

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 FAColor Namespace Reference	7
4.2.1 Detailed Description	8
4.2.2 Function Documentation	8
4.2.2.1 operator*() [1/3]	8
4.2.2.2 operator*() [2/3]	8
4.2.2.3 operator*() [3/3]	9
4.2.2.4 operator+()	9
4.2.2.5 operator-()	9
4.3 FARender Namespace Reference	9
4.3.1 Detailed Description	10
4.4 FATime Namespace Reference	10
4.4.1 Detailed Description	10
4.5 FAWindow Namespace Reference	10
4.5.1 Detailed Description	10
<b>5 Class Documentation</b>	<b>11</b>
5.1 FACamera::Camera Class Reference	11
5.1.1 Detailed Description	13
5.1.2 Constructor & Destructor Documentation	13
5.1.2.1 Camera()	13
5.1.3 Member Function Documentation	14
5.1.3.1 Backward()	14
5.1.3.2 Down()	14
5.1.3.3 Foward()	14
5.1.3.4 GetAngularVelocity()	14
5.1.3.5 GetAspectRatio()	15
5.1.3.6 GetCameraPosition()	15
5.1.3.7 GetCameraVelocity()	15
5.1.3.8 GetPerspectiveProjectionMatrix()	15
5.1.3.9 GetVerticalFov()	15
5.1.3.10 GetViewMatrix()	15
5.1.3.11 GetViewPerspectiveProjectionMatrix()	16

5.1.3.12 GetX()	16
5.1.3.13 GetY()	16
5.1.3.14 GetZ()	16
5.1.3.15 GetZFar()	16
5.1.3.16 GetZNear()	16
5.1.3.17 KeyboardInput()	16
5.1.3.18 Left()	17
5.1.3.19 LookAt()	17
5.1.3.20 MouseInput()	17
5.1.3.21 Right()	17
5.1.3.22 RotateCameraLeftRight()	18
5.1.3.23 RotateCameraUpDown()	18
5.1.3.24 SetAngularVelocity()	18
5.1.3.25 SetAspectRatio()	18
5.1.3.26 SetCameraPosition()	19
5.1.3.27 SetCameraVelocity()	19
5.1.3.28 SetVerticalFov()	19
5.1.3.29 SetX()	19
5.1.3.30 SetY()	19
5.1.3.31 SetZ()	19
5.1.3.32 SetZFar()	20
5.1.3.33 SetZNear()	20
5.1.3.34 Up()	20
5.1.3.35 UpdatePerspectiveProjectionMatrix()	20
5.1.3.36 UpdateViewMatrix()	20
5.1.3.37 UpdateViewPerspectiveProjectionMatrix()	21
5.2 FColor::Color Class Reference	21
5.2.1 Detailed Description	22
5.2.2 Constructor & Destructor Documentation	22
5.2.2.1 Color() [1/2]	22
5.2.2.2 Color() [2/2]	22
5.2.3 Member Function Documentation	22
5.2.3.1 GetAlpha()	22
5.2.3.2 GetBlue()	22
5.2.3.3 GetColor()	23
5.2.3.4 GetGreen()	23
5.2.3.5 GetRed()	23
5.2.3.6 operator*=( ) [1/2]	23
5.2.3.7 operator*=( ) [2/2]	23
5.2.3.8 operator+=( )	23
5.2.3.9 operator-=( )	24
5.2.3.10 SetAlpha()	24

5.2.3.11 SetBlue()	24
5.2.3.12 SetColor()	24
5.2.3.13 SetGreen()	24
5.2.3.14 SetRed()	25
5.3 FAREnder::DepthStencilBuffer Class Reference	25
5.3.1 Detailed Description	25
5.3.2 Constructor & Destructor Documentation	25
5.3.2.1 DepthStencilBuffer()	25
5.3.3 Member Function Documentation	26
5.3.3.1 ClearDepthStencilBuffer()	26
5.3.3.2 CreateDepthStencilBufferAndView()	26
5.3.3.3 GetDepthStencilFormat()	27
5.3.3.4 ResetBuffer()	27
5.4 FAREnder::DeviceResources Class Reference	27
5.4.1 Detailed Description	28
5.4.2 Constructor & Destructor Documentation	28
5.4.2.1 ~DeviceResources()	29
5.4.3 Member Function Documentation	29
5.4.3.1 AfterTextDraw()	29
5.4.3.2 BeforeTextDraw()	29
5.4.3.3 Execute()	29
5.4.3.4 FlushCommandQueue()	29
5.4.3.5 GetBackBufferFormat()	29
5.4.3.6 GetCBVSize()	30
5.4.3.7 GetCommandList()	30
5.4.3.8 GetCurrentFrame()	30
5.4.3.9 GetDepthStencilFormat()	30
5.4.3.10 GetDevice()	30
5.4.3.11 GetInstance()	30
5.4.3.12 GetTextResources()	31
5.4.3.13 NextFrame()	31
5.4.3.14 Present()	31
5.4.3.15 Resize()	31
5.4.3.16 RTBufferTransition()	32
5.4.3.17 Signal()	32
5.4.3.18 UpdateCurrentFrameFenceValue()	32
5.4.3.19 WaitForGPU()	32
5.4.4 Member Data Documentation	33
5.4.4.1 NUM_OF_FRAMES	33
5.5 DirectXException Class Reference	33
5.5.1 Detailed Description	33
5.5.2 Constructor & Destructor Documentation	33

5.5.2.1 DirectXException()	33
5.5.3 Member Function Documentation	34
5.5.3.1 ErrorMsg()	34
5.6 FAREnder::DynamicBuffer Class Reference	34
5.6.1 Detailed Description	35
5.6.2 Constructor & Destructor Documentation	35
5.6.2.1 ~DynamicBuffer()	35
5.6.3 Member Function Documentation	35
5.6.3.1 CopyData()	35
5.6.3.2 CreateConstantBufferView()	36
5.6.3.3 CreateDynamicBuffer() [1/2]	36
5.6.3.4 CreateDynamicBuffer() [2/2]	36
5.6.3.5 CreateIndexBufferView()	38
5.6.3.6 CreateVertexBufferView()	38
5.6.3.7 GetFormat()	38
5.6.3.8 GetGPUAddress()	38
5.6.3.9 GetIndexBufferView()	39
5.6.3.10 GetStride()	39
5.6.3.11 GetVertexBufferView()	39
5.7 FAREnder::MultiSampling Class Reference	39
5.7.1 Detailed Description	40
5.7.2 Constructor & Destructor Documentation	40
5.7.2.1 MultiSampling()	40
5.7.3 Member Function Documentation	40
5.7.3.1 ClearDepthStencilBuffer()	41
5.7.3.2 ClearRenderTargetBuffer()	41
5.7.3.3 CreateDepthStencilBufferAndView()	41
5.7.3.4 CreateRenderTargetBufferAndView()	42
5.7.3.5 GetRenderTargetBuffer()	42
5.7.3.6 ResetBuffers()	43
5.7.3.7 Transition()	43
5.8 FAREnder::RenderScene Class Reference	43
5.8.1 Detailed Description	45
5.8.2 Member Function Documentation	45
5.8.2.1 AfterRender()	45
5.8.2.2 AfterRenderObjects()	45
5.8.2.3 AfterRenderText()	45
5.8.2.4 BeforeRenderObjects()	46
5.8.2.5 BeforeRenderText()	46
5.8.2.6 CompileShader()	46
5.8.2.7 CopyDataIntoDynamicBuffer()	46
5.8.2.8 CreateDynamicBuffer()	47

5.8.2.9 CreateInputElementDescription()	47
5.8.2.10 CreatePSO()	48
5.8.2.11 CreateRootParameter()	49
5.8.2.12 CreateRootSignature()	49
5.8.2.13 CreateStaticBuffer()	49
5.8.2.14 ExecuteAndFlush()	50
5.8.2.15 LoadShader()	50
5.8.2.16 RemoveShader()	50
5.8.2.17 RenderObject()	51
5.8.2.18 RenderText()	51
5.8.2.19 Resize()	52
5.8.2.20 SetDynamicBuffer()	52
5.8.2.21 SetPSOAndRootSignature()	53
5.8.2.22 SetStaticBuffer()	53
5.9 FARender::RenderTargetBuffer Class Reference	53
5.9.1 Detailed Description	54
5.9.2 Constructor & Destructor Documentation	54
5.9.2.1 RenderTargetBuffer()	54
5.9.3 Member Function Documentation	54
5.9.3.1 ClearRenderTargetBuffer()	55
5.9.3.2 CreateRenderTargetBufferAndView()	55
5.9.3.3 GetRenderTargetBuffer() [1/2]	55
5.9.3.4 GetRenderTargetBuffer() [2/2]	56
5.9.3.5 GetRenderTargetFormat()	56
5.9.3.6 ResetBuffer()	56
5.10 FARender::StaticBuffer Class Reference	56
5.10.1 Detailed Description	57
5.10.2 Member Function Documentation	57
5.10.2.1 CreateIndexBufferView()	57
5.10.2.2 CreateStaticBuffer()	57
5.10.2.3 CreateVertexBufferView()	57
5.10.2.4 GetIndexBufferView()	59
5.10.2.5 GetVertexBufferView()	59
5.11 FARender::SwapChain Class Reference	59
5.11.1 Detailed Description	60
5.11.2 Constructor & Destructor Documentation	60
5.11.2.1 SwapChain()	60
5.11.3 Member Function Documentation	61
5.11.3.1 ClearCurrentBackBuffer()	61
5.11.3.2 ClearDepthStencilBuffer()	61
5.11.3.3 CreateDepthStencilBufferAndView()	62
5.11.3.4 CreateRenderTargetBuffersAndViews()	62

5.11.3.5 GetBackBufferFormat()	63
5.11.3.6 GetCurrentBackBuffer()	63
5.11.3.7 GetCurrentBackBufferIndex()	63
5.11.3.8 GetDepthStencilFormat()	63
5.11.3.9 GetNumRenderTargetBuffers()	63
5.11.3.10 GetRenderTargetBuffers()	63
5.11.3.11 Present()	64
5.11.3.12 ResetBuffers()	64
5.11.3.13 ResizeSwapChain()	64
5.11.3.14 Transition()	64
5.12 FARender::Text Class Reference	65
5.12.1 Detailed Description	65
5.12.2 Constructor & Destructor Documentation	65
5.12.2.1 Text()	65
5.12.3 Member Function Documentation	66
5.12.3.1 GetTextColor()	66
5.12.3.2 GetTextLocation()	66
5.12.3.3 GetTextSize()	66
5.12.3.4 GetTextString()	66
5.12.3.5 SetTextColor()	67
5.12.3.6 SetTextLocation()	67
5.12.3.7 SetTextSize()	67
5.12.3.8 SetTextString()	67
5.13 FARender::TextResources Class Reference	67
5.13.1 Detailed Description	68
5.13.2 Constructor & Destructor Documentation	68
5.13.2.1 TextResources()	68
5.13.3 Member Function Documentation	68
5.13.3.1 AfterRenderText()	69
5.13.3.2 BeforeRenderText()	69
5.13.3.3 GetDirect2DDeviceContext()	69
5.13.3.4 GetDirectWriteFactory()	69
5.13.3.5 ResetBuffers()	69
5.13.3.6 ResizeBuffers()	70
5.14 FATime::Time Class Reference	70
5.14.1 Detailed Description	70
5.14.2 Constructor & Destructor Documentation	70
5.14.2.1 Time()	71
5.14.3 Member Function Documentation	71
5.14.3.1 DeltaTime()	71
5.14.3.2 Reset()	71
5.14.3.3 Start()	71



5.14.3.4 Stop()	71
5.14.3.5 Tick()	71
5.14.3.6 TotalTime()	72
5.15 FAWindow::Window Class Reference	72
5.15.1 Detailed Description	72
5.15.2 Constructor & Destructor Documentation	72
5.15.2.1 Window() [1/2]	72
5.15.2.2 Window() [2/2]	73
5.15.3 Member Function Documentation	73
5.15.3.1 GetHeight()	73
5.15.3.2 GetWidth()	74
5.15.3.3 GetWindowHandle()	74
5.15.3.4 SetHeight()	74
5.15.3.5 SetWidth()	74
<b>6 File Documentation</b>	<b>75</b>
6.1 Direct3DLink.h	75
6.2 FABuffer.h File Reference	75
6.2.1 Detailed Description	76
6.3 FABuffer.h	76
6.4 FACamera.h File Reference	78
6.4.1 Detailed Description	78
6.4.2 Typedef Documentation	78
6.4.2.1 vec2	78
6.5 FACamera.h	79
6.6 FAColor.h File Reference	80
6.6.1 Detailed Description	81
6.7 FAColor.h	81
6.8 FADeviceResources.h File Reference	82
6.8.1 Detailed Description	82
6.9 FADeviceResources.h	82
6.10 FADirectXException.h	84
6.11 FAMultiSampling.h	84
6.12 FARenderScene.h File Reference	85
6.12.1 Detailed Description	85
6.13 FARenderScene.h	86
6.14 FASwapChain.h	87
6.15 FAText.h File Reference	88
6.15.1 Detailed Description	88
6.16 FAText.h	89
6.17 FATextResources.h	89
6.18 FATime.h File Reference	90

