SetTextString, [57](#)
  - Text, [55](#)
- FARender::TextResources, [57](#)
  - AfterRenderText, [58](#)
  - BeforeRenderText, [58](#)
  - GetDirect2DDeviceContext, [58](#)
  - GetDirectWriteFactory, [58](#)
  - ResetBuffers, [59](#)
  - ResizeBuffers, [59](#)
  - TextResources, [58](#)
- FARender::VertexBuffer, [61](#)
  - CreateVertexBuffer, [62](#)
  - CreateVertexBufferView, [62](#)
  - GetVertexBufferView, [62](#)
- FARenderScene.h, [76](#)
- FAText.h, [81](#)
- FATime.h, [83](#)
- FATime::Time, [59](#)
  - DeltaTime, [60](#)
  - Reset, [60](#)
  - Start, [60](#)
  - Stop, [60](#)
  - Tick, [60](#)
  - Time, [60](#)
  - TotalTime, [61](#)
- FAWindow, [8](#)
- FAWindow.h, [84](#)
- FAWindow::Window, [62](#)
  - GetHeight, [64](#)
  - GetWidth, [64](#)
  - GetWindowHandle, [64](#)
  - SetHeight, [64](#)
  - SetWidth, [64](#)
  - Window, [63](#)
- FlushCommandQueue
  - FARender::DeviceResources, [30](#)
- Foward
  - FACamera::Camera, [12](#)
- GetAlpha
  - FAColor::Color, [20](#)
- GetAngularVelocity
  - FACamera::Camera, [12](#)
- GetAspectRatio
  - FACamera::Camera, [12](#)
- GetBackBufferFormat
  - FARender::DeviceResources, [30](#)
  - FARender::SwapChain, [53](#)
- GetBlue
  - FAColor::Color, [20](#)
- GetCameraPosition
  - FACamera::Camera, [12](#)
- GetCameraVelocity
  - FACamera::Camera, [12](#)
- GetCBVSize
  - FARender::DeviceResources, [31](#)
- GetColor
  - FAColor::Color, [20](#)

- GetCommandList
  - FARender::DeviceResources, 31
- GetCurrentBackBuffer
  - FARender::SwapChain, 53
- GetCurrentBackBufferIndex
  - FARender::SwapChain, 53
- GetCurrentFrame
  - FARender::DeviceResources, 31
- GetDepthStencilFormat
  - FARender::DepthStencilBuffer, 27
  - FARender::DeviceResources, 31
  - FARender::SwapChain, 53
- GetDevice
  - FARender::DeviceResources, 31
- GetDirect2DDeviceContext
  - FARender::TextResources, 58
- GetDirectWriteFactory
  - FARender::TextResources, 58
- GetGreen
  - FAColor::Color, 20
- GetHeight
  - FAWindow::Window, 64
- GetIndexBufferView
  - FARender::IndexBuffer, 35
- GetInstance
  - FARender::DeviceResources, 31
- GetNumRenderTargetBuffers
  - FARender::SwapChain, 53
- GetPerspectiveProjectionMatrix
  - FACamera::Camera, 13
- GetRed
  - FAColor::Color, 20
- GetRenderTargetBuffer
  - FARender::MultiSampling, 37
  - FARender::RenderTargetBuffer, 49, 50
- GetRenderTargetBuffers
  - FARender::SwapChain, 53
- GetRenderTargetFormat
  - FARender::RenderTargetBuffer, 50
- GetTextColor
  - FARender::Text, 56
- GetTextLocation
  - FARender::Text, 56
- GetTextResources
  - FARender::DeviceResources, 32
- GetTextSize
  - FARender::Text, 56
- GetTextString
  - FARender::Text, 56
- GetVertexBufferView
  - FARender::VertexBuffer, 62
- GetVerticalFov
  - FACamera::Camera, 13
- GetViewPerspectiveProjectionMatrix
  - FACamera::Camera, 13
- GetViewTransformationMatrix
  - FACamera::Camera, 13
- GetWidth
  - FAWindow::Window, 64
- GetWindowHandle
  - FAWindow::Window, 64
- GetX
  - FACamera::Camera, 13
- GetY
  - FACamera::Camera, 13
- GetZ
  - FACamera::Camera, 14
- GetZFar
  - FACamera::Camera, 14
- GetZNear
  - FACamera::Camera, 14
- IsMSAAEnabled
  - FARender::DeviceResources, 32
  - FARender::RenderScene, 46
- KeyboardInput
  - FACamera::Camera, 14
- Left
  - FACamera::Camera, 14
