Farouq Adepetu's Rendering Engine

Generated by Doxygen 1.9.4

Chapter 1

Namespace Index

1.1 Namespace List

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

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

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

Chapter 5

Namespace Documentation

5.1 FACamera Namespace Reference

Has Camera class.

Classes

- class Camera

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

5.1.1 Detailed Description

Has Camera class.

5.2 FAColor Namespace Reference

Has the Color class.

Classes

- class Color

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

Functions

- Color operator+ (const Color &c1, const Color &c2)

  Returns the result of c1 + c2.
- Color operator- (const Color &c1, const Color &c2)

  Returns the result of c1 - c2.
- Color operator∗ (const Color &c, float k)

  Returns the result of c ∗ k.
- Color operator∗ (float k, const Color &c)

  Returns the result of k ∗ c.
- Color operator∗ (const Color &c1, const Color &c2)

  Returns the result of c1 ∗ c2.

5.2.1 Detailed Description

Has the Color class.

5.2.2 Function Documentation

5.2.2.1 operator∗() [1/3]

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

Returns the result of *c* ∗ *k*.

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

5.2.2.2 operator∗() [2/3]

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

Returns the result of *c1* ∗ *c2*.

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

5.2.2.3 operator∗() [3/3]

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

Returns the result of $k * c$.

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

5.2.2.4 operator+()

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

Returns the result of *c1* + *c2*.

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

5.2.2.5 operator-()

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

Returns the result of *c1* - *c2*.

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

5.3 FAProjection Namespace Reference

Within the namespace is the interface IProjection and class PerspectiveProjection.

Classes

- class IProjection

  An interface for projections.
- class PerspectiveProjection

  A class for doing perspective projection.

5.3.1 Detailed Description

Within the namespace is the interface IProjection and class PerspectiveProjection.

5.4 FARender Namespace Reference

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

Classes

- class DepthStencilBuffer

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

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

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

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

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

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

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

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

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

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

Enumerations

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

5.4.1 Detailed Description

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

5.5 FATime Namespace Reference

Has Time class.

Classes

- class Time

5.5.1 Detailed Description

Has Time class.

5.6 FAWindow Namespace Reference

Has Window class.

Classes

- class Window

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

5.6.1 Detailed Description

Has Window class.

Chapter 6

Class Documentation

6.1 FACamera::Camera Class Reference

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

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

Public Member Functions

- Camera ()

  Creates a new camera. Initializes all properties to 0.
- Camera (vec4 cameraPosition, vec4 x, vec4 y, vec4 z, float cameraVelocity, float angularVelocity)

  Creates a new camera.
- void SetProperties (vec4 cameraPosition, vec4 x, vec4 y, vec4 z, float cameraVelocity, float angularVelocity)

  Sets the properties of the camera.
- const vec4 & GetCameraPosition () const

  Returns a constant reference to the position of the camera in world coordinates.
- const vec4 & GetX () const

  Returns a constant reference to the x-axis of the camera.
- const vec4 & GetY () const

  Returnsa constant reference to the y-axis of the camera.
- const vec4 & 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 (vec4 cameraPosition, vec4 target, vec4 up)

  Defines the camera space using UVN.
- void SetCameraPosition (const vec4 &position)

Sets the camera's position to the specified position.

- void SetX (const vec4 &x)

  Sets the camera's x-axis to the specified vector x.
- void SetY (const vec4 &y)

  Sets the camera's y-axis to the specified vector y.
- void SetZ (const vec4 &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 velcoity)

  Sets the camera's angular velocity to the specified angular velocity.
- void UpdateViewMatrix ()

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

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

  Moves the camera right along the camera's x-axis.
- void Foward (float dt)

  Moves the camera foward along the camera's z-axis.
- void Backward (float dt)

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

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

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

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

  Rotates the camera to look up and down.
- void KeyboardInput (float dt)

  Polls keyboard input and moves the camera.
- void KeyboardInputWASD (float dt)

  Polls keyboard input and moves the camera.
- void KeyboardInputArrow (float dt)

  Polls keyboard input and moves the camera.
- void MouseInput ()

  Polls mouse input and rotates the camera.

6.1.1 Detailed Description

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

6.1.2 Constructor & Destructor Documentation

6.1.2.1 Camera() [1/2]

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

Creates a new camera. Initializes all properties to 0.

6.1.2.2 Camera() [2/2]

```
FACamera::Camera::Camera (
            vec4 cameraPosition,
            vec4 x,
            vec4 y,
            vec4 z,
            float cameraVelocity,
            float angularVelocity )
```

Creates a new camera.

Parameters

in	*camerPosition*	The position of the camera.
in	*x*	The x axis of the local coordinate system of the camera.
in	*y*	The y axis of the local coordinate system of the camera.
in	*z*	The z axis of the local coordinate system of the camera.
in	*cameraVelocity*	The translational velocity of the camera.
in	*angularVelocity*	The angular velocity of the camera.

6.1.3 Member Function Documentation

6.1.3.1 Backward()

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

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

Parameters

in	*dt*	The time between frames.

6.1.3.2 Down()

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

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

Parameters

in	*dt*	The time between frames.

6.1.3.3 Foward()

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

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

Parameters

in	*dt*	The time between frames.

6.1.3.4 GetAngularVelocity()

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

Returns the camera's angular velocity.

6.1.3.5 GetCameraPosition()

```
const vec4 & FACamera::Camera::GetCameraPosition ( ) const
```

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

6.1.3.6 GetCameraVelocity()

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

Returns the camera's velocity.

6.1.3.7 GetViewMatrix()

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

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

6.1.3.8 GetX()

```
const vec4 & FACamera::Camera::GetX ( ) const
```

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

6.1.3.9 GetY()

```
const vec4 & FACamera::Camera::GetY ( ) const
```

Returnsa constant reference to the y-axis of the camera.

6.1.3.10 GetZ()

```
const vec4 & FACamera::Camera::GetZ ( ) const
```

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

6.1.3.11 KeyboardInput()

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

Polls keyboard input and moves the camera.

Moves the camera foward/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	*dt*	The time between frames.

6.1.3.12 KeyboardInputArrow()

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

Polls keyboard input and moves the camera.

Moves the camera foward/backward if up/down arrow was pressed.
Moves the camera left/right if left/right arrow was pressed.
Moves the camera up/down if space/crtl was pressed.

Parameters

in	*dt*	The time between frames.

6.1.3.13 KeyboardInputWASD()

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

Polls keyboard input and moves the camera.