6.18.1 Detailed Description . . . . .	90
6.19 FATime.h . . . . .	90
6.20 FAWindow.h File Reference . . . . .	91
6.20.1 Detailed Description . . . . .	91
6.21 FAWindow.h . . . . .	91
<b>Index</b>	<b>93</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 . . . . .	7
<a href="#">FAColor</a>	Has the <a href="#">Color</a> class . . . . .	7
<a href="#">FARender</a>	Has classes that are used for rendering objects and text through the Direct3D 12 API . . . . .	9
<a href="#">FATime</a>	Has <a href="#">Time</a> class . . . . .	10
<a href="#">FAWindow</a>	Has <a href="#">Window</a> class . . . . .	10



## 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.

11

### [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

21

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

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

25

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

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

27

### [DirectXException](#)

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

33

### [FARender::DynamicBuffer](#)

This class stores data in a Direct3D 12 upload buffer

34

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

A wrapper for multisampling resources. Uses DirectD 12 API

39

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

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

43

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

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

53

### [FARender::StaticBuffer](#)

This class stores data in a Direct3D 12 default buffer

56

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

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

59

### [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

65

### [FARender::TextResources](#)

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

67

[FATime::Time](#)

This class is used to get the time between each frame. You can stop start, reset and get the total time . . . . . 70

[FAWindow::Window](#)

The window class is used to make a [Window](#) using Windows API . . . . . 72

## 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 <a href="#">RenderTargetBuffer</a> , <a href="#">DepthStencilBuffer</a> , <a href="#">StaticBuffer</a> and <a href="#">DynamicBuffer</a> under namespace <a href="#">FARender</a>	75
<a href="#">FACamera.h</a>	
File that has namespace <a href="#">FACamera</a> . Withn the namespace is the class <a href="#">Camera</a>	78
<a href="#">FAColor.h</a>	
File has class <a href="#">Color</a> under namespace <a href="#">FAColor</a>	80
<a href="#">FADeviceResources.h</a>	
File has class <a href="#">DeviceResources</a> under namespace <a href="#">FARender</a>	82
<a href="#">FADirectXException.h</a>	??
<a href="#">FAMultiSampling.h</a>	??
<a href="#">FARenderScene.h</a>	
File has class <a href="#">RenderScene</a> under namespace <a href="#">FARender</a>	85
<a href="#">FASwapChain.h</a>	??
<a href="#">FAText.h</a>	
File has class <a href="#">Text</a> under namespace <a href="#">FARender</a>	88
<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>	90
<a href="#">FAWindow.h</a>	
File that has namespace <a href="#">FAWindow</a> . Withn the namespace is the class <a href="#">Window</a>	91





## 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 FAColor Namespace Reference

Has the [Color](#) class.

#### Classes

- class [Color](#)

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

## Functions

- **Color operator+** (const **Color** &c1, const **Color** &c2)  
*Returns the result of  $c1 + c2$ .*
- **Color operator-** (const **Color** &c1, const **Color** &c2)  
*Returns the result of  $c1 - c2$ .*
- **Color operator\*** (const **Color** &c, float k)  
*Returns the result of  $c * k$ .*
- **Color operator\*** (float k, const **Color** &c)  
*Returns the result of  $k * c$ .*
- **Color operator\*** (const **Color** &c1, const **Color** &c2)  
*Returns the result of  $c1 * c2$ .*

### 4.2.1 Detailed Description

Has the **Color** class.

### 4.2.2 Function Documentation

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

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

Returns the result of  $c * k$ .

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

#### 4.2.2.2 operator\*() [2/3]

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

Returns the result of  $c1 * c2$ .

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

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

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

Returns the result of  $k * c$ .

If  $k < 0.0f$ , no multiplication happens and [Color](#)  $c$  is returned.

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

#### 4.2.2.4 operator+()

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

Returns the result of  $c1 + c2$ .

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

#### 4.2.2.5 operator-()

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

Returns the result of  $c1 - c2$ .

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

## 4.3 FARender Namespace Reference

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

### Classes

- class [DepthStencilBuffer](#)  
*A wrapper for depth stencil buffer resources. Uses DirectD 12 API.*
- class [DeviceResources](#)  
*A wrapper for resources that are needed to render objects and text using the Direct3D 12 API.*
- class [DynamicBuffer](#)  
*This class stores data in a Direct3D 12 upload 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 [StaticBuffer](#)  
*This class stores data in a Direct3D 12 default buffer.*
- 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.*

### 4.3.1 Detailed Description

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

## 4.4 FATime Namespace Reference

Has [Time](#) class.

### Classes

- class [Time](#)

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

### 4.4.1 Detailed Description

Has [Time](#) class.

## 4.5 FAWindow Namespace Reference

Has [Window](#) class.

### Classes

- class [Window](#)

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

### 4.5.1 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)  
*Creates a new camera.*
- const vec3 & [GetCameraPosition](#) () const  
*Returns a constant reference to the position of the camera in world coordinates.*
- const vec3 & [GetX](#) () const  
*Returns a constant reference to the x-axis of the camera.*
- const vec3 & [GetY](#) () const  
*Returns a constant reference to the y-axis of the camera.*
- const vec3 & [GetZ](#) () const  
*Returns a constant reference to the z-axis of the camera.*
- const mat4 & [GetViewMatrix](#) () const  
*Returns a constant reference to the view transformation matrix of this camera.*
- float [GetCameraVelocity](#) () const  
*Returns the camera's velocity.*
- float [GetAngularVelocity](#) () const  
*Returns the camera's angular velocity.*
- void [LookAt](#) (vec3 cameraPosition, vec3 target, vec3 up)  
*Defines the camera space using UVN.*
- 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 x.*
- void [SetY](#) (const vec3 &y)

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

*Sets the camera's z-axis to the specified vector z.*
- void [SetCameraVelocity](#) (float velocity)

*Sets the camera's velocity to the specified velocity.*
- void [SetAngularVelocity](#) (float velocity)

*Sets the camera's angular velocity to the specified angular velocity.*
- void [SetZNear](#) (float znear)

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

*Sets the camera's far plane value to the specified value.*
- void [SetVerticalFov](#) (float fov)

*Sets the camera's vertical field of view to the specified vertical field of view .*
- void [SetAspectRatio](#) (float ar)

*Sets the camera's aspect ratio to the specified aspect ratio.*
- 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](#) ()

*Updates the camera's view perspective projection matrix.*
- void [Left](#) (float dt)

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

*Moves the camera right along the camera's x-axis.*
- void [Forward](#) (float dt)

*Moves the camera forward along the camera's z-axis.*
- void [Backward](#) (float dt)

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

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

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

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

*Rotates the camera to look up and down.*

- void [KeyboardInput](#) (float dt)  
*Polls keyboard input and moves the camera.*
- void [MouseInput](#) ()  
*Polls mouse input and rotates the camera.*

### 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 )
```

Creates a new camera.

#### Parameters

in	<i>camerPosition</i>	The position of the camera.
in	<i>x</i>	The x axis of the local coordinate system of the camera.
in	<i>y</i>	The y axis of the local coordinate system of the camera.
in	<i>z</i>	The z axis of the local coordinate system of the camera.
in	<i>znear</i>	The distance of the near plane from the camera.
in	<i>zfar</i>	The distance of the far plane from the camera.
in	<i>aspectRatio</i>	The aspect ratio of the view plane.
in	<i>vFov</i>	The vertical field of view of the frustrum.
in	<i>cameraVelocity</i>	The translational velocity of the camera.
in	<i>angularVelocity</i>	The angular velocity of the camera.

## 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.

#### Parameters

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

### 5.1.3.2 Down()

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

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

#### Parameters

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

### 5.1.3.3 Foward()

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

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

#### Parameters

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

### 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 GetViewMatrix()

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

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

#### 5.1.3.11 GetViewPerspectiveProjectionMatrix()

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

Returns a constant reference to the view perspective projection 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.

## Parameters

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

**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.

## Parameters

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

**5.1.3.20 MouseInput()**

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

Polls mouse input and rotates the camera.

**5.1.3.21 Right()**

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

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

**Parameters**

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

**5.1.3.22 RotateCameraLeftRight()**

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

Rotates the camera to look left and right.

**Parameters**

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

**5.1.3.23 RotateCameraUpDown()**

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

Rotates the camera to look up and down.

**Parameters**

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

**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 *x*.

#### 5.1.3.30 SetY()

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

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

#### 5.1.3.31 SetZ()

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

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

#### 5.1.3.32 SetZFar()

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

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

#### 5.1.3.33 SetZNear()

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

Sets the camera's near plane 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.

##### Parameters

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

#### 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 FCamera::Camera::UpdateViewPerspectiveProjectionMatrix ( )
```

Updates the camera's view perspective projection matrix.

Call this to rebuild the view perspective projection transformation matrix.

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

- [FCamera.h](#)

## 5.2 FColor::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 "FColor.h"
```

### Public Member Functions

- [Color](#) (float r=0.0f, float g=0.0f, float b=0.0f, float a=1.0f)  
*Initializes the color to the specified RGBA values.*
- [Color](#) (const FAMath::Vector4D &color)  
*Initializes the color to the specified color.*
- const FAMath::Vector4D & [GetColor](#) () const  
*Returns the color.*
- float [GetRed](#) () const  
*Returns the value of the red component.*
- float [GetGreen](#) () const  
*Returns the value of the blue component.*
- float [GetBlue](#) () const  
*Returns the value of the green component.*
- float [GetAlpha](#) () const  
*Returns the value of the alpha component.*
- void [SetColor](#) (const FAMath::Vector4D &color)  
*Sets the color to the specified color.*
- void [SetRed](#) (float r)  
*Sets the red component to the specified float value.*
- void [SetGreen](#) (float g)  
*Sets the green component to the specified float value.*
- void [SetBlue](#) (float b)  
*Sets the blue component to the specified float value.*
- void [SetAlpha](#) (float a)  
*Sets the alpha component to the specified float value.*
- [Color](#) & [operator+=](#) (const [Color](#) &c)  
*Adds this objects color to the specified color c and stores the result in this object.*
- [Color](#) & [operator-=](#) (const [Color](#) &c)  
*Subtracts the specified color c from this objects color and stores the result in this object.*
- [Color](#) & [operator\\*=](#) (float k)  
*Multiplies this objects color by the specified value k and stores the result in this object.*
- [Color](#) & [operator\\*=](#) (const [Color](#) &c)  
*Multiplies this objects color by the specified color c and stores the result in this object.*

### 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]

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

Initializes the color to the specified RGBA values.

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

```
FAColor::Color::Color (
    const FAMath::Vector4D & color )
```

Initializes the color to the specified *color*.

### 5.2.3 Member Function Documentation

#### 5.2.3.1 GetAlpha()

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

Returns the value of the alpha component.

#### 5.2.3.2 GetBlue()

```
float FAColor::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 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 *c* and stores the result in this object.

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

#### 5.2.3.9 operator-=( )

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

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

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

#### 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 FIColor::Color::SetRed (
    float r )
```

Sets the red component to the specified float value.

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

- [FIColor.h](#)

## 5.3 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](#) ()  
*Resets 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.*

### 5.3.1 Detailed Description

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

### 5.3.2 Constructor & Destructor Documentation

#### 5.3.2.1 DepthStencilBuffer()

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

Default Constructor.

## Parameters

in	<i>format</i>	The format of the depth stencil buffer.
----	---------------	---

### 5.3.3 Member Function Documentation

#### 5.3.3.1 ClearDepthStencilBuffer()

```
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.