- LookAt
  - FACamera::Camera, 14
- MouseInput
  - FACamera::Camera, 15
- MultiSampling
  - FARender::MultiSampling, 36
- NextFrame
  - FARender::DeviceResources, 32
  - FARender::RenderScene, 46
- operator\*
  - FAColor.h, 70, 71
- operator\*=
  - FAColor::Color, 20, 21
- operator+
  - FAColor.h, 71
- operator+=
  - FAColor::Color, 21
- operator-
  - FAColor.h, 71
- operator-=
  - FAColor::Color, 21
- Present
  - FARender::DeviceResources, 32
  - FARender::SwapChain, 54
- RemoveDrawArgument
  - FARender::RenderScene, 46
- RemoveDrawSettings
  - FARender::RenderScene, 46
- RemoveText
  - FARender::RenderScene, 46
- RenderTargetBuffer
  - FARender::RenderTargetBuffer, 49

- RenderText
  - FARender::RenderScene, [47](#)
- Reset
  - FATime::Time, [60](#)
- ResetBuffer
  - FARender::DepthStencilBuffer, [27](#)
  - FARender::RenderTargetBuffer, [50](#)
- ResetBuffers
  - FARender::MultiSampling, [38](#)
  - FARender::SwapChain, [54](#)
  - FARender::TextResources, [59](#)
- Resize
  - FARender::DeviceResources, [32](#)
  - FARender::RenderScene, [47](#)
- ResizeBuffers
  - FARender::TextResources, [59](#)
- ResizeSwapChain
  - FARender::SwapChain, [54](#)
- Right
  - FACamera::Camera, [15](#)
- RotateCameraLeftRight
  - FACamera::Camera, [15](#)
- RotateCameraUpDown
  - FACamera::Camera, [15](#)
- RTBufferTransition
  - FARender::DeviceResources, [32](#)
- SetAlpha
  - FAColor::Color, [21](#)
- SetAngularVelocity
  - FACamera::Camera, [15](#)
- SetAspectRatio
  - FACamera::Camera, [16](#)
- SetBlue
  - FAColor::Color, [21](#)
- SetCameraPosition
  - FACamera::Camera, [16](#)
- SetCameraVelocity
  - FACamera::Camera, [16](#)
- SetColor
  - FAColor::Color, [22](#)
- SetGreen
  - FAColor::Color, [22](#)
- SetHeight
  - FAWindow::Window, [64](#)
- SetPrimitive
  - FARender::RenderScene, [47](#)
- SetPSO
  - FARender::RenderScene, [47](#)
- SetRed
  - FAColor::Color, [22](#)
- SetRootSignature
  - FARender::RenderScene, [48](#)
- SetTextColor
  - FARender::Text, [56](#)
- SetTextLocation
  - FARender::Text, [56](#)
- SetTextSize
  - FARender::Text, [57](#)
- SetTextString
  - FARender::Text, [57](#)
- SetVerticalFov
  - FACamera::Camera, [16](#)
- SetWidth
  - FAWindow::Window, [64](#)
- SetX
  - FACamera::Camera, [16](#)
- SetY
  - FACamera::Camera, [16](#)
- SetZ
  - FACamera::Camera, [17](#)
- SetZFar
  - FACamera::Camera, [17](#)
- SetZNear
  - FACamera::Camera, [17](#)
- Signal
  - FARender::DeviceResources, [33](#)
- Start
  - FATime::Time, [60](#)
- Stop
  - FATime::Time, [60](#)
- SwapChain
  - FARender::SwapChain, [51](#)
- Text
  - FARender::Text, [55](#)
- TextResources
  - FARender::TextResources, [58](#)
- Tick
  - FATime::Time, [60](#)
- Time, [61](#)
  - FATime::Time, [60](#)
- TotalTime
  - FATime::Time, [61](#)
- Transition
  - FARender::MultiSampling, [38](#)
  - FARender::SwapChain, [54](#)
- Up
  - FACamera::Camera, [17](#)
- UpdateCurrentFrameFenceValue
  - FARender::DeviceResources, [33](#)
- UpdatePerspectiveProjectionMatrix
  - FACamera::Camera, [17](#)
- UpdateViewMatrix
  - FACamera::Camera, [17](#)
- UpdateViewPerspectiveProjectionMatrix
  - FACamera::Camera, [18](#)
- vec2
  - FACamera.h, [68](#)
- WaitForGPU
  - FARender::DeviceResources, [33](#)
- Window
  - FAWindow::Window, [63](#)