Moves the camera foward/backward if w/s was pressed.
Moves the camera left/right if a/d was pressed.
Moves the camera up/down if space/crtl was pressed.

Parameters

in	*dt*	The time between frames.

6.1.3.14 Left()

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

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

6.1.3.15 LookAt()

```
void FACamera::Camera::LookAt (
            vec4 cameraPosition,
            vec4 target,
            vec4 up )
```

Defines the camera space using UVN.

Parameters

in	*camerPosition*	The position of the camera.
in	*target*	The point the camera is looking at.
in	*up*	The up direction of the world.

6.1.3.16 MouseInput()

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

Polls mouse input and rotates the camera.

6.1.3.17 Right()

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

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

Parameters

in	*dt*	The time between frames.

6.1.3.18 RotateCameraLeftRight()

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

Rotates the camera to look left and right.

Parameters

in	*xDiff*	How many degrees to rotate.

6.1.3.19 RotateCameraUpDown()

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

Rotates the camera to look up and down.

Parameters

in	*yDiff*	How many degrees to rotate.

6.1.3.20 SetAngularVelocity()

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

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

6.1.3.21 SetCameraPosition()

```
void FACamera::Camera::SetCameraPosition (
            const vec4 & position )
```

Sets the camera's position to the specified position.

6.1.3.22 SetCameraVelocity()

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

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

6.1.3.23 SetProperties()

```
void FACamera::Camera::SetProperties (
            vec4 cameraPosition,
            vec4 x,
            vec4 y,
            vec4 z,
            float cameraVelocity,
            float angularVelocity )
```

Sets the properties of the camera.

Parameters

in	*camerPosition*	The position of the camera.
in	*x*	The x axis of the local coordinate system of the camera.
in	*y*	The y axis of the local coordinate system of the camera.
in	*z*	The z axis of the local coordinate system of the camera.
in	*cameraVelocity*	The translational velocity of the camera.
in	*angularVelocity*	The angular velocity of the camera.

6.1.3.24 SetX()

```
void FACamera::Camera::SetX (
            const vec4 & x )
```

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

6.1.3.25 SetY()

```
void FACamera::Camera::SetY (
            const vec4 & y )
```

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

6.1.3.26 SetZ()

```
void FACamera::Camera::SetZ (
            const vec4 & z )
```

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

6.1.3.27 Up()

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

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

Parameters

in	*dt*	The time between frames.

6.1.3.28 UpdateViewMatrix()

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

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

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

- C:/Users/Work/Desktop/First Game Engine/First-Game-Engine/FA Rendering Engine/Header Files/FACamera.h

6.2 FAColor::Color Class Reference

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

`#include "FAColor.h"`

Public Member Functions

- Color (float r=0.0f, float g=0.0f, float b=0.0f, float a=1.0f)

  Initializes the color to the specified RGBA values.
- Color (const FAMath::Vector4D &color)

  Initializes the color to the specified color.
- const FAMath::Vector4D & GetColor () const

  Returns the color.
- float GetRed () const

  Returns the value of the red component.
- float GetGreen () const

  Returns the value of the blue component.
- float GetBlue () const

  Returns the value of the green component.
- float GetAlpha () const

  Returns the value of the alpha component.
- void SetColor (const FAMath::Vector4D &color)

  Sets the color to the specified color.
- void SetRed (float r)

  Sets the red component to the specified float value.
- void SetGreen (float g)

  Sets the green component to the specified float value.
- void SetBlue (float b)

  Sets the blue component to the specified float value.
- void SetAlpha (float a)

  Sets the alpha component to the specified float value.
- Color & operator+= (const Color &c)

  Adds this objects color to the specified color c and stores the result in this object.
- Color & operator-= (const Color &c)

  Subtracts the specified color c from this objects color and stores the result in this object.
- Color & operator*= (float k)

  Multiplies this objects color by the specified value k and stores the result in this object.
- Color & operator*= (const Color &c)