## Parameters

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

#### 5.3.3.2 CreateDepthStencilBufferAndView()

```
void 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.

## Parameters

in	<i>device</i>	A Direct3D 12 device.
in	<i>dsvHeap</i>	A descriptor heap for storing depth stencil descriptors.
in	<i>indexOfWhereToStoreView</i>	The index of where to store the created descriptor in the descriptor heap.

## Parameters

in	<i>dsvSize</i>	The size of a depth stencil descriptor.
in	<i>width</i>	The width of the depth stencil buffer.
in	<i>height</i>	The height of the depth stencil buffer.
in	<i>sampleCount</i>	The sample count of the depth stencil buffer.

## 5.3.3.3 GetDepthStencilFormat()

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

Returns the format of the depth stencil buffer.

## 5.3.3.4 ResetBuffer()

```
void FARender::DepthStencilBuffer::ResetBuffer ( )
```

Resets the depth stencil buffer.

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

- [FABuffer.h](#)

## 5.4 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.*

- unsigned int [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.*
- void [UpdateCurrentFrameFenceValue](#) ()  
*Updates the current frames fence value.*
- void [FlushCommandQueue](#) ()  
*Synchronizes the CPU and GPU.*
- void [WaitForGPU](#) () const  
*Waits for the GPU to execute all of the commands of the current frame.*
- void [Signal](#) ()  
*Adds an instruction to the GPU to set the fence value to the current fence value.*
- void [Resize](#) (int width, int height, const HWND &handle, bool isMSAAEnabled, bool isTextEnabled)  
*Call when the window gets resized.*
- void [RTBufferTransition](#) (bool isMSAAEnabled, bool isTextEnabled)  
*Transitions the render target buffer.*
- void [BeforeTextDraw](#) ()  
*Prepares to render text.*
- void [AfterTextDraw](#) ()  
*Executes the text commands.*
- void [Execute](#) () const  
*Executes the command list.*
- void [Present](#) ()  
*Swaps the front and back buffers.*
- void [Draw](#) (bool isMSAAEnabled)
- void [NextFrame](#) ()  
*Updates the current frame value to go to the next frame.*

## Static Public Member Functions

- static [DeviceResources](#) & [GetInstance](#) (unsigned int width, unsigned int height, HWND windowHandle, bool isMSAAEnabled, bool isTextEnabled)  
*Call to make an object of [DeviceResources](#).*

## Static Public Attributes

- static const unsigned int [NUM\\_OF\\_FRAMES](#) { 3 }  
*The number of frames in the ciruclar array.*

### 5.4.1 Detailed Description

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

### 5.4.2 Constructor & Destructor Documentation

#### 5.4.2.1 ~DeviceResources()

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

Flushes the command queue.

### 5.4.3 Member Function Documentation

#### 5.4.3.1 AfterTextDraw()

```
void FAREnder::DeviceResources::AfterTextDraw ( )
```

Executes the text commands.

#### 5.4.3.2 BeforeTextDraw()

```
void FAREnder::DeviceResources::BeforeTextDraw ( )
```

Prepares to render text.

#### 5.4.3.3 Execute()

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

Executes the command list.

#### 5.4.3.4 FlushCommandQueue()

```
void FAREnder::DeviceResources::FlushCommandQueue ( )
```

Synchronizes the CPU and GPU.

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

#### 5.4.3.5 GetBackBufferFormat()

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

Returns a constant reference to the back buffer format.

#### 5.4.3.6 GetCBVSize()

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

The size of a constant buffer view.

#### 5.4.3.7 GetCommandList()

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

Returns a constant reference to the ID3D12GraphicsCommandList object.

#### 5.4.3.8 GetCurrentFrame()

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

Returns the current frame.

#### 5.4.3.9 GetDepthStencilFormat()

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

Returns a constant reference to the depth stencil format.

#### 5.4.3.10 GetDevice()

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

Returns a constant reference to the ID3D12Device object.

#### 5.4.3.11 GetInstance()

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

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

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



## Parameters

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

#### 5.4.3.12 GetTextResources()

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

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

#### 5.4.3.13 NextFrame()

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

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

#### 5.4.3.14 Present()

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

Swaps the front and back buffers.

#### 5.4.3.15 Resize()

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

Call when the window gets resized.

Call when you initialize your program.

**Parameters**

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

**5.4.3.16 RTBufferTransition()**

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

Transistions the render target buffer.

**Parameters**

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

**5.4.3.17 Signal()**

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

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

**5.4.3.18 UpdateCurrentFrameFenceValue()**

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

Updates the current frames fence value.

**5.4.3.19 WaitForGPU()**

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

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

Signal should have been called before this function is called.

## 5.4.4 Member Data Documentation

### 5.4.4.1 NUM\_OF\_FRAMES

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

The number of frames in the circular array.

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

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

- [FADeviceResources.h](#)

## 5.5 DirectXException Class Reference

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

```
#include "FADirectXException.h"
```

### Public Member Functions

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

### 5.5.1 Detailed Description

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

### 5.5.2 Constructor & Destructor Documentation

#### 5.5.2.1 DirectXException()

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

Constructs a [DirectXException](#) object.

## Parameters

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

### 5.5.3 Member Function Documentation

#### 5.5.3.1 ErrorMsg()

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

Returns a message describing the error.

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

- FADirectXException.h

## 5.6 FARender::DynamicBuffer Class Reference

This class stores data in a Direct3D 12 upload buffer.

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

### Public Member Functions

- **DynamicBuffer** (const [DynamicBuffer](#) &)=delete
- **DynamicBuffer** & **operator=** (const [DynamicBuffer](#) &)=delete
- **DynamicBuffer** ([DynamicBuffer](#) &&)=default
- **~DynamicBuffer** ()  
*Unmaps the pointer to the dynamic buffer.*
- D3D12\_GPU\_VIRTUAL\_ADDRESS [GetGPUAddress](#) () const  
*Returns the GPU virtual address of the first byte of the dynamic buffer.*
- const unsigned int & [GetStride](#) () const  
*Returns the stride of the dynamic buffer.*
- const DXGI\_FORMAT & [GetFormat](#) () const  
*Returns the format of the dynamic buffer.*
- void [CreateDynamicBuffer](#) (const Microsoft::WRL::ComPtr< ID3D12Device > &device, UINT numBytes, UINT stride)  
*Creates and maps the dynamic buffer.*
- void [CreateDynamicBuffer](#) (const Microsoft::WRL::ComPtr< ID3D12Device > &device, UINT numBytes, DXGI\_FORMAT format)  
*Creates and maps the dynamic buffer.*

- void [CreateConstantBufferView](#) (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &cbvHeap, UINT cbvSize, UINT cbvHeapIndex, UINT cBufferIndex)  
*Creates the constant buffer view and stores it in the specified descriptor heap.*
- void [CreateVertexBufferView](#) (UINT numBytes)  
*Creates a vertex buffer view and stores it.*
- void [CreateIndexBufferView](#) (UINT numBytes)  
*Creates an index buffer view and stores it.*
- const D3D12\_VERTEX\_BUFFER\_VIEW & [GetVertexBufferView](#) ()  
*Returns a constant reference to the vertex buffer view.*
- const D3D12\_INDEX\_BUFFER\_VIEW & [GetIndexBufferView](#) ()  
*Returns a constant reference to the index buffer view.*
- void [CopyData](#) (UINT index, const void \*data, UINT64 numBytes)  
*Copies data from the given data into the dynamic buffer. Uses 0-indexing.*

### 5.6.1 Detailed Description

This class stores data in a Direct3D 12 upload buffer.

### 5.6.2 Constructor & Destructor Documentation

#### 5.6.2.1 ~DynamicBuffer()

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

Unmaps the pointer to the dynamic buffer.

### 5.6.3 Member Function Documentation

#### 5.6.3.1 CopyData()

```
void FAREnder::DynamicBuffer::CopyData (
    UINT index,
    const void * data,
    UINT64 numBytes )
```

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

#### Parameters

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

### 5.6.3.2 CreateConstantBufferView()

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

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

#### Parameters

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

### 5.6.3.3 CreateDynamicBuffer() [1/2]

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

Creates and maps the dynamic buffer.

Call if you want to create a dynamic index buffer.

#### Parameters

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

### 5.6.3.4 CreateDynamicBuffer() [2/2]

```
void FARender::DynamicBuffer::CreateDynamicBuffer (
    const Microsoft::WRL::ComPtr< ID3D12Device > & device,
    UINT numOfBytes,
    UINT stride )
```

Creates and maps the dynamic buffer.

Call if you want to create a dynamic vertex buffer or dynamic constant buffer.

## Parameters

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

**5.6.3.5 CreateIndexBufferView()**

```
void FARender::DynamicBuffer::CreateIndexBufferView (
    UINT numBytes )
```

Creates a index buffer view and stores it.

## Parameters

in	<i>numBytes</i>	The number of bytes in the dynamic buffer.
----	-----------------	--

**5.6.3.6 CreateVertexBufferView()**

```
void FARender::DynamicBuffer::CreateVertexBufferView (
    UINT numBytes )
```

Creates a vertex buffer view and stores it.

## Parameters

in	<i>numBytes</i>	The number of bytes in the dynamic buffer.
----	-----------------	--

**5.6.3.7 GetFormat()**

```
const DXGI_FORMAT & FARender::DynamicBuffer::GetFormat ( ) const
```

Returns the format of the dynamic buffer.

**5.6.3.8 GetGPUAddress()**

```
D3D12_GPU_VIRTUAL_ADDRESS FARender::DynamicBuffer::GetGPUAddress ( ) const
```

Returns the GPU virtual address of the first byte of the dynamic buffer.



### 5.6.3.9 GetIndexBufferView()

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

Returns a constant reference to the vertex buffer view.

### 5.6.3.10 GetStride()

```
const unsigned int & FAREnder::DynamicBuffer::GetStride ( ) const
```

Returns the stride of the dynamic buffer.

### 5.6.3.11 GetVertexBufferView()

```
const D3D12_VERTEX_BUFFER_VIEW & FAREnder::DynamicBuffer::GetVertexBufferView ( )
```

Returns a constant reference to the vertex buffer view.

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

- [FABuffer.h](#)

## 5.7 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)  
*Checks if the specified format and sample count are supported by the specified device for multi-sampling.*
- 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.*

## 5.7.1 Detailed Description

A wrapper for multisampling resources. Uses DirectD 12 API.

## 5.7.2 Constructor & Destructor Documentation

### 5.7.2.1 MultiSampling()

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

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

Throws a runtime\_error if they are not supported.

#### Parameters

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

## 5.7.3 Member Function Documentation

### 5.7.3.1 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.

#### Parameters

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

### 5.7.3.2 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.

#### Parameters

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

### 5.7.3.3 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.

#### Parameters

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

#### 5.7.3.4 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.

#### Parameters

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

#### 5.7.3.5 GetRenderTargetBuffer()

```

const Microsoft::WRL::ComPtr< ID3D12Resource > & FARender::MultiSampling::GetRenderTarget↵
Buffer ( )

```

Returns the MSAA render target buffer.

### 5.7.3.6 ResetBuffers()

```
void FAREnder::MultiSampling::ResetBuffers ( )
```

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

### 5.7.3.7 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.

#### Parameters

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

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

- FAMultiSampling.h

## 5.8 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, bool isMSAAEnabled=false, bool isEnabled=false)
- **RenderScene** (const [RenderScene](#) &)=delete
- **RenderScene & operator=** (const [RenderScene](#) &)=delete
- **RenderScene** ([RenderScene](#) &&)=default
- void [LoadShader](#) (unsigned int shaderKey, const std::wstring &filename)
 

*Loads a shaders bytecode and maps it to the specified shaderKey.*
- void [CompileShader](#) (unsigned int shaderKey, const std::wstring &filename, const std::string &entryPoint↔ Name, const std::string &target)
 

*Loads a shader file, compiles it into bytecode and and maps the bytecode to the specified shaderKey.*
- void [RemoveShader](#) (unsigned int shaderKey)
 

*Removes the shader bytecode mapped to the specified shaderKey.*
- void [CreateStaticBuffer](#) (unsigned int bufferType, unsigned int staticBufferKey, const void \*data, unsigned numBytes, unsigned int stride=0, DXGI\_FORMAT format=DXGI\_FORMAT\_R32\_UINT)

Creates a static buffer and stores the specified data in the buffer. The user cannot update/change the data once it is stored in the buffer.

Make sure to pass in different keys to store the static buffers with to prevent replacing the static buffer at that key with the newly created static buffer.

- void [CreateDynamicBuffer](#) (unsigned int bufferSize, unsigned int dynamicBufferKey, unsigned numBytes, unsigned int stride=0, DXGI\_FORMAT format=DXGI\_FORMAT\_R32\_UINT)

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

Make sure to pass in different keys to store the dynamic buffers with to prevent replacing the dynamic buffer at that key with the newly created dynamic buffer.

- void [CreateInputElementDescription](#) (unsigned int key, const char \*semanticName, unsigned int semanticIndex, DXGI\_FORMAT format, unsigned int inputSlot, unsigned int byteOffset, D3D12\_INPUT\_CLASSIFICATION inputSlotClass=D3D12\_INPUT\_CLASSIFICATION\_PER\_VERTEX\_DATA, unsigned int instanceStepRate=0)

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

- void [CreateRootParameter](#) (unsigned int rootParameterKey, unsigned int shaderRegister)

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

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

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

- void [CreatePSO](#) (unsigned int psKey, D3D12\_FILL\_MODE fillMode, BOOL enableMultisample, unsigned int vsKey, unsigned int psKey, unsigned int inputElementDescriptionsKey, unsigned int rootSigKey, const D3D12\_PRIMITIVE\_TOPOLOGY\_TYPE &primitiveType, UINT sampleCount=1)

Creates a PSO and maps it to the specified psKey.

- void [SetPSOAndRootSignature](#) (unsigned int psKey, unsigned int rootSigKey)

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

- void [SetStaticBuffer](#) (unsigned int bufferSize, unsigned int staticBufferKey)

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

- void [SetDynamicBuffer](#) (unsigned int bufferSize, unsigned int dynamicBufferKey, unsigned int index, ConstantData=0, unsigned int rootParameterIndex=0)

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

- void [BeforeRenderObjects](#) (bool isMSAAEnabled=false)

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

- void [RenderObject](#) (unsigned int indexCount, unsigned int locationFirstIndex, int indexOfFirstVertex, D3D12\_PRIMITIVE\_TOPOLOGY primitive)

Renders an object with the specified draw arguments.

- void [AfterRenderObjects](#) (bool isMSAAEnabled=false, bool isTextEnabled=false)

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

- void [BeforeRenderText](#) ()

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

- void [RenderText](#) (const FAMath::Vector4D &textLocation, const FColor::Color &textColor, float textSize, const std::wstring &textString)

Draws the Text object mapped to the specified textKey. Call in between a BeforeRenderText function and a AfterRenderText function.

- void [AfterRenderText](#) ()

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

- void [AfterRender](#) ()

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

- void [ExecuteAndFlush](#) ()

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

- void [Resize](#) (unsigned int width, unsigned int height, HWND windowHandle, bool isMSAAEnabled=false, bool isTextEnabled=false)

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

- void [CopyDataIntoDynamicBuffer](#) (unsigned int dynamicBufferKey, unsigned int index, const void \*data, UINT64 numOfBytes)

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

## 5.8.1 Detailed Description

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

## 5.8.2 Member Function Documentation

### 5.8.2.1 AfterRender()

```
void FAREnder::RenderScene::AfterRender ( )
```

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

Call after rendering all your objects and text.

### 5.8.2.2 AfterRenderObjects()

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

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

#### Parameters

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

### 5.8.2.3 AfterRenderText()

```
void FAREnder::RenderScene::AfterRenderText ( )
```

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

Call after calling all the RenderText functions.

#### 5.8.2.4 BeforeRenderObjects()

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

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

Call before calling the first RenderObjects function.

##### Parameters

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

#### 5.8.2.5 BeforeRenderText()

```
void FARender::RenderScene::BeforeRenderText ( )
```

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

#### 5.8.2.6 CompileShader()

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

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

##### Parameters

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

#### 5.8.2.7 CopyDataIntoDynamicBuffer()

```
void FARender::RenderScene::CopyDataIntoDynamicBuffer (
    unsigned int dynamicBufferKey,
    unsigned int index,
```



```
const void * data,
UINT64 numBytes )
```

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

#### Parameters

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

### 5.8.2.8 CreateDynamicBuffer()

```
void FAREnder::RenderScene::CreateDynamicBuffer (
    unsigned int bufferType,
    unsigned int dynamicBufferKey,
    unsigned numBytes,
    unsigned int stride = 0,
    DXGI_FORMAT format = DXGI_FORMAT_R32_UINT )
```

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

Make sure to pass in different keys to store the dynamic buffers with to prevent replacing the dynamic buffer at that key with the newly created dynamic buffer.

#### Parameters

in	<i>bufferType</i>	The type of buffer. Must be the values 0, 1, or 2. If it isn't one of those values a runtime_error exception is thrown. If 0 a dynamic vertex buffer is created. If 1 a dynamic index buffer is created. If 2 a dynamic constant buffer is created. If 1 pass in 0 for the stride.
in	<i>dynamicBufferKey</i>	The key to map the dynamic buffer to.
in	<i>numBytes</i>	The number of bytes to allocate for the dynamic buffer.
in	<i>stride</i>	The number of bytes to get from one element to the next element. Used for vertex and constant buffers.
	<i>[in,optional]</i>	format The number of bytes to get from one element to the next element. Used for index buffers.

### 5.8.2.9 CreateInputElementDescription()

```
void FAREnder::RenderScene::CreateInputElementDescription (
    unsigned int key,
    const char * semanticName,
    unsigned int semanticIndex,
    DXGI_FORMAT format,
    unsigned int inputSlot,
    unsigned int byteOffset,
```

```

D3D12_INPUT_CLASSIFICATION inputSlotClass = D3D12_INPUT_CLASSIFICATION_PER_↵
VERTEX_DATA,
    unsigned int instanceStepRate = 0 )

```

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

#### Parameters

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

#### 5.8.2.10 CreatePSO()

```

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

```

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

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

#### Parameters

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

### 5.8.2.11 CreateRootParameter()

```
void FARender::RenderScene::CreateRootParameter (
    unsigned int rootParameterKey,
    unsigned int shaderRegister )
```

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

#### Parameters

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

### 5.8.2.12 CreateRootSignature()

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

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

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

#### Parameters

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

### 5.8.2.13 CreateStaticBuffer()

```
void FARender::RenderScene::CreateStaticBuffer (
    unsigned int bufferType,
    unsigned int staticBufferKey,
    const void * data,
    unsigned numBytes,
    unsigned int stride = 0,
    DXGI_FORMAT format = DXGI_FORMAT_R32_UINT )
```

Creates a static buffer and stores the specified *data* in the buffer. The user cannot update/change the data once it is stored in the buffer.

Make sure to pass in different keys to store the static buffers with to prevent replacing the static buffer at that key with the newly created static buffer.

## Parameters

in	<i>bufferType</i>	The type of buffer. Must be the values 0 or 1. If it isn't one of those values a <code>runtime_error</code> exception is thrown. If 0 a static vertex buffer is created. If 1 a static index buffer is created. If 1 pass in 0 for the stride.
in	<i>staticBufferKey</i>	The key to map the dynamic buffer to.
in	<i>numBytes</i>	The number of bytes to allocate for the dynamic buffer.
in	<i>stride</i>	The number of bytes to get from one element to the next element. Used for vertex and constant buffers.
	<i>[in,optional]</i>	format The number of bytes to get from one element to the next element. Used for index buffers.

**5.8.2.14 ExecuteAndFlush()**

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

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

**5.8.2.15 LoadShader()**

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

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

## Parameters

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

**5.8.2.16 RemoveShader()**

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

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

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

### 5.8.2.17 RenderObject()

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

Renders an object with the specified draw arguments.

Call in between a BeforeRenderObjects function and a AfterRenderObjects function.

Ex.

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

#### Parameters

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

### 5.8.2.18 RenderText()

```
void FAREnder::RenderScene::RenderText (
    const FAMath::Vector4D & textLocation,
    const FIColor::Color & textColor,
    float textSize,
    const std::wstring & textString )
```

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

.

Ex.

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

Throws an out\_of\_range exception if the textKey is not mapped to a [Text](#) object.

#### Parameters

in	<i>textLocation</i>	The location of the text. The first 2 values are the top left corner and last two values are bottom right corner.
in	<i>textColor</i>	The color of the text.
in	<i>textSize</i>	The size of the size.
in	<i>textString</i>	The text to render.

### 5.8.2.19 Resize()

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

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

#### Parameters

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

### 5.8.2.20 SetDynamicBuffer()

```
void FARender::RenderScene::SetDynamicBuffer (
    unsigned int bufferType,
    unsigned int dynamicBufferKey,
    unsigned int indexConstantData = 0,
    unsigned int rootParameterIndex = 0 )
```

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

An out\_of\_range exception is thrown if the dynamic buffer key does not have a mapped dynamic buffer.

#### Parameters

in		
----	--	--

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

#### Parameters

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

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

#### 5.8.2.21 SetPSOAndRootSignature()

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

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

##### Parameters

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

#### 5.8.2.22 SetStaticBuffer()

```
void FARender::RenderScene::SetStaticBuffer (
    unsigned int bufferType,
    unsigned int staticBufferKey )
```

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

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

##### Parameters

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

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

- [FARenderScene.h](#)

## 5.9 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](#) ()  
*Resets 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.*

### 5.9.1 Detailed Description

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

### 5.9.2 Constructor & Destructor Documentation

#### 5.9.2.1 RenderTargetBuffer()

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

Default Constructor.

Parameters

in	<i>format</i>	The format of the render target buffer.
----	---------------	---

### 5.9.3 Member Function Documentation



### 5.9.3.1 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.

#### Parameters

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

### 5.9.3.2 CreateRenderTargetBufferAndView()

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

Creates the render target buffer and view.

#### Parameters

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

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

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

Returns a reference to the render target buffer.

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

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

Returns a constant reference to the render target buffer.

#### 5.9.3.5 GetRenderTargetFormat()

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

Returns the format of the render target buffer.

#### 5.9.3.6 ResetBuffer()

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

Resets the render target buffer.

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

- [FABuffer.h](#)

## 5.10 FARender::StaticBuffer Class Reference

This class stores data in a Direct3D 12 default buffer.

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

### Public Member Functions

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

### 5.10.1 Detailed Description

This class stores data in a Direct3D 12 default buffer.

### 5.10.2 Member Function Documentation

#### 5.10.2.1 CreateIndexBufferView()

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

Creates a index buffer view and stores it.

##### Parameters

in	<i>numBytes</i>	The number of bytes in the static buffer.
in	<i>format</i>	The number of bytes to get from one element to another in the static buffer.

#### 5.10.2.2 CreateStaticBuffer()

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

Creates the static buffer and stores all of the specified data.

##### Parameters

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

#### 5.10.2.3 CreateVertexBufferView()

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

Creates a vertex buffer view and stores it.

## Parameters

in	<i>numBytes</i>	The number of bytes in the static buffer.
in	<i>stride</i>	The number of bytes to get from one element to another in the static buffer.

**5.10.2.4 GetIndexBufferView()**

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

Returns a constant reference to the vertex buffer view.

**5.10.2.5 GetVertexBufferView()**

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

Returns a constant reference to the vertex buffer view.

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

- [FABuffer.h](#)

**5.11 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)  
*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.*

### 5.11.1 Detailed Description

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

### 5.11.2 Constructor & Destructor Documentation

#### 5.11.2.1 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 )
```

Creates a swap chain.

## Parameters

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

### 5.11.3 Member Function Documentation

#### 5.11.3.1 ClearCurrentBackBuffer()

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

Clears the current render target buffer.

## Parameters

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

#### 5.11.3.2 ClearDepthStencilBuffer()

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

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

## Parameters

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

## 5.11.3.3 CreateDepthStencilBufferAndView()

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

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

## Parameters

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

## 5.11.3.4 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.

## Parameters

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



#### 5.11.3.5 GetBackBufferFormat()

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

Returns the format of the swap chain.

#### 5.11.3.6 GetCurrentBackBuffer()

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

Returns a constant reference to the current render target buffer.

#### 5.11.3.7 GetCurrentBackBufferIndex()

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

Returns the current back buffer index.

#### 5.11.3.8 GetDepthStencilFormat()

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

Returns the format of the depth stencil buffer.

#### 5.11.3.9 GetNumRenderTargetBuffers()

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

Returns the number of swap chain buffers.

#### 5.11.3.10 GetRenderTargetBuffers()

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

Returns a constant pointer to the render target buffers.

### 5.11.3.11 Present()

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

Swaps the front and back buffers.

### 5.11.3.12 ResetBuffers()

```
void FAREnder::SwapChain::ResetBuffers ( )
```

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

### 5.11.3.13 ResizeSwapChain()

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

Resizes the swap chain.

#### Parameters

in	<i>width</i>	The width to resize the render target buffers to.
in	<i>height</i>	The height to resize the render target buffers to.

### 5.11.3.14 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.

#### Parameters

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

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

- FASwapChain.h

## 5.12 FAREnder::Text Class Reference

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

```
#include "FAText.h"
```

### Public Member Functions

- [Text](#) (const FAMath::Vector4D &textLocation, const std::wstring &textString, float textSize, const [FAColor::Color](#) &textColor)  
*Constructs a [Text](#) object.*
- const FAMath::Vector4D & [GetTextLocation](#) () const  
*Returns a constant reference to the text location.*
- const std::wstring & [GetTextString](#) () const  
*Returns a constant reference to the text string.*
- float [GetTextSize](#) () const  
*Returns the text size.*
- const [FAColor::Color](#) & [GetTextColor](#) () const  
*Returns a constant reference to the text color.*
- void [SetTextSize](#) (float textSize)  
*Changes the text size to the specified textSize.*
- void [SetTextColor](#) (const [FAColor::Color](#) &textColor)  
*Changes the text color to the specified textColor.*
- void [SetTextString](#) (const std::wstring &textString)  
*Changes the text string to the specified textString.*
- void [SetTextLocation](#) (const FAMath::Vector4D &textLocation)  
*Changes the text location to the specified textLocation.*

### 5.12.1 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.

### 5.12.2 Constructor & Destructor Documentation

#### 5.12.2.1 Text()

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

Constructs a [Text](#) object.