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

6.2.1 Detailed Description

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

6.2.2 Constructor & Destructor Documentation

6.2.2.1 Color() [1/2]

```
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.

6.2.2.2 Color() [2/2]

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

Initializes the color to the specified *color*.

6.2.3 Member Function Documentation

6.2.3.1 GetAlpha()

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

Returns the value of the alpha component.

6.2.3.2 GetBlue()

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

Returns the value of the green component.

6.2.3.3 GetColor()

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

Returns the color.

6.2.3.4 GetGreen()

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

Returns the value of the blue component.

6.2.3.5 GetRed()

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

Returns the value of the red component.

6.2.3.6 operator*=() [1/2]

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

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

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

6.2.3.7 operator*=() [2/2]

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

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

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

6.2.3.8 operator+=()

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

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

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

6.2.3.9 operator-=()

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

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

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

6.2.3.10 SetAlpha()

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

Sets the alpha component to the specified float value.

6.2.3.11 SetBlue()

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

Sets the blue component to the specified float value.

6.2.3.12 SetColor()

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

Sets the color to the specified color.

6.2.3.13 SetGreen()

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

Sets the green component to the specified float value.

6.2.3.14 SetRed()

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

Sets the red component to the specified float value.

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

- C:/Users/Work/Desktop/First Game Engine/First-Game-Engine/FA Rendering Engine/Header Files/FAColor.h

6.3 FARender::DepthStencilBuffer Class Reference

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

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

Public Member Functions

- **DepthStencilBuffer** (const DepthStencilBuffer &)=delete
- DepthStencilBuffer & **operator=** (const DepthStencilBuffer &)=delete
- DepthStencilBuffer ()

  Creates a depth stencil buffer object. Call the CreateDepthStencilBufferAndView() to allocate memory for the buffer.
- DepthStencilBuffer (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &dsvHeap, unsigned int index, unsigned int dsvSize, unsigned int width, unsigned int height, DXGI_FORMAT format=DXGI_FORMAT_D24_UNORM_S8_UINT, unsigned int sampleCount=1)

  Creates the depth stencil buffer and view.
- DXGI_FORMAT GetDepthStencilFormat () const

  Returns the format of the depth stencil buffer.
- void CreateDepthStencilBufferAndView (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &dsvHeap, unsigned int index, unsigned int dsvSize, unsigned int width, unsigned int height, DXGI_FORMAT format=DXGI_FORMAT_D24_UNORM_S8_UINT, unsigned int sampleCount=1)

  Creates the depth stencil buffer and view.
- void ReleaseBuffer ()

  Frees the memory of the buffer.
- void ClearDepthStencilBuffer (const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &dsvHeap, unsigned int indexOfView, unsigned int dsvSize, float clearValue)

  Clears the depth stencil buffer with the specified clear value.

6.3.1 Detailed Description

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

6.3.2 Constructor & Destructor Documentation

6.3.2.1 DepthStencilBuffer() [1/2]

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

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

6.3.2.2 DepthStencilBuffer() [2/2]

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

Creates the depth stencil buffer and view.

Parameters

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

6.3.3 Member Function Documentation

6.3.3.1 ClearDepthStencilBuffer()

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

Clears the depth stencil buffer with the specified clear value.

Parameters

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

6.3.3.2 CreateDepthStencilBufferAndView()

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

Creates the depth stencil buffer and view.

Parameters

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

6.3.3.3 GetDepthStencilFormat()

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

Returns the format of the depth stencil buffer.

6.3.3.4 ReleaseBuffer()

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

Frees the memory of the buffer.

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

- C:/Users/Work/Desktop/First Game Engine/First-Game-Engine/FA Rendering Engine/Header Files/FABuffer.h

6.4 FARender::DeviceResources Class Reference

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

`#include "FADeviceResources.h"`

Public Member Functions

- **DeviceResources** (const DeviceResources &)=delete
- DeviceResources & **operator=** (const DeviceResources &)=delete
- ~DeviceResources ()

  Flushes the command queue.
- const Microsoft::WRL::ComPtr< ID3D12Device > & GetDevice () const

  Returns a constant reference to the ID3D12Device objcet.
- const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & GetCommandList () const

  Returns a constant reference to the ID3D12GraphicsCommandList objcet.
- DXGI_FORMAT GetBackBufferFormat () const

  Returns a constant reference to the back buffer format.
- DXGI_FORMAT GetDepthStencilFormat () const

  Returns a constant reference to the depth stencil format.
- unsigned int GetCBVSRVUAVSize () const

  The size of a constant buffer/shader resource/unordered access view.
- unsigned int GetCurrentFrame () const

  Returns the current frame.
- const TextResources & GetTextResources () const

  Returns a constant reference to the TextResources object.
- void UpdateCurrentFrameFenceValue ()

  Updates the current frames fence value.
- void FlushCommandQueue ()

  Synchronizes the CPU and GPU.
- void WaitForGPU () const

  Waits for the GPU to execute all of the commands of the current frame.
- void Signal ()

  Adds an instruction to the GPU to set the fence value to the current fence value.
- void Resize (int width, int height, const HWND &handle, bool isMSAAEnabled, bool isTextEnabled)

  Call when the window gets resized.
- void RTBufferTransition (bool isMSAAEnabled, bool isTextEnabled)

  Transistions 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.

6.4.1 Detailed Description

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

6.4.2 Constructor & Destructor Documentation

6.4.2.1 ~DeviceResources()

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

Flushes the command queue.

6.4.3 Member Function Documentation

6.4.3.1 AfterTextDraw()

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

Executes the text commands.

6.4.3.2 BeforeTextDraw()

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

Prepares to render text.

6.4.3.3 Execute()

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

Executes the command list.

6.4.3.4 FlushCommandQueue()

```
void FARender::DeviceResources::FlushCommandQueue ( )
```

Synchronizes the CPU and GPU.

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

6.4.3.5 GetBackBufferFormat()

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

Returns a constant reference to the back buffer format.

6.4.3.6 GetCBVSRVUAVSize()

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

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

6.4.3.7 GetCommandList()