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

**Parameters**

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

### 5.12.3 Member Function Documentation

#### 5.12.3.1 GetTextColor()

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

Returns a constant reference to the text color.

#### 5.12.3.2 GetTextLocation()

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

Returns a constant reference to the text location.

#### 5.12.3.3 GetTextSize()

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

Returns the text size.

#### 5.12.3.4 GetTextString()

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

Returns a constant reference to the text string.

#### 5.12.3.5 SetTextColor()

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

Changes the text color to the specified *textColor*.

#### 5.12.3.6 SetTextLocation()

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

Changes the text location to the specified *textLocation*.

#### 5.12.3.7 SetTextSize()

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

Changes the text size to the specified *textSize*.

#### 5.12.3.8 SetTextString()

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

Changes the text string to the specified *textString*.

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

- [FAText.h](#)

## 5.13 FARender::TextResources Class Reference

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

```
#include "FATextResources.h"
```

## Public Member Functions

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

### 5.13.1 Detailed Description

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

### 5.13.2 Constructor & Destructor Documentation

#### 5.13.2.1 TextResources()

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

Initializes the text resources.

#### Parameters

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

### 5.13.3 Member Function Documentation

**5.13.3.1 AfterRenderText()**

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

Executes text commands.

**Parameters**

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

**5.13.3.2 BeforeRenderText()**

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

Prepares to render text.

**Parameters**

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

**5.13.3.3 GetDirect2DDeviceContext()**

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

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

**5.13.3.4 GetDirectWriteFactory()**

```
const Microsoft::WRL::ComPtr< IDWriteFactory > & FAREnder::TextResources::GetDirectWrite↵
Factory ( ) const
```

Returns a constant reference to the direct direct write factory.

**5.13.3.5 ResetBuffers()**

```
void FAREnder::TextResources::ResetBuffers ( )
```

Resets the text buffers.

### 5.13.3.6 ResizeBuffers()

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

Resizes the buffers.

#### Parameters

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

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

- FATextResources.h

## 5.14 FATime::Time Class Reference

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

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

### Public Member Functions

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

### 5.14.1 Detailed Description

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

### 5.14.2 Constructor & Destructor Documentation



### 5.14.2.1 Time()

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

Gets and stores the seconds per count.

## 5.14.3 Member Function Documentation

### 5.14.3.1 DeltaTime()

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

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

### 5.14.3.2 Reset()

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

Resets all time variables.

### 5.14.3.3 Start()

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

Starts the timer.

### 5.14.3.4 Stop()

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

Stops the timer.

### 5.14.3.5 Tick()

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

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

### 5.14.3.6 TotalTime()

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

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

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

- [FATime.h](#)

## 5.15 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.*
- [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.*
- 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 o the specified height.*

### 5.15.1 Detailed Description

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

### 5.15.2 Constructor & Destructor Documentation

#### 5.15.2.1 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.

## Parameters

in	<i>hInstance</i>	The handle to a module used to identify the executable.
in	<i>windowClassName</i>	The name of the window class.
in	<i>windowName</i>	The name of the window.
in	<i>winProcFunction</i>	The window procedure.
in	<i>width</i>	The width of the window.
in	<i>height</i>	The height of the window.
	<i>[in,optional]</i>	additionalData A pointer to data to access in the window procedure.

## 5.15.2.2 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.

## Parameters

in	<i>hInstance</i>	The handle to a module used to identify the executable.
in	<i>windowClass</i>	The window class for this window.
in	<i>windowName</i>	The name of the window.
in	<i>width</i>	The width of the window.
in	<i>height</i>	The height of the window.
	<i>[in,optional]</i>	additionalData A pointer to data to access in the window procedure.

## 5.15.3 Member Function Documentation

## 5.15.3.1 GetHeight()

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

Returns the height of the window.

#### 5.15.3.2 GetWidth()

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

Returns the width of the window.

#### 5.15.3.3 GetWindowHandle()

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

Returns the window handle.

#### 5.15.3.4 SetHeight()

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

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

#### 5.15.3.5 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 `RenderTargetBuffer`, `DepthStencilBuffer`, `StaticBuffer` and `DynamicBuffer` under namespace [FARender](#).

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

#### Classes

- class [FARender::RenderTargetBuffer](#)  
*A wrapper for render target buffer resources. Uses DirectD 12 API.*
- class [FARender::DepthStencilBuffer](#)  
*A wrapper for depth stencil buffer resources. Uses DirectD 12 API.*
- class [FARender::StaticBuffer](#)  
*This class stores data in a Direct3D 12 default buffer.*
- class [FARender::DynamicBuffer](#)  
*This class stores 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 `RenderTargetBuffer`, `DepthStencilBuffer`, `StaticBuffer` and `DynamicBuffer` 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 StaticBuffer
63     {
64     public:
65         StaticBuffer() = default;
66         StaticBuffer(const StaticBuffer&) = delete;
67         StaticBuffer& operator=(const StaticBuffer&) = delete;
68     };
69 }

```

```

142         StaticBuffer(StaticBuffer&&) = default;
143
144     void CreateStaticBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
145         const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const void* data, UINT
146         numBytes);
147
148     void CreateVertexBufferView(UINT numBytes, UINT stride);
149
150     void CreateIndexBufferView(UINT numBytes, DXGI_FORMAT format);
151
152     const D3D12_VERTEX_BUFFER_VIEW& GetVertexBufferView();
153
154     const D3D12_INDEX_BUFFER_VIEW& GetIndexBufferView();
155
156 private:
157     Microsoft::WRL::ComPtr<ID3D12Resource> mStaticDefaultBuffer;
158     Microsoft::WRL::ComPtr<ID3D12Resource> mStaticUploadBuffer;
159
160     union
161     {
162         D3D12_VERTEX_BUFFER_VIEW mVertexBufferView{};
163         D3D12_INDEX_BUFFER_VIEW mIndexBufferView;
164     };
165
166     class DynamicBuffer
167     {
168     public:
169         DynamicBuffer() = default;
170
171         DynamicBuffer(const DynamicBuffer&) = delete;
172         DynamicBuffer& operator=(const DynamicBuffer&) = delete;
173
174         DynamicBuffer(DynamicBuffer&&) = default;
175
176         ~DynamicBuffer();
177
178         D3D12_GPU_VIRTUAL_ADDRESS GetGPUAddress() const;
179
180         const unsigned int& GetStride() const;
181
182         const DXGI_FORMAT& GetFormat() const;
183
184         void CreateDynamicBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device, UINT numBytes,
185             UINT stride);
186
187         void CreateDynamicBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device, UINT numBytes,
188             DXGI_FORMAT format);
189
190         void CreateConstantBufferView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
191             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& cbvHeap, UINT cbvSize, UINT
192             cbvHeapIndex,
193             UINT cBufferIndex);
194
195         void CreateVertexBufferView(UINT numBytes);
196
197         void CreateIndexBufferView(UINT numBytes);
198
199         const D3D12_VERTEX_BUFFER_VIEW& GetVertexBufferView();
200
201         const D3D12_INDEX_BUFFER_VIEW& GetIndexBufferView();
202
203         void CopyData(UINT index, const void* data, UINT64 numBytes);
204
205     private:
206         Microsoft::WRL::ComPtr<ID3D12Resource> mDynamicBuffer;
207         BYTE* mMappedData{ nullptr };
208
209         union
210         {
211             UINT mStride;
212             DXGI_FORMAT mFormat;
213         };
214
215         union
216         {
217             D3D12_VERTEX_BUFFER_VIEW mVertexBufferView{};
218             D3D12_INDEX_BUFFER_VIEW mIndexBufferView;
219         };
220     };
221 }

```

## 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 {
25     class Camera
26     {
27     public:
28         Camera(vec3 cameraPosition = vec3(0.0f, 0.0f, 0.0f),
29             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,
30             1.0f),
31             float znear = 1.0f, float zfar = 100.f, float aspectRatio = 1.0f, float vFov = 45.0f,
32             float cameraVelocity = 10.0f, float angularVelocity = 0.25f);
33
34         const vec3& GetCameraPosition() const;
35
36         const vec3& GetX() const;
37
38         const vec3& GetY() const;
39
40         const vec3& GetZ() const;
41
42         const mat4& GetViewMatrix() const;
43
44         float GetCameraVelocity() const;
45
46         float GetAngularVelocity() const;
47
48         void LookAt(vec3 cameraPosition, vec3 target, vec3 up);
49
50         float GetZNear() const;
51
52         float GetZFar() const;
53
54         float GetVerticalFov() const;
55
56         float GetAspectRatio() const;
57
58         void SetCameraPosition(const vec3& position);
59
60         void SetX(const vec3& x);
61
62         void SetY(const vec3& y);
63
64         void SetZ(const vec3& z);
65
66         void SetCameraVelocity(float velocity);
67
68         void SetAngularVelocity(float velocity);
69
70         void SetZNear(float znear);
71
72         void SetZFar(float zfar);
73
74         void SetVerticalFov(float fov);
75
76         void SetAspectRatio(float ar);
77
78         const mat4& GetPerspectiveProjectionMatrix() const;
79
80         const mat4& GetViewPerspectiveProjectionMatrix() const;
81
82         void UpdateViewMatrix();
83
84         void UpdatePerspectiveProjectionMatrix();
85
86         void UpdateViewPerspectiveProjectionMatrix();
87
88         void Left(float dt);
89
90         void Right(float dt);
91
92         void Forward(float dt);
93
94         void Backward(float dt);

```

```

186
191     void Up(float dt);
192
197     void Down(float dt);
198
203     void RotateCameraLeftRight(float xDiff);
204
209     void RotateCameraUpDown(float yDiff);
210
219     void KeyboardInput(float dt);
220
223     void MouseInput();
224
225 private:
226     //camera position in world coordinates
227     vec3 mCameraPosition;
228
229     //z-axis of the camera coordinate system
230     vec3 mN;
231
232     //y-axis of the camera coordinate system
233     vec3 mV;
234
235     //x-axis of the camera coordinate system
236     vec3 mU;
237
238     //stores the world to camera transform
239     mat4 mViewMatrix;
240
241     //frustrum properties
242     float mNear;
243     float mFar;
244     float mVerticalFov;
245     float mAspectRatio;
246     mat4 mPerspectiveProjectionMatrix;
247
248     mat4 mViewPerspectiveProjectionMatrix;
249
250     float mCameraVelocity;
251     float mAngularVelocity;
252
253     vec2 mLastMousePosition;
254 };
255 }

```

## 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.*

### Namespaces

- namespace `FAColor`

*Has the `Color` class.*

## Functions

- Color `FIColor::operator+` (const Color &c1, const Color &c2)  
*Returns the result of  $c1 + c2$ .*
- Color `FIColor::operator-` (const Color &c1, const Color &c2)  
*Returns the result of  $c1 - c2$ .*
- Color `FIColor::operator*` (const Color &c, float k)  
*Returns the result of  $c * k$ .*
- Color `FIColor::operator*` (float k, const Color &c)  
*Returns the result of  $k * c$ .*
- Color `FIColor::operator*` (const Color &c1, const Color &c2)  
*Returns the result of  $c1 * c2$ .*

### 6.6.1 Detailed Description

File has class Color under namespace `FIColor`.