```
const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & FARender::DeviceResources::Get↩
CommandList ( ) const
```

Returns a constant reference to the ID3D12GraphicsCommandList objcet.

6.4.3.8 GetCurrentFrame()

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

Returns the current frame.

6.4.3.9 GetDepthStencilFormat()

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

Returns a constant reference to the depth stencil format.

6.4.3.10 GetDevice()

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

Returns a constant reference to the ID3D12Device objcet.

6.4.3.11 GetInstance()

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

Call to make an object of DeviceResources.

Only one instance of DeviceResources can exist in a program.

Parameters

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

6.4.3.12 GetTextResources()

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

Returns a constant reference to the TextResources object.

6.4.3.13 NextFrame()

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

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

6.4.3.14 Present()

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

Swaps the front and back buffers.

6.4.3.15 Resize()

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

Call when the window gets resized.

Call when you initialize your program.

Parameters

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

6.4.3.16 RTBufferTransition()

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

Transistions the render target buffer.

Parameters

in	*isMSAAEnabled*	Pass in true if MSAA enabled, false otherwise.
in	*isTextEnabled*	Pass in true if text enabled, false otherwise.

6.4.3.17 Signal()

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

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

6.4.3.18 UpdateCurrentFrameFenceValue()

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

Updates the current frames fence value.

6.4.3.19 WaitForGPU()

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

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

Signal should have been called before this function is called.

6.4.4 Member Data Documentation

6.4.4.1 NUM_OF_FRAMES

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

The number of frames in the ciruclar array.

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

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

- C:/Users/Work/Desktop/First Game Engine/First-Game-Engine/FA Rendering Engine/Header Files/FADeviceResources.h

6.5 DirectXException Class Reference

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

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

Public Member Functions

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

  Constructs a DirectXException object.
- std::wstring ErrorMsg () const

  Returns a message describing the error.

6.5.1 Detailed Description

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

6.5.2 Constructor & Destructor Documentation

6.5.2.1 DirectXException()

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

Constructs a DirectXException object.

Parameters

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

6.5.3 Member Function Documentation

6.5.3.1 ErrorMsg()

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

Returns a message describing the error.

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

- C:/Users/Work/Desktop/First Game Engine/First-Game-Engine/FA Rendering Engine/Header Files/FADirectXException.h

6.6 FADrawArguments::DrawArguments Struct Reference

Data that are used as parameters to draw an object.

`#include "FADrawArgumentsStructure.h"`

Public Attributes

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

6.6.1 Detailed Description

Data that are used as parameters to draw an object.

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

- C:/Users/Work/Desktop/First Game Engine/First-Game-Engine/FA Rendering Engine/Header Files/FADraw↩ ArgumentsStructure.h

6.7 FARender::DynamicBuffer Class Reference

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

`#include "FABuffer.h"`

Public Member Functions

- **DynamicBuffer** (const DynamicBuffer &)=delete
- DynamicBuffer & **operator=** (const DynamicBuffer &)=delete
- DynamicBuffer ()

  Creates a dynamic buffer object. No memory is allocated for the buffer. Call one of the CreateDynamicBuffer() functions to allocate memory for the buffer.
- DynamicBuffer (const Microsoft::WRL::ComPtr< ID3D12Device > &device, unsigned int numOfBytes, unsigned int stride)

  Creates and maps a dynamic vertex buffer or a dynamic constant buffer.
- DynamicBuffer (const Microsoft::WRL::ComPtr< ID3D12Device > &device, unsigned int numOfBytes, DXGI_FORMAT format)

  Creates and maps a dynamic index buffer.
- ~DynamicBuffer ()

  Unmaps the pointer to the dynamic buffer.

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

  Creates and maps a dynamic vertex buffer or a dynamic constant buffer.
- void CreateDynamicBuffer (const Microsoft::WRL::ComPtr< ID3D12Device > &device, unsigned int numOfBytes, DXGI_FORMAT format)

  Creates and maps a dynamic index buffer.
- const D3D12_GPU_VIRTUAL_ADDRESS GetGPUAddress (unsigned int index) const

  Returns the GPU address of the data at the specified index.
- void CreateConstantBufferView (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &cbvHeap, unsigned int cbvSize, unsigned int cbvHeapIndex, unsigned int cBufferIndex)

  Creates the constant buffer view and stores it in the specified descriptor heap.
- const D3D12_VERTEX_BUFFER_VIEW GetVertexBufferView ()

  Returns a the vertex buffer view of the dynamic buffer.
- const D3D12_INDEX_BUFFER_VIEW GetIndexBufferView ()

  Returns the index buffer view of the dynamic buffer.
- void CopyData (unsigned int index, const void *data, unsigned long long numOfBytes)

  Copies data from the given data into the dynamic buffer. Uses 0-indexing.
- void ReleaseBuffer ()

  Frees the dynamic buffer memory.

6.7.1 Detailed Description

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

6.7.2 Constructor & Destructor Documentation

6.7.2.1 DynamicBuffer() [1/3]

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

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

6.7.2.2 DynamicBuffer() [2/3]

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

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

Parameters

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

6.7.2.3 DynamicBuffer() [3/3]

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

Creates and maps a dynamic index buffer.

Parameters

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

6.7.2.4 ~DynamicBuffer()

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

Unmaps the pointer to the dynamic buffer.

6.7.3 Member Function Documentation

6.7.3.1 CopyData()

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

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

Parameters

in	*data*	The data to copy in the dynamic buffer.
in	*numOfBytes*	The number of bytes to copy.

6.7.3.2 CreateConstantBufferView()

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

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

Parameters

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

6.7.3.3 CreateDynamicBuffer() [1/2]

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

Creates and maps a dynamic index buffer.

Parameters

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

6.7.3.4 CreateDynamicBuffer() [2/2]

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

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

Parameters

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

6.7.3.5 GetGPUAddress()

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

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

6.7.3.6 GetIndexBufferView()

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

Returns the index buffer view of the dynamic buffer.

6.7.3.7 GetVertexBufferView()

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

Returns a the vertex buffer view of the dynamic buffer.

6.7.3.8 ReleaseBuffer()

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

Frees the dynamic buffer memory.

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

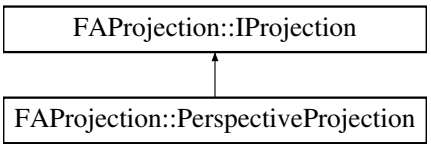
- C:/Users/Work/Desktop/First Game Engine/First-Game-Engine/FA Rendering Engine/Header Files/FABuffer.h

6.8 FAProjection::IProjection Class Reference

An interface for projections.

`#include "FAProjection.h"`

Inheritance diagram for FAProjection::IProjection:



Public Member Functions

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

Protected Attributes

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

6.8.1 Detailed Description

An interface for projections.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 IProjection()

`FAProjection::IProjection::IProjection ( )`

Default Constructor.

6.8.3 Member Function Documentation

6.8.3.1 GetProjectionMatrix()

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

Returns the projection matrix.

6.8.3.2 UpdateProjectionMatrix()

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

Implemented in FAProjection::PerspectiveProjection.

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

- C:/Users/Work/Desktop/First Game Engine/First-Game-Engine/FA Rendering Engine/Header Files/FAProjection.h

6.9 FARender::MultiSampling Class Reference

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

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

Public Member Functions

- **MultiSampling** (const MultiSampling &)=delete
- MultiSampling & **operator=** (const MultiSampling &)=delete
- MultiSampling ()

  Creates a multisampling object. Call the function CheckMultiSamplingSupport() to check if the desired formats and sample count is supported by a GPU. If they are supported, a mulitsampling render target buffer and a mulitsampling depth stencil buffer can be created.
- MultiSampling (const Microsoft::WRL::ComPtr< ID3D12Device > &device, DXGI_FORMAT rtFormat, unsigned int sampleCount)

  Checks if the specifed format and sample count are supported by the specified device for multi-sampling.
- void CheckMultiSamplingSupport (const Microsoft::WRL::ComPtr< ID3D12Device > &device, DXGI_↩ FORMAT rtFormat, unsigned int sampleCount)

  Checks if the specifed format and sample count are supported by the specified device for multi-sampling.
- const Microsoft::WRL::ComPtr< ID3D12Resource > & GetRenderTargetBuffer ()

  Returns a constant refererence to the MSAA render target buffer.
- DXGI_FORMAT GetRenderTargetFormat ()

  Returns the format of the MSAA render target buffer.
- DXGI_FORMAT GetDepthStencilFormat ()

  Returns the format of the MSAA depth stencil buffer.
- void ReleaseBuffers ()

  Resets the MSAA render target buffer and MSAA depth stencil buffer.
- void CreateRenderTargetBufferAndView (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &rtvHeap, unsigned int indexOfWhereToStoreView, 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 indexOfWhereToStoreView, unsigned int dsvSize, unsigned int width, unsigned int height)

  Creates the MSAA depth stencil buffer and a view to it.
- void Transition (const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, D3D12_RESOURCE_STATES before, D3D12_RESOURCE_STATES after)

  Transitions the MSAA render target buffer from the specified before state to the specified after state.
- void ClearRenderTargetBuffer (const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &rtvHeap, unsigned int indexOfView, unsigned int rtvSize, const float *clearValue)

  Clears the MSAA render target buffer with the specified clear value.
- void ClearDepthStencilBuffer (const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &dsvHeap, unsigned int indexOfView, unsigned int dsvSize, float clearValue)

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

6.9.1 Detailed Description

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

6.9.2 Constructor & Destructor Documentation

6.9.2.1 MultiSampling() [1/2]

```
FARender::MultiSampling::MultiSampling ( )
```

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

6.9.2.2 MultiSampling() [2/2]

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

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

Throws a runtime_error if they are not supproted.

Parameters

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

6.9.3 Member Function Documentation

6.9.3.1 CheckMultiSamplingSupport()

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

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

Throws a runtime_error if they are not supproted.

Parameters

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

6.9.3.2 ClearDepthStencilBuffer()

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

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

Parameters

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

6.9.3.3 ClearRenderTargetBuffer()

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

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

Parameters

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

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

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

6.9.3.6 GetDepthStencilFormat()

```
DXGI_FORMAT FARender::MultiSampling::GetDepthStencilFormat ( )
```

Returns the format of the MSAA depth stencil buffer.

6.9.3.7 GetRenderTargetBuffer()

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

Returns a constant refererence to the MSAA render target buffer.

6.9.3.8 GetRenderTargetFormat()

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

Returns the format of the MSAA render target buffer.

6.9.3.9 ReleaseBuffers()

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

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

6.9.3.10 Transition()

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

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

Parameters

in	*commandList*	A Direct3D 12 graphics command list.

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

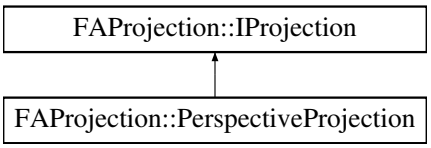
- C:/Users/Work/Desktop/First Game Engine/First-Game-Engine/FA Rendering Engine/Header Files/FAMulti↩ Sampling.h

6.10 FAProjection::PerspectiveProjection Class Reference

A class for doing perspective projection.

`#include "FAProjection.h"`

Inheritance diagram for FAProjection::PerspectiveProjection:



Public Member Functions

- PerspectiveProjection ()
  Initializes all properties to 0.0f.
- PerspectiveProjection (float znear, float zfar, float vFov, float aspectRatio)
  Initializes all the properties to the specified values.
- void SetProperties (float znear, float zfar, float vFov, float aspectRatio)
  Sets all the properties to the specified values.
- float GetNear () const
  Returns the near value of the frustrum.
- float GetFar () const
  Returns the far value of the frustrum.
- float GetVerticalFov () const
  Returns the vertical field of view of the frustrum in degrees.
- float GetAspectRatio () const
  Returns the aspect ratio of the frustrum.
- void SetNear (float znear)

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

- void SetFar (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.

- void UpdateProjectionMatrix () override final

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

Additional Inherited Members

6.10.1 Detailed Description

A class for doing perspective projection.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 PerspectiveProjection() [1/2]

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

Initializes all properties to 0.0f.

6.10.2.2 PerspectiveProjection() [2/2]

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

Initializes all the properties to the specified values.

Parameters

in	*znear*	The z value of where the near plane of the frustrum intersects the z-axis.
in	*zfar*	The z value of where the far plane of the frustrum intersects the z-axis.
in	*vFov*	The vertical field of view of the frustrum.
in	*aspectRatio*	The aspect ratio of the view plane.

6.10.3 Member Function Documentation

6.10.3.1 GetAspectRatio()

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

Returns the aspect ratio of the frustrum.

6.10.3.2 GetFar()

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

Returns the far value of the frustrum.

6.10.3.3 GetNear()

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

Returns the near value of the frustrum.

6.10.3.4 GetVerticalFov()

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

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

6.10.3.5 SetAspectRatio()

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

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

6.10.3.6 SetFar()

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

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

6.10.3.7 SetNear()

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

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

6.10.3.8 SetProperties()

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

Sets all the properties to the specified values.

Parameters

in	*znear*	The z value of where the near plane of the frustrum intersects the z-axis.
in	*zfar*	The z value of where the far plane of the frustrum intersects the z-axis.
in	*vFov*	The vertical field of view of the frustrum.
in	*aspectRatio*	The aspect ratio of the view plane.

6.10.3.9 SetVerticalFov()

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

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

6.10.3.10 UpdateProjectionMatrix()

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

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

Implements FAProjection::IProjection.

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

- C:/Users/Work/Desktop/First Game Engine/First-Game-Engine/FA Rendering Engine/Header Files/FAProjection.h

6.11 FARender::RenderScene Class Reference

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

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

Public Member Functions

- **RenderScene** (const RenderScene &)=delete
- RenderScene **operator=** (const RenderScene &)=delete
- **RenderScene** (unsigned int width, unsigned int height, HWND windowHandle, bool isMSAAEnabled=false, bool isTextEnabled=false)
- void **CreateDeviceResources** (unsigned int width, unsigned int height, HWND windowHandle, bool is↩ MSAAEnabled=false, bool isTextEnabled=false)
- void LoadShader (unsigned int shaderKey, std::wstring_view filename)

  Loads a shaders bytecode and maps it to the specified shaderKey.
- void CompileShader (unsigned int shaderKey, std::wstring_view filename, std::string_view entryPointName, std::string_view target)

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

  Removes the shader bytecode mapped to the specified shaderKey.
- void LoadShader (std::wstring_view shaderKey, std::wstring_view filename)

  Loads a shaders bytecode and maps it to the specified shaderKey.
- void CompileShader (std::wstring_view shaderKey, std::wstring_view filename, std::string_view entryPoint↩ Name, std::string_view target)

  Loads a shader file, compiles it into bytecode and and maps the bytecode to the specified shaderKey.
- void RemoveShader (std::wstring_view shaderKey)

  Removes the shader bytecode mapped to the specified shaderKey.
- void CreateInputElementDescription (unsigned int key, const char ∗semanticName, unsigned int semantic↩ Index, DXGI_FORMAT format, unsigned int inputSlot, unsigned int byteOffset, D3D12_INPUT_↩ CLASSIFICATION inputSlotClass=D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA, unsigned int instanceStepRate=0)

  Creates an input element description and stores in an array mapped to the specified key.
- void CreateInputElementDescription (std::wstring_view key, const char ∗semanticName, unsigned int semanticIndex, DXGI_FORMAT format, unsigned int inputSlot, unsigned int byteOffset, D3D12_INPUT↩ _CLASSIFICATION inputSlotClass=D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA, unsigned int instanceStepRate=0)

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

- void CreateRootDescriptor (unsigned int rootParameterKey, unsigned int shaderRegister)

  Creates a root descriptor and stores it in the array mapped to the specified rootParameterKey.
- void CreateDescriptorRange (unsigned int descriptorRangeKey, D3D12_DESCRIPTOR_RANGE_TYPE type, unsigned int numDescriptors, unsigned int shaderRegister, unsigned int registerSpace, unsigned int offset)

  Creates a descriptor range and stores it in the array mapped to the specified descriptorRangeKey.
- void CreateDescriptorTable (unsigned int rootParameterKey, unsigned int descriptorRangeKey)

  Creates a root descriptor table and stores it in the array mapped to the specified rootParameterKey.
- void CreateRootConstants (unsigned int rootParameterKey, unsigned int shaderRegister, unsigned int num↩ Values)

  Creates a root constant and stores it in the array mapped to the specified rootParameterKey.
- void CreateRootDescriptor (std::wstring_view rootParameterKey, unsigned int shaderRegister)

  Creates a root descriptor and stores it in the array mapped to the specified rootParameterKey.