## 6.7 FIColor.h

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

```

1  #pragma once
2
3  #include "FIMathEngine.h"
4
12 namespace FIColor
13 {
14     class Color
15     {
16     public:
17
25         Color(float r = 0.0f, float g = 0.0f, float b = 0.0f, float a = 1.0f);
26
29         Color(const FIMath::Vector4D& color);
30
33         const FIMath::Vector4D& GetColor() const;
34
37         float GetRed() const;
38
41         float GetGreen() const;
42
45         float GetBlue() const;
46
49         float GetAlpha() const;
50
53         void SetColor(const FIMath::Vector4D& color);
54
57         void SetRed(float r);
58
61         void SetGreen(float g);
62
65         void SetBlue(float b);
66
69         void SetAlpha(float a);
70
75         Color& operator+=(const Color& c);
76
81         Color& operator-=(const Color& c);
82
88         Color& operator*=(float k);
89
95         Color& operator*=(const Color& c);
96
97     private:
98         FIMath::Vector4D mColor;
99     };
100
105     Color operator+(const Color& c1, const Color& c2);
106
111     Color operator-(const Color& c1, const Color& c2);

```

```

112
118     Color operator*(const Color& c, float k);
119
126     Color operator*(float k, const Color& c);
127
132     Color operator*(const Color& c1, const Color& c2);
133 }

```

## 6.8 FADeviceResources.h File Reference

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

```

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

```

### Classes

- class [FARender::DeviceResources](#)

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

### Namespaces

- namespace [FARender](#)

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

### 6.8.1 Detailed Description

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

## 6.9 FADeviceResources.h

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

```

1 #pragma once
2
7 #include <wrl.h>
8 #include <d3d12.h>
9 #include <dxgil_4.h>
10 #include "FASwapChain.h"
11 #include "FAMultiSampling.h"
12 #include "FATextResources.h"
13
14 namespace FASwapChain
15 {
16     class DeviceResources
17     {
18     public:
19
20         static const unsigned int NUM_OF_FRAMES{ 3 };
21
22         static DeviceResources& GetInstance(unsigned int width, unsigned int height, HWND windowHandle,
23         bool isMSAAEnabled, bool isTextEnabled);
24
25     };
26 }

```

```

42     DeviceResources(const DeviceResources&) = delete;
43     DeviceResources& operator=(const DeviceResources&) = delete;
44
45     ~DeviceResources();
46
47     const Microsoft::WRL::ComPtr<ID3D12Device>& GetDevice() const;
48
49     const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& GetCommandList() const;
50
51     DXGI_FORMAT GetBackBufferFormat() const;
52
53     DXGI_FORMAT GetDepthStencilFormat() const;
54
55     unsigned int GetCBVSize() const;
56
57     unsigned int GetCurrentFrame() const;
58
59     const TextResources& GetTextResources() const;
60
61     void UpdateCurrentFrameFenceValue();
62
63     void FlushCommandQueue();
64
65     void WaitForGPU() const;
66
67     void Signal();
68
69     void Resize(int width, int height, const HWND& handle, bool isMSAAEnabled, bool isTextEnabled);
70
71     void RTBufferTransition(bool isMSAAEnabled, bool isTextEnabled);
72
73     void BeforeTextDraw();
74
75     void AfterTextDraw();
76
77     void Execute() const;
78
79     void Present();
80
81     /*@brief Calls the necessary functions to let the user draw their objects.
82     *
83     * @param[in] isMSAAEnabled Pass in true if MSAA enabled, false otherwise.
84     */
85     void Draw(bool isMSAAEnabled);
86
87     void NextFrame();
88
89 private:
90
91     DeviceResources(unsigned int width, unsigned int height, HWND windowHandle,
92                     bool isMSAAEnabled, bool isTextEnabled);
93
94     unsigned int mCurrentFrameIndex;
95
96     Microsoft::WRL::ComPtr<ID3D12Device> mDirect3DDevice;
97
98     Microsoft::WRL::ComPtr<IDXGIFactory4> mDXGIFactory;
99
100    Microsoft::WRL::ComPtr<ID3D12Fence> mFence;
101    UINT64 mFenceValue;
102    UINT64 mCurrentFrameFenceValue[NUM_OF_FRAMES];
103
104    Microsoft::WRL::ComPtr<ID3D12CommandQueue> mCommandQueue;
105    Microsoft::WRL::ComPtr<ID3D12CommandAllocator> mCommandAllocators[NUM_OF_FRAMES];
106    Microsoft::WRL::ComPtr<ID3D12CommandAllocator> mDirectCommandAllocator;
107    Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList> mCommandList;
108
109    UINT mRTVSize;
110    UINT mDSVSize;
111    UINT mCBVSize;
112
113    Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mRTVHeap;
114    Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mDSVHeap;
115
116    SwapChain mSwapChain;
117
118    MultiSampling mMultiSampling;
119
120    D3D12_VIEWPORT mViewport{};
121    D3D12_RECT mScissor{};
122
123    TextResources mTextResources;
124
125    //Call all of these functions to initialize Direct3D
126    void mEnableDebugLayer();
127    void mCreateDirect3DDevice();
128    void mCreateDXGIFactory();

```

```

195         void mCreateFence();
196         void mQueryDescriptorSizes();
197         void mCreateRTVHeap();
198         void mCreateDSVHeap();
199         void mCreateCommandObjects();
200     };
201 }

```

## 6.10 FADirectXException.h

```

1 #pragma once
2
3 #include <wrl.h>
4 #include <dxgidebug.h>
5 #include <comdef.h>
6 #include <string>
7 #include <sstream>
8 #include <vector>
9
10 inline std::wstring AnsiToWString(const std::string& str)
11 {
12     WCHAR buffer[1024];
13     MultiByteToWideChar(CP_ACP, 0, str.c_str(), -1, buffer, 1024);
14     return std::wstring(buffer);
15 }
16
17 class DirectXException
18 {
19 public:
20     DirectXException(HRESULT hr, const std::wstring& functionName, const std::wstring& fileName, int
        lineNumber);
21
22     std::wstring ErrorMessage() const;
23
24 private:
25     HRESULT errorCode;
26     std::wstring functionName;
27     std::wstring fileName;
28     int lineNumber;
29     Microsoft::WRL::ComPtr<IDXGIInfoQueue> mInfoQueue;
30 };
31
32 #ifndef ThrowIfFailed
33 #define ThrowIfFailed(x) \
34 { \
35     HRESULT hr = (x); \
36     std::wstring filename(AnsiToWString(__FILE__)); \
37     if (FAILED(hr)) { throw DirectXException(hr, L#x, filename, __LINE__); } \
38 } \
39 #endif

```

## 6.11 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         const Microsoft::WRL::ComPtr<ID3D12Resource>& GetRenderTargetBuffer();
17
18         DXGI_FORMAT GetRenderTargetFormat();
19
20         DXGI_FORMAT GetDepthStencilFormat();
21
22         void ResetBuffers();
23     };
24 }

```

```

51     void CreateRenderTargetBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
52     const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int
indexOfWhereToStoreView, unsigned int rtvSize,
53     unsigned int width, unsigned int height);
54
55     void CreateDepthStencilBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
56     const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int
indexOfWhereToStoreView, unsigned int dsvSize,
57     unsigned int width, unsigned int height);
58
59     void Transition(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList,
60     D3D12_RESOURCE_STATES before, D3D12_RESOURCE_STATES after);
61
62     void ClearRenderTargetBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
commandList,
63     const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int indexOfView,
64     unsigned int rtvSize,
65     const float* clearValue);
66
67     void ClearDepthStencilBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
commandList,
68     const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int indexOfView,
69     unsigned int dsvSize,
70     float clearValue);
71
72     private:
73     RenderTargetBuffer mMSAARenderTargetBuffer;
74     DepthStencilBuffer mMSAADepthStencilBuffer;
75     unsigned int mSampleCount;
76 };
77 }

```

## 6.12 FARenderScene.h File Reference

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

```

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

```

### Classes

- class [FARender::RenderScene](#)  
*This class is used to render a scene using Direct3D 12 API.*

### Namespaces

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

#### 6.12.1 Detailed Description

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

## 6.13 FARenderScene.h

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

```

1 #pragma once
2
3 #include <d3dcompiler.h>
4 #include <unordered_map>
5 #include "FADeviceResources.h"
6 #include "FABuffer.h"
7 #include "FAColor.h"
8
9 namespace FARender
10 {
11     class RenderScene
12     {
13     public:
14
15         /*@brief Initializes all necessary resources.
16
17         * @param[in] width The width of a window.
18         * @param[in] height The height of a window.
19         * @param[in] windowHandle A handle to a window.
20         * @param[in, optional] isMSAAEnabled Pass in true if you want to have MSAA enabled for the initial frame,
21           false otherwise.
22         * @param[in, optional] isTextEnabled Pass in true if you want to have text enabled for the initial frame,
23           false otherwise.
24
25         */
26         RenderScene(unsigned int width, unsigned int height, HWND windowHandle, bool isMSAAEnabled =
27           false, bool isTextEnabled = false);
28
29         RenderScene(const RenderScene&) = delete;
30         RenderScene& operator=(const RenderScene&) = delete;
31
32         RenderScene(RenderScene&&) = default;
33
34         void LoadShader(unsigned int shaderKey, const std::wstring& filename);
35
36         void CompileShader(unsigned int shaderKey, const std::wstring& filename,
37           const std::string& entryPointName, const std::string& target);
38
39         void RemoveShader(unsigned int shaderKey);
40
41         void CreateStaticBuffer(unsigned int bufferType, unsigned int staticBufferKey,
42           const void* data, unsigned numBytes, unsigned int stride = 0, DXGI_FORMAT format =
43           DXGI_FORMAT_R32_UINT);
44
45         void CreateDynamicBuffer(unsigned int bufferType, unsigned int dynamicBufferKey,
46           unsigned numBytes, unsigned int stride = 0, DXGI_FORMAT format = DXGI_FORMAT_R32_UINT);
47
48         void CreateInputElementDescription(unsigned int key, const char* semanticName, unsigned int
49           semanticIndex,
50           DXGI_FORMAT format, unsigned int inputSlot, unsigned int byteOffset,
51           D3D12_INPUT_CLASSIFICATION inputSlotClass = D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA,
52           unsigned int instanceStepRate = 0);
53
54         void CreateRootParameter(unsigned int rootParameterKey, unsigned int shaderRegister);
55
56         void CreateRootSignature(unsigned int rootSigKey, unsigned int rootParametersKey);
57
58         void CreatePSO(unsigned int psKey, D3D12_FILL_MODE fillMode, BOOL enableMultisample,
59           unsigned int vsKey, unsigned int psKey, unsigned int inputElementDescriptionsKey,
60           unsigned int rootSigKey,
61           const D3D12_PRIMITIVE_TOPOLOGY_TYPE& primitiveType, UINT sampleCount = 1);
62
63         void SetPSOAndRootSignature(unsigned int psKey, unsigned int rootSigKey);
64
65         void SetStaticBuffer(unsigned int bufferType, unsigned int staticBufferKey);
66
67         void SetDynamicBuffer(unsigned int bufferType, unsigned int dynamicBufferKey, unsigned int
68           indexConstantData = 0,
69           unsigned int rootParameterIndex = 0);
70
71         void BeforeRenderObjects(bool isMSAAEnabled = false);
72
73         void RenderObject(unsigned int indexCount, unsigned int locationFirstIndex, int
74           indexOffsetFirstVertex,
75           D3D_PRIMITIVE_TOPOLOGY primitive);
76
77         void AfterRenderObjects(bool isMSAAEnabled = false, bool isTextEnabled = false);
78
79         void BeforeRenderText();
80
81         void RenderText(const FAMath::Vector4D& textLocation, const FAColor::Color& textColor, float
82           textSize,
83           const std::wstring& textString);
84     };
85 }
```



```

257
262     void AfterRenderText();
263
268     void AfterRender();
269
272     void ExecuteAndFlush();
273
282     void Resize(unsigned int width, unsigned int height, HWND windowHandle, bool isMSAAEnabled =
false, bool isTextEnabled = false);
283
291     void CopyDataIntoDynamicBuffer(unsigned int dynamicBufferKey, unsigned int index, const void*
data, UINT64 numBytes);
292
293     private:
294         //The device resources object that all RenderScene objects share.
295         DeviceResources& mDeviceResources;
296
297         //Stores all of the shaders for this scene.
298         std::unordered_map<unsigned int, Microsoft::WRL::ComPtr<ID3DBlob>> mShaders;
299
300         //Stores input element descriptions for a set of shaders.
301         std::unordered_map<unsigned int, std::vector<D3D12_INPUT_ELEMENT_DESC>
mInputElementDescriptions;
302
303         //Stores root parameters for root signatures.
304         std::unordered_map<unsigned int, std::vector<D3D12_ROOT_PARAMETER>> mRootParameters;
305
306         //The root signatures for the scene.
307         //Describes all of the constant data that is expected in a set of shaders.
308         //Microsoft::WRL::ComPtr<ID3D12RootSignature> mRootSignature;
309         std::unordered_map<unsigned int, Microsoft::WRL::ComPtr<ID3D12RootSignature>> mRootSignatures;
310
311         //Stores pipeline state objects.
312         std::unordered_map<unsigned int, Microsoft::WRL::ComPtr<ID3D12PipelineState>> mPSOs;
313
314         //Stores data that will not be updated on a per-frame basis.
315         std::unordered_map<unsigned int, StaticBuffer> mStaticBuffers;
316
317         //Stores data that will be updated on a per-frame basis.
318         //We can't update a dynamic buffer until the GPU
319         //is done executing all the commands that reference it, so each frame needs its own dynamic
buffer.
320         std::unordered_map<unsigned int, std::vector<DynamicBuffer>> mDynamicBuffers;
321     };
322 }

```

## 6.14 FASwapChain.h