- void CreateDescriptorRange (std::wstring_view descriptorRangeKey, D3D12_DESCRIPTOR_RANGE_TYPE type, unsigned int numDescriptors, unsigned int shaderRegister, unsigned int registerSpace, unsigned int offset)

  Creates a descriptor range and stores it in the array mapped to the specified descriptorRangeKey.
- void CreateDescriptorTable (std::wstring_view rootParameterKey, unsigned int descriptorRangeKey)

  Creates a root descriptor table and stores it in the array mapped to the specified rootParameterKey.
- void CreateRootConstants (std::wstring_view rootParameterKey, unsigned int shaderRegister, unsigned int numValues)

  Creates a root constant and stores it in the array mapped to the specified rootParameterKey.
- void CreateRootSignature (unsigned int rootSigKey, unsigned int rootParametersKey)

  Creates a root signature and maps it to the specified rootSigKey.
- void CreateRootSignature (unsigned int rootSigKey, unsigned int rootParametersKey, unsigned int statics↩ SamplerKey)

  Creates a root signature and maps it to the specified rootSigKey.
- void CreateStaticSampler (unsigned int staticSamplerKey, D3D12_FILTER filter, D3D12_TEXTURE_↩ ADDRESS_MODE u, D3D12_TEXTURE_ADDRESS_MODE v, D3D12_TEXTURE_ADDRESS_MODE w, unsigned int shaderRegister)

  Creates a static sampler and stores in an an array mapped to the specified key.
- void CreateRootSignature (std::wstring_view rootSigKey, std::wstring_view rootParametersKey)

  Creates a root signature and maps it to the specified rootSigKey.
- void CreateRootSignature (std::wstring_view rootSigKey, std::wstring_view rootParametersKey, std↩ ::wstring_view staticsSamplerKey)

  Creates a root signature and maps it to the specified rootSigKey.
- void CreateStaticSampler (std::wstring_view staticSamplerKey, D3D12_FILTER filter, D3D12_TEXTURE↩ _ADDRESS_MODE u, D3D12_TEXTURE_ADDRESS_MODE v, D3D12_TEXTURE_ADDRESS_MODE w, unsigned int shaderRegister)

  Creates a static sampler and stores in an an array mapped to the specified key.
- void CreatePSO (unsigned int psoKey, D3D12_FILL_MODE fillMode, BOOL enableMultisample, unsigned int vsKey, unsigned int psKey, unsigned int inputElementDescriptionsKey, unsigned int rootSigKey, const D3↩ D12_PRIMITIVE_TOPOLOGY_TYPE &primitiveType, UINT sampleCount=1)

  Creates a PSO and maps it to the specified psoKey.
- void LinkPSOAndRootSignature (unsigned int psoKey, unsigned int rootSigKey)

  Links the PSO and its associated root signature to the pipeline to indicate what settings you want to use to render objects. An out_of_range exception is thrown if any of the keys don't have a mapped values.
- void CreatePSO (std::wstring_view psoKey, D3D12_FILL_MODE fillMode, BOOL enableMultisample, std↩ ::wstring_view vsKey, std::wstring_view psKey, std::wstring_view inputElementDescriptionsKey, std::wstring↩ _view rootSigKey, const D3D12_PRIMITIVE_TOPOLOGY_TYPE &primitiveType, UINT sampleCount=1)

  Creates a PSO and maps it to the specified psoKey.
- void LinkPSOAndRootSignature (std::wstring_view psoKey, std::wstring_view rootSigKey)

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

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

  Creates a static vertex buffer and stores the specified data in the buffer.
- void CreateStaticBuffer (unsigned int staticBufferKey, const void *data, unsigned numBytes, DXGI_FORMAT format)

  Creates a static index buffer and stores the specified data in the buffer.
- void CreateStaticBuffer (unsigned int staticBufferKey, const wchar_t *filename, unsigned int texType, unsigned int index)

  Creates a static texture buffer, stores the data from the file into the buffer and creates a view of the texture.
- void LinkStaticBuffer (unsigned int bufferType, unsigned int staticBufferKey)

  Links the static buffer mapped to the static buffer key to the pipeline.
- void CreateStaticBuffer (std::wstring_view staticBufferKey, const void *data, unsigned numBytes, unsigned int stride)

  Creates a static vertex buffer and stores the specified data in the buffer.
- void CreateStaticBuffer (std::wstring_view staticBufferKey, const void *data, unsigned numBytes, DXGI_↩FORMAT format)

  Creates a static index buffer and stores the specified data in the buffer.
- void CreateStaticBuffer (std::wstring_view staticBufferKey, const wchar_t *filename, unsigned int texType, unsigned int index)

  Creates a static texture buffer, stores the data from the file into the buffer and creates a view of the texture.
- void LinkStaticBuffer (unsigned int bufferType, std::wstring_view staticBufferKey)

  Links the static buffer mapped to the static buffer key to the pipeline.
- void CreateDynamicBuffer (unsigned int dynamicBufferKey, unsigned numBytes, const void *data, unsigned int stride)

  Creates a dynamic vertex buffer or a dynamic constant buffer.
- void CreateDynamicBuffer (unsigned int dynamicBufferKey, unsigned numBytes, const void *data, DXGI_↩FORMAT format)

  Creates a dynamic index buffer.
- void LinkDynamicBuffer (unsigned int bufferType, unsigned int dynamicBufferKey, unsigned int index↩ConstantData=0, unsigned int rootParameterIndex=0)

  Links the dynamic buffer mapped to the dynamic buffer key to the pipeline.
- void CopyDataIntoDynamicBuffer (unsigned int dynamicBufferKey, unsigned int index, const void *data, UINT64 numOfBytes)

  Copies the specified data into the dyanmic buffer mapped to the dynamic buffer key.
- void CreateDynamicBuffer (std::wstring_view dynamicBufferKey, unsigned numBytes, const void *data, unsigned int stride)

  Creates a dynamic vertex buffer or a dynamic constant buffer.
- void CreateDynamicBuffer (std::wstring_view dynamicBufferKey, unsigned numBytes, const void *data, DXGI_FORMAT format)

  Creates a dynamic index buffer.
- void LinkDynamicBuffer (unsigned int bufferType, std::wstring_view dynamicBufferKey, unsigned int index↩ConstantData=0, unsigned int rootParameterIndex=0)

  Links the dynamic buffer mapped to the dynamic buffer key to the pipeline.
- void CopyDataIntoDynamicBuffer (std::wstring_view dynamicBufferKey, unsigned int index, const void *data, UINT64 numOfBytes)

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

  Creates a descriptor heap to store views of textures.
- void LinkTextureViewHeap ()

  Links the texture view heap to the pipeline.
- void LinkTexture (unsigned int rootParameterIndex)

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

- void LinkTexture (unsigned int rootParameterIndex, unsigned int textureViewIndex)

  Links a texture to the pipeline.
- void BeforeRenderObjects (bool isMSAAEnabled=false)

  Puts all of the commands needed in the command list before drawing the objects of the scene.
- void RenderObject (unsigned int indexCount, unsigned int locationFirstIndex, int indexOfFirstVertex, D3D_↩ PRIMITIVE_TOPOLOGY primitive)

  Renders an object with the specified draw arguments.
- void AfterRenderObjects (bool isMSAAEnabled=false, bool isTextEnabled=false)

  Transitions the render target buffer to the correct state and executes commands.
- void BeforeRenderText ()

  Puts all of the commands needed in the command list before drawing the text of the scene. Call before calling the first RenderText function.
- void RenderText (const FAMath::Vector4D &textLocation, const FAColor::Color &textColor, float textSize, const std::wstring &textString, DWRITE_PARAGRAPH_ALIGNMENT alignment=DWRITE_PARAGRAPH↩ _ALIGNMENT_CENTER)

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

  .
- void AfterRenderText ()

  Transitions the render target buffer and executes all of the text drawing commands.
- void AfterRender ()

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

  Executes the commands to fill the vertex and index buffer with data and flushes the queue.
- void Resize (unsigned int width, unsigned int height, HWND windowHandle, bool isMSAAEnabled=false, bool isTextEnabled=false)

  Resizes the window-dependent resources when the window gets resized.
- void SetConstants (unsigned int rootParameterIndex, unsigned int numValues, void *data, unsigned int index)

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

6.11.1 Detailed Description

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

6.11.2 Member Function Documentation

6.11.2.1 AfterRender()

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

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

Call after rendering all your objects and text.

6.11.2.2 AfterRenderObjects()

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

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

Parameters

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

6.11.2.3 AfterRenderText()

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

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

Call after calling all the RenderText functions.

6.11.2.4 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

[in,optional]	isMSAAEnabled Pass in true if MSAA is enabled.

6.11.2.5 BeforeRenderText()

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

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

6.11.2.6 CompileShader() [1/2]

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

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

Parameters

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

6.11.2.7 CompileShader() [2/2]

```
void FARender::RenderScene::CompileShader (
            unsigned int shaderKey,
            std::wstring_view filename,
            std::string_view entryPointName,
            std::string_view target )
```

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

Parameters

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

6.11.2.8 CopyDataIntoDynamicBuffer() [1/2]

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

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

Parameters

in	*dynamicBufferKey*	The key mapped to a dynamic buffer.
in	*index*	The index of where to copy the data to.
in	*data*	The data to copy.
in	*numOfBytes*	The number of bytes to copy.

6.11.2.9 CopyDataIntoDynamicBuffer() [2/2]

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

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

Parameters

in	*dynamicBufferKey*	The key mapped to a dynamic buffer.
in	*index*	The index of where to copy the data to.
in	*data*	The data to copy.
in	*numOfBytes*	The number of bytes to copy.

6.11.2.10 CreateDescriptorRange() [1/2]

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

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

Parameters

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

6.11.2.11 CreateDescriptorRange() [2/2]

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

```
            unsigned int registerSpace,
            unsigned int offset )
```

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

Parameters

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

6.11.2.12 CreateDescriptorTable() [1/2]

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

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

Parameters

in	*rootParameterKey*	The key to a mappped array to store the created root parameter in.
in	*descriptorRangeKey*	The key to an array of descriptor ranges.

6.11.2.13 CreateDescriptorTable() [2/2]

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

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

Parameters

in	*rootParameterKey*	The key to a mappped array to store the created root parameter in.
in	*descriptorRangeKey*	The key to an array of descriptor ranges.

6.11.2.14 CreateDynamicBuffer() [1/4]

```
void FARender::RenderScene::CreateDynamicBuffer (
```

```
            std::wstring_view dynamicBufferKey,
            unsigned numBytes,
            const void * data,
            DXGI_FORMAT format )
```

Creates a dynamic index buffer.

The user can update the data on a per-frame basis.
If the specified key is already mapped to a dynamic buffer, this function does nothing.

Parameters

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

6.11.2.15 CreateDynamicBuffer() [2/4]

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

Creates a dynamic vertex buffer or a dynamic constant buffer.

The user can update the data on a per-frame basis.
If the specified key is already mapped to a dynamic buffer, this function does nothing.

Parameters

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

6.11.2.16 CreateDynamicBuffer() [3/4]

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

Creates a dynamic index buffer.

The user can update the data on a per-frame basis.
If the specified key is already mapped to a dynamic buffer, this function does nothing.

Parameters

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

6.11.2.17 CreateDynamicBuffer() [4/4]

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

Creates a dynamic vertex buffer or a dynamic constant buffer.

The user can update the data on a per-frame basis.
If the specified key is already mapped to a dynamic buffer, this function does nothing.

Parameters

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

6.11.2.18 CreateInputElementDescription() [1/2]

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

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

Parameters

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

6.11.2.19 CreateInputElementDescription() [2/2]

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

6.11.2.20 CreatePSO() [1/2]

```
void FARender::RenderScene::CreatePSO (
            std::wstring_view psoKey,
            D3D12_FILL_MODE fillMode,
            BOOL enableMultisample,
            std::wstring_view vsKey,
            std::wstring_view psKey,
```