```

1 #pragma once
2
3 #include <wrl.h>
4 #include "d3dx12.h"
5 #include <dxgi1_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 =
DXGI_FORMAT_D24_UNORM_S8_UINT,
20             unsigned int numRenderTargetBuffers = 2);
21
22         const RenderTargetBuffer* GetRenderTargetBuffers() const;
23
24         const Microsoft::WRL::ComPtr<ID3D12Resource>& GetCurrentBackBuffer() const;
25
26         unsigned int GetNumRenderTargetBuffers() const;
27
28         unsigned int GetCurrentBackBufferIndex() const;
29
30         DXGI_FORMAT GetBackBufferFormat() const;
31
32         DXGI_FORMAT GetDepthStencilFormat() const;
33
34         void ResetBuffers();

```

```

61
62     void ResizeSwapChain(unsigned width, unsigned height);
63
64     void CreateRenderTargetBuffersAndViews(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
65     const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int
66     indexWhereToStoreFirstView,
67     unsigned int rtvSize);
68
69     void CreateDepthStencilBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
70     const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int index, unsigned int
71     dsvSize,
72     unsigned int width, unsigned int height);
73
74     void ClearCurrentBackBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
75     commandList,
76     const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int indexOfFirstView,
77     unsigned int rtvSize,
78     const float* backBufferClearValue);
79
80     void ClearDepthStencilBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
81     commandList,
82     const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int indexOfView,
83     unsigned int dsvSize,
84     float clearValue);
85
86     void Transition(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList,
87     D3D12_RESOURCE_STATES before, D3D12_RESOURCE_STATES after);
88
89     void Present();
90
91     private:
92     unsigned int mNumRenderTargetBuffers = 0;
93     unsigned int mCurrentBackBufferIndex = 0;
94
95     Microsoft::WRL::ComPtr<IDXGISwapChain1> mSwapChain;
96     std::vector<RenderTargetBuffer> mRenderTargetBuffers;
97
98     DepthStencilBuffer mDepthStencilBuffer;
99 };
100
101 }
```

## 6.15 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.*

#### 6.15.1 Detailed Description

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

## 6.16 FAText.h

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

```

1  #pragma once
2
3  #include <string>
4  #include "FAColor.h"
5
6  namespace FASystem
7  {
8      namespace FASystem
9      {
10         namespace FASystem
11         {
12             class Text
13             {
14             public:
15
16                 Text() = default;
17
18                 Text(const FAMath::Vector4D& textLocation, const std::wstring& textString, float textSize, const
19                     FAColor::Color& textColor);
20
21                 const FAMath::Vector4D& GetTextLocation() const;
22
23                 const std::wstring& GetTextString() const;
24
25                 float GetTextSize() const;
26
27                 const FAColor::Color& GetTextColor() const;
28
29                 void SetTextSize(float textSize);
30
31                 void SetTextColor(const FAColor::Color& textColor);
32
33                 void SetTextString(const std::wstring& textString);
34
35                 void SetTextLocation(const FAMath::Vector4D& textLocation);
36
37             private:
38
39                 FAMath::Vector4D mTextLocation;
40                 std::wstring mText;
41                 float mTextSize{ 0.0f };
42                 FAColor::Color mTextColor;
43             };
44         }
45     }
46 }

```

## 6.17 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 FASystem
12 {
13     class TextResources
14     {
15     public:
16
17         TextResources() = default;
18
19         TextResources(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
20             const Microsoft::WRL::ComPtr<ID3D12CommandQueue>& commandQueue, unsigned int
21             numSwapChainBuffers);
22
23         const Microsoft::WRL::ComPtr<ID2D1DeviceContext>& GetDirect2DDeviceContext() const;
24
25         const Microsoft::WRL::ComPtr<IDWriteFactory>& GetDirectWriteFactory() const;
26
27         void ResetBuffers();
28
29         void ResizeBuffers(const RenderTargetBuffer* renderTargetBuffers, HWND windowHandle);
30
31         void BeforeRenderText(unsigned int currentBackBuffer);
32
33         void AfterRenderText(unsigned int currentBackBuffer);
34
35     private:
36
37         Microsoft::WRL::ComPtr<ID3D11Device> mDevice11;
38     };
39 }

```

```

63     Microsoft::WRL::ComPtr<ID3D11DeviceContext> mDevice11Context;
64     Microsoft::WRL::ComPtr<ID3D11On12Device> mDevice11on12;
65
66     Microsoft::WRL::ComPtr<ID2D1Device2> mDirect2DDevice;
67     Microsoft::WRL::ComPtr<ID2D1Factory3> mDirect2DFactory;
68     Microsoft::WRL::ComPtr<ID2D1DeviceContext> mDirect2DDeviceContext;
69
70     Microsoft::WRL::ComPtr<IDWriteFactory> mDirectWriteFactory;
71
72     std::vector<Microsoft::WRL::ComPtr<ID3D11Resource> > mWrappedBuffers;
73     std::vector<Microsoft::WRL::ComPtr<ID2D1Bitmap1> > mDirect2DBuffers;
74     std::vector<Microsoft::WRL::ComPtr<IDXGISurface> > mSurfaces;
75 };
76 }

```

## 6.18 FATime.h File Reference

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

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

### Classes

- class [FATime::Time](#)

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

### Namespaces

- namespace [FATime](#)

*Has [Time](#) class.*

### 6.18.1 Detailed Description

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

## 6.19 FATime.h

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

```

1  #pragma once
2
3  #include <Windows.h>
4
5  namespace FATime
6  {
7      class Time
8      {
9      public:
10         Time();
11
12         void Tick();
13
14         float DeltaTime() const;
15
16         void Reset();
17
18         void Stop();
19
20         void Start();

```

```

43
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 / resetted
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 }
60
61

```

## 6.20 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.

#### 6.20.1 Detailed Description

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

## 6.21 FAWindow.h

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

```

1 #pragma once
2
3 #include <Windows.h>
4 #include <string>
5 #include <stdexcept>
6
7 namespace FAWindow
8 {
9     class Window
10     {
11     public:
12         Window() = default;
13
14     };
15 }

```

```
35     Window(const HINSTANCE& hInstance, const std::wstring& windowClassName, const std::wstring&
windowName,
36     WNDPROC winProcFunction, unsigned int width, unsigned int height, void* additionalData =
nullptr);
37
47     Window(const HINSTANCE& hInstance, const WNDCLASSEX& windowClass, const std::wstring& windowName,
48     unsigned int width, unsigned int height, void* additionalData = nullptr);
49
52     HWND GetWindowHandle() const;
53
56     unsigned int GetWidth() const ;
57
60     unsigned int GetHeight() const;
61
64     void SetWidth(unsigned int width);
65
68     void SetHeight(unsigned int height);
69
70 private:
71     HWND mWindowHandle;
72
73     WNDCLASSEX mWindowClass;
74     std::wstring mWindowClassName;
75
76     unsigned int mWidth;
77     unsigned int mHeight;
78 };
79 }
```

# Index

- ~DeviceResources
  - FARender::DeviceResources, [28](#)
- ~DynamicBuffer
  - FARender::DynamicBuffer, [35](#)
- AfterRender
  - FARender::RenderScene, [45](#)
- AfterRenderObjects
  - FARender::RenderScene, [45](#)
- AfterRenderText
  - FARender::RenderScene, [45](#)
  - FARender::TextResources, [68](#)
- AfterTextDraw
  - FARender::DeviceResources, [29](#)
- Backward
  - FACamera::Camera, [14](#)
- BeforeRenderObjects
  - FARender::RenderScene, [45](#)
- BeforeRenderText
  - FARender::RenderScene, [46](#)
  - FARender::TextResources, [69](#)
- BeforeTextDraw
  - FARender::DeviceResources, [29](#)
- Camera
  - FACamera::Camera, [13](#)
- ClearCurrentBackBuffer
  - FARender::SwapChain, [61](#)
- ClearDepthStencilBuffer
  - FARender::DepthStencilBuffer, [26](#)
  - FARender::MultiSampling, [40](#)
  - FARender::SwapChain, [61](#)
- ClearRenderTargetBuffer
  - FARender::MultiSampling, [41](#)
  - FARender::RenderTargetBuffer, [54](#)
- Color
  - FAColor::Color, [22](#)
- CompileShader
  - FARender::RenderScene, [46](#)
- CopyData
  - FARender::DynamicBuffer, [35](#)
- CopyDataIntoDynamicBuffer
  - FARender::RenderScene, [46](#)
- CreateConstantBufferView
  - FARender::DynamicBuffer, [36](#)
- CreateDepthStencilBufferAndView
  - FARender::DepthStencilBuffer, [26](#)
  - FARender::MultiSampling, [41](#)
  - FARender::SwapChain, [62](#)
- CreateDynamicBuffer
  - FARender::DynamicBuffer, [36](#)
  - FARender::RenderScene, [47](#)
- CreateIndexBufferView
  - FARender::DynamicBuffer, [38](#)
  - FARender::StaticBuffer, [57](#)
- CreateInputElementDescription
  - FARender::RenderScene, [47](#)
- CreatePSO
  - FARender::RenderScene, [48](#)
- CreateRenderTargetBufferAndView
  - FARender::MultiSampling, [42](#)
  - FARender::RenderTargetBuffer, [55](#)
- CreateRenderTargetBuffersAndViews
  - FARender::SwapChain, [62](#)
- CreateRootParameter
  - FARender::RenderScene, [49](#)
- CreateRootSignature
  - FARender::RenderScene, [49](#)
- CreateStaticBuffer
  - FARender::RenderScene, [49](#)
  - FARender::StaticBuffer, [57](#)
- CreateVertexBufferView
  - FARender::DynamicBuffer, [38](#)
  - FARender::StaticBuffer, [57](#)
- DeltaTime
  - FATime::Time, [71](#)
- DepthStencilBuffer
  - FARender::DepthStencilBuffer, [25](#)
- DirectXException, [33](#)
  - DirectXException, [33](#)
  - ErrorMsg, [34](#)
- Down
  - FACamera::Camera, [14](#)
- ErrorMsg
  - DirectXException, [34](#)
- Execute
  - FARender::DeviceResources, [29](#)
- ExecuteAndFlush
  - FARender::RenderScene, [50](#)
- FABuffer.h, [75](#)
- FACamera, [7](#)
- FACamera.h, [78](#)
  - vec2, [78](#)
- FACamera::Camera, [11](#)
  - Backward, [14](#)
  - Camera, [13](#)

- Down, [14](#)
- Foward, [14](#)
- GetAngularVelocity, [14](#)
- GetAspectRatio, [15](#)
- GetCameraPosition, [15](#)
- GetCameraVelocity, [15](#)
- GetPerspectiveProjectionMatrix, [15](#)
- GetVerticalFov, [15](#)
- GetViewMatrix, [15](#)
- GetViewPerspectiveProjectionMatrix, [15](#)
- GetX, [16](#)
- GetY, [16](#)
- GetZ, [16](#)
- GetZFar, [16](#)
- GetZNear, [16](#)
- KeyboardInput, [16](#)
- Left, [17](#)
- LookAt, [17](#)
- MouseInput, [17](#)
- Right, [17](#)
- RotateCameraLeftRight, [18](#)
- RotateCameraUpDown, [18](#)
- SetAngularVelocity, [18](#)
- SetAspectRatio, [18](#)
- SetCameraPosition, [18](#)
- SetCameraVelocity, [19](#)
- SetVerticalFov, [19](#)
- SetX, [19](#)
- SetY, [19](#)
- SetZ, [19](#)
- SetZFar, [19](#)
- SetZNear, [20](#)
- Up, [20](#)
- UpdatePerspectiveProjectionMatrix, [20](#)
- UpdateViewMatrix, [20](#)
- UpdateViewPerspectiveProjectionMatrix, [20](#)
- FAColor, [7](#)
  - operator\*, [8](#)
  - operator+, [9](#)
  - operator-, [9](#)
- FAColor.h, [80](#)
- FAColor::Color, [21](#)
  - Color, [22](#)
  - GetAlpha, [22](#)
  - GetBlue, [22](#)
  - GetColor, [22](#)
  - GetGreen, [23](#)
  - GetRed, [23](#)
  - operator\*=[23](#)
  - operator+=[23](#)
  - operator-=[23](#)
  - SetAlpha, [24](#)
  - SetBlue, [24](#)
  - SetColor, [24](#)
  - SetGreen, [24](#)
  - SetRed, [24](#)
- FADeviceResources.h, [82](#)
- FARender, [9](#)
  - FARender::DepthStencilBuffer, [25](#)
    - ClearDepthStencilBuffer, [26](#)
    - CreateDepthStencilBufferAndView, [26](#)
    - DepthStencilBuffer, [25](#)
    - GetDepthStencilFormat, [27](#)
    - ResetBuffer, [27](#)
  - FARender::DeviceResources, [27](#)
    - ~DeviceResources, [28](#)
    - AfterTextDraw, [29](#)
    - BeforeTextDraw, [29](#)
    - Execute, [29](#)
    - FlushCommandQueue, [29](#)
    - GetBackBufferFormat, [29](#)
    - GetCBVSize, [29](#)
    - GetCommandList, [30](#)
    - GetCurrentFrame, [30](#)
    - GetDepthStencilFormat, [30](#)
    - GetDevice, [30](#)
    - GetInstance, [30](#)
    - GetTextResources, [31](#)
    - NextFrame, [31](#)
    - NUM\_OF\_FRAMES, [33](#)
    - Present, [31](#)
    - Resize, [31](#)
    - RTBufferTransition, [32](#)
    - Signal, [32](#)
    - UpdateCurrentFrameFenceValue, [32](#)
    - WaitForGPU, [32](#)
  - FARender::DynamicBuffer, [34](#)
    - ~DynamicBuffer, [35](#)
    - CopyData, [35](#)
    - CreateConstantBufferView, [36](#)
    - CreateDynamicBuffer, [36](#)
    - CreateIndexBufferView, [38](#)
    - CreateVertexBufferView, [38](#)
    - GetFormat, [38](#)
    - GetGPUAddress, [38](#)
    - GetIndexBufferView, [38](#)
    - GetStride, [39](#)
    - GetVertexBufferView, [39](#)
  - FARender::MultiSampling, [39](#)
    - ClearDepthStencilBuffer, [40](#)
    - ClearRenderTargetBuffer, [41](#)
    - CreateDepthStencilBufferAndView, [41](#)
    - CreateRenderTargetBufferAndView, [42](#)
    - GetRenderTargetBuffer, [42](#)
    - MultiSampling, [40](#)
    - ResetBuffers, [42](#)
    - Transition, [43](#)
  - FARender::RenderScene, [43](#)
    - AfterRender, [45](#)
    - AfterRenderObjects, [45](#)
    - AfterRenderText, [45](#)
    - BeforeRenderObjects, [45](#)
    - BeforeRenderText, [46](#)
    - CompileShader, [46](#)
    - CopyDataIntoDynamicBuffer, [46](#)
    - CreateDynamicBuffer, [47](#)



- CreateInputElementDescription, 47
  - CreatePSO, 48
  - CreateRootParameter, 49
  - CreateRootSignature, 49
  - CreateStaticBuffer, 49
  - ExecuteAndFlush, 50
  - LoadShader, 50
  - RemoveShader, 50
  - RenderObject, 50
  - RenderText, 51
  - Resize, 52
  - SetDynamicBuffer, 52
  - SetPSOAndRootSignature, 53
  - SetStaticBuffer, 53
- FARender::RenderTargetBuffer, 53
  - ClearRenderTargetBuffer, 54
  - CreateRenderTargetBufferAndView, 55
  - GetRenderTargetBuffer, 55, 56
  - GetRenderTargetFormat, 56
  - RenderTargetBuffer, 54
  - ResetBuffer, 56
- FARender::StaticBuffer, 56
  - CreateIndexBufferView, 57
  - CreateStaticBuffer, 57
  - CreateVertexBufferView, 57
  - GetIndexBufferView, 59
  - GetVertexBufferView, 59
- FARender::SwapChain, 59
  - ClearCurrentBackBuffer, 61
  - ClearDepthStencilBuffer, 61
  - CreateDepthStencilBufferAndView, 62
  - CreateRenderTargetBuffersAndViews, 62
  - GetBackBufferFormat, 63
  - GetCurrentBackBuffer, 63
  - GetCurrentBackBufferIndex, 63
  - GetDepthStencilFormat, 63
  - GetNumRenderTargetBuffers, 63
  - GetRenderTargetBuffers, 63
  - Present, 63
  - ResetBuffers, 64
  - ResizeSwapChain, 64
  - SwapChain, 60
  - Transition, 64
- FARender::Text, 65
  - GetTextColor, 66
  - GetTextLocation, 66
  - GetTextSize, 66
  - GetTextString, 66
  - SetTextColor, 66
  - SetTextLocation, 67
  - SetTextSize, 67
  - SetTextString, 67
  - Text, 65
- FARender::TextResources, 67
  - AfterRenderText, 68
  - BeforeRenderText, 69
  - GetDirect2DDeviceContext, 69
  - GetDirectWriteFactory, 69
  - ResetBuffers, 69
  - ResizeBuffers, 69
  - TextResources, 68
- FARenderScene.h, 85
- FAText.h, 88
- FATime, 10
- FATime.h, 90
- FATime::Time, 70
  - DeltaTime, 71
  - Reset, 71
  - Start, 71
  - Stop, 71
  - Tick, 71
  - Time, 70
  - TotalTime, 71
- FAWindow, 10
- FAWindow.h, 91
- FAWindow::Window, 72
  - GetHeight, 73
  - GetWidth, 73
  - GetWindowHandle, 74
  - SetHeight, 74
  - SetWidth, 74
  - Window, 72, 73
- FlushCommandQueue
  - FARender::DeviceResources, 29
- Foward
  - FACamera::Camera, 14
- GetAlpha
  - FAColor::Color, 22
- GetAngularVelocity
  - FACamera::Camera, 14
- GetAspectRatio
  - FACamera::Camera, 15
- GetBackBufferFormat
  - FARender::DeviceResources, 29
  - FARender::SwapChain, 63
- GetBlue
  - FAColor::Color, 22
- GetCameraPosition
  - FACamera::Camera, 15
- GetCameraVelocity
  - FACamera::Camera, 15
- GetCBVSize
  - FARender::DeviceResources, 29
- GetColor
  - FAColor::Color, 22
- GetCommandList
  - FARender::DeviceResources, 30
- GetCurrentBackBuffer
  - FARender::SwapChain, 63
- GetCurrentBackBufferIndex
  - FARender::SwapChain, 63
- GetCurrentFrame
  - FARender::DeviceResources, 30
- GetDepthStencilFormat
  - FARender::DepthStencilBuffer, 27
  - FARender::DeviceResources, 30

- FARender::SwapChain, 63
- GetDevice
  - FARender::DeviceResources, 30
- GetDirect2DDeviceContext
  - FARender::TextResources, 69
- GetDirectWriteFactory
  - FARender::TextResources, 69
- GetFormat
  - FARender::DynamicBuffer, 38
- GetGPUAddress
  - FARender::DynamicBuffer, 38
- GetGreen
  - FAColor::Color, 23
- GetHeight
  - FAWindow::Window, 73
- GetIndexBufferView
  - FARender::DynamicBuffer, 38
  - FARender::StaticBuffer, 59
- GetInstance
  - FARender::DeviceResources, 30
- GetNumRenderTargetBuffers
  - FARender::SwapChain, 63
- GetPerspectiveProjectionMatrix
  - FACamera::Camera, 15
- GetRed
  - FAColor::Color, 23
- GetRenderTargetBuffer
  - FARender::MultiSampling, 42
  - FARender::RenderTargetBuffer, 55, 56
- GetRenderTargetBuffers
  - FARender::SwapChain, 63
- GetRenderTargetFormat
  - FARender::RenderTargetBuffer, 56
- GetStride
  - FARender::DynamicBuffer, 39
- GetTextColor
  - FARender::Text, 66
- GetTextLocation
  - FARender::Text, 66
- GetTextResources
  - FARender::DeviceResources, 31
- GetTextSize
  - FARender::Text, 66
- GetTextString
  - FARender::Text, 66
- GetVertexBufferView
  - FARender::DynamicBuffer, 39
  - FARender::StaticBuffer, 59
- GetVerticalFov
  - FACamera::Camera, 15
- GetViewMatrix
  - FACamera::Camera, 15
- GetViewPerspectiveProjectionMatrix
  - FACamera::Camera, 15
- GetWidth
  - FAWindow::Window, 73
- GetWindowHandle
  - FAWindow::Window, 74
- GetX
  - FACamera::Camera, 16
- GetY
  - FACamera::Camera, 16
- GetZ
  - FACamera::Camera, 16
- GetZFar
  - FACamera::Camera, 16
- GetZNear
  - FACamera::Camera, 16
- KeyboardInput
  - FACamera::Camera, 16
- Left
  - FACamera::Camera, 17
- LoadShader
  - FARender::RenderScene, 50
- LookAt
  - FACamera::Camera, 17
- MouseInput
  - FACamera::Camera, 17
- MultiSampling
  - FARender::MultiSampling, 40
- NextFrame
  - FARender::DeviceResources, 31
- NUM\_OF\_FRAMES
  - FARender::DeviceResources, 33
- operator\*
  - FAColor, 8
- operator\*=
  - FAColor::Color, 23
- operator+
  - FAColor, 9
- operator+=
  - FAColor::Color, 23
- operator-
  - FAColor, 9
- operator-=
  - FAColor::Color, 23
- Present
  - FARender::DeviceResources, 31
  - FARender::SwapChain, 63
- RemoveShader
  - FARender::RenderScene, 50
- RenderObject
  - FARender::RenderScene, 50
- RenderTargetBuffer
  - FARender::RenderTargetBuffer, 54
- RenderText
  - FARender::RenderScene, 51
- Reset
  - FATime::Time, 71
- ResetBuffer
  - FARender::DepthStencilBuffer, 27

- FARender::RenderTargetBuffer, 56
- ResetBuffers
  - FARender::MultiSampling, 42
  - FARender::SwapChain, 64
  - FARender::TextResources, 69
- Resize
  - FARender::DeviceResources, 31
  - FARender::RenderScene, 52
- ResizeBuffers
  - FARender::TextResources, 69
- ResizeSwapChain
  - FARender::SwapChain, 64
- Right
  - FACamera::Camera, 17
- RotateCameraLeftRight
  - FACamera::Camera, 18
- RotateCameraUpDown
  - FACamera::Camera, 18
- RTBufferTransition
  - FARender::DeviceResources, 32
- SetAlpha
  - FAColor::Color, 24
- SetAngularVelocity
  - FACamera::Camera, 18
- SetAspectRatio
  - FACamera::Camera, 18
- SetBlue
  - FAColor::Color, 24
- SetCameraPosition
  - FACamera::Camera, 18
- SetCameraVelocity
  - FACamera::Camera, 19
- SetColor
  - FAColor::Color, 24
- SetDynamicBuffer
  - FARender::RenderScene, 52
- SetGreen
  - FAColor::Color, 24
- SetHeight
  - FAWindow::Window, 74
- SetPSOAndRootSignature
  - FARender::RenderScene, 53
- SetRed
  - FAColor::Color, 24
- SetStaticBuffer
  - FARender::RenderScene, 53
- SetTextColor
  - FARender::Text, 66
- SetTextLocation
  - FARender::Text, 67
- SetTextSize
  - FARender::Text, 67
- SetTextString
  - FARender::Text, 67
- SetVerticalFov
  - FACamera::Camera, 19
- SetWidth
  - FAWindow::Window, 74
- SetX
  - FACamera::Camera, 19
- SetY
  - FACamera::Camera, 19
- SetZ
  - FACamera::Camera, 19
- SetZFar
  - FACamera::Camera, 19
- SetZNear
  - FACamera::Camera, 20
- Signal
  - FARender::DeviceResources, 32
- Start
  - FATime::Time, 71
- Stop
  - FATime::Time, 71
- SwapChain
  - FARender::SwapChain, 60
- Text
  - FARender::Text, 65
- TextResources
  - FARender::TextResources, 68
- Tick
  - FATime::Time, 71
- Time
  - FATime::Time, 70
- TotalTime
  - FATime::Time, 71
- Transition
  - FARender::MultiSampling, 43
  - FARender::SwapChain, 64
- Up
  - FACamera::Camera, 20
- UpdateCurrentFrameFenceValue
  - FARender::DeviceResources, 32
- UpdatePerspectiveProjectionMatrix
  - FACamera::Camera, 20
- UpdateViewMatrix
  - FACamera::Camera, 20
- UpdateViewPerspectiveProjectionMatrix
  - FACamera::Camera, 20
- vec2
  - FACamera.h, 78
- WaitForGPU
  - FARender::DeviceResources, 32
- Window
  - FAWindow::Window, 72, 73