```
            std::wstring_view inputElementDescriptionsKey,
            std::wstring_view rootSigKey,
            const D3D12_PRIMITIVE_TOPOLOGY_TYPE & primitiveType,
            UINT sampleCount = 1 )
```

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

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

Parameters

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

6.11.2.21 CreatePSO() [2/2]

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

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

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

Parameters

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

Parameters

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

6.11.2.22 CreateRootConstants() [1/2]

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

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

Parameters

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

6.11.2.23 CreateRootConstants() [2/2]

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

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

Parameters

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

6.11.2.24 CreateRootDescriptor() [1/2]

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

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

Parameters

in	*rootParameterKey*	The key to a mappped array to store the created root parameter in.
in	*shaderRegister*	The register where constant data will be stored.

6.11.2.25 CreateRootDescriptor() [2/2]

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

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

Parameters

in	*rootParameterKey*	The key to a mappped array to store the created root parameter in.
in	*shaderRegister*	The register where constant data will be stored.

6.11.2.26 CreateRootSignature() [1/4]

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

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

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

Parameters

in	*rootSigKey*	The key to map the created root signature to.
in	*rootParameterKey*	The key to a mapped array of root parameters.

6.11.2.27 CreateRootSignature() [2/4]

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

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

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

Parameters

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

6.11.2.28 CreateRootSignature() [3/4]

```
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 excepetion is thrown.

Parameters

in	*rootSigKey*	The key to map the created root signature to.
in	*rootParameterKey*	The key to a mapped array of root parameters.

6.11.2.29 CreateRootSignature() [4/4]

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

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

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

Parameters

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

6.11.2.30 CreateStaticBuffer() [1/6]

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

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

The user cannot update/change the data once it is stored in the buffer.
If the specified key is already mapped to a static buffer, this function does nothing.

Parameters

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

6.11.2.31 CreateStaticBuffer() [2/6]

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

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

The user cannot update/change the data once it is stored in the buffer.
If the specified key is already mapped to a static buffer, this function does nothing.

Parameters

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

6.11.2.32 CreateStaticBuffer() [3/6]

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

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

The user cannot update/change the data once it is stored in the buffer.
If the specified key is already mapped to a static buffer, this function does nothing.

Parameters

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

6.11.2.33 CreateStaticBuffer() [4/6]

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

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

The user cannot update/change the data once it is stored in the buffer.
If the specified key is already mapped to a static buffer, this function does nothing.

Parameters

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

6.11.2.34 CreateStaticBuffer() [5/6]

```
void FARender::RenderScene::CreateStaticBuffer (
```

```
            unsigned int staticBufferKey,
            const void * data,
            unsigned numBytes,
            unsigned int stride )
```

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

The user cannot update/change the data once it is stored in the buffer.
If the specified key is already mapped to a static buffer, this function does nothing.

Parameters

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

6.11.2.35 CreateStaticBuffer() [6/6]

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

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

The user cannot update/change the data once it is stored in the buffer.
If the specified key is already mapped to a static buffer, this function does nothing.

Parameters

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

6.11.2.36 CreateStaticSampler() [1/2]

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

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

Parameters

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

6.11.2.37 CreateStaticSampler() [2/2]

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

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

Parameters

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

6.11.2.38 CreateTextureViewHeap()

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

Creates a descriptor heap to store views of textures.

Parameters

in	*numDescriptors*	The number of views to be stored in the heap.

6.11.2.39 ExecuteAndFlush()

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

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

6.11.2.40 LinkDynamicBuffer() [1/2]

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

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

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

Parameters

in		

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

Parameters

in	*dynamicBufferKey*	The key mapped to a dynamic buffer.
	[in,optional]	indexConstantData The index of where the constant data is in the dynamic buffer.
	[in,optional]	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.

6.11.2.41 LinkDynamicBuffer() [2/2]

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

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

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

Parameters

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	*dynamicBufferKey*	The key mapped to a dynamic buffer.
	[in,optional]	indexConstantData The index of where the constant data is in the dynamic buffer.
	[in,optional]	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.

6.11.2.42 LinkPSOAndRootSignature() [1/2]

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

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

Parameters

in	*psoKey*	The key to a mapped PSO.
in	*rootSigKey*	The key to a mapped root signature.

6.11.2.43 LinkPSOAndRootSignature() [2/2]

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

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

Parameters

in	*psoKey*	The key to a mapped PSO.
in	*rootSigKey*	The key to a mapped root signature.

6.11.2.44 LinkStaticBuffer() [1/2]

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

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

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

Parameters

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

6.11.2.45 LinkStaticBuffer() [2/2]

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

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

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

Parameters

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

6.11.2.46 LinkTexture() [1/2]

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

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

Parameters

in	*rootParameterIndex*	The index of the root parameter in the root signature that has the register the texture will be stored in.

6.11.2.47 LinkTexture() [2/2]

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

Links a texture to the pipeline.

Parameters

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

6.11.2.48 LinkTextureViewHeap()

```
void FARender::RenderScene::LinkTextureViewHeap ( )
```

Links the texture view heap to the pipeline.

6.11.2.49 LoadShader() [1/2]

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

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

Parameters

in	*shaderKey*	The key to map the bytecode to.
in	*filename*	The name of the .cso file.

6.11.2.50 LoadShader() [2/2]

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

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

Parameters

in	*shaderKey*	The key to map the bytecode to.
in	*filename*	The name of the .cso file.

6.11.2.51 RemoveShader() [1/2]

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

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

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

6.11.2.52 RemoveShader() [2/2]

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

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

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

6.11.2.53 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	*indexCount*	The number of indices used to connect the vertices of the objects.
in	*locationFirstIndex*	The location of the first index of the object in an index buffer.
in	*indexOfFirstVertex*	The index of the first vertex of the object in a vertex buffer.
in	*primitive*	The primitve used to render the object.

6.11.2.54 RenderText()

```
void FARender::RenderScene::RenderText (
            const FAMath::Vector4D & textLocation,
            const FAColor::Color & textColor,
            float textSize,
            const std::wstring & textString,
            DWRITE_PARAGRAPH_ALIGNMENT alignment = DWRITE_PARAGRAPH_ALIGNMENT_CENTER )
```

Draws the Text object mapped to the specified *textKey*. Call in between a BeforeRenderText function and a After↩ RenderText function.
.

Ex.
BeforeRenderText()
RenderText()
RenderText()
AfterRenderText()
Throws an out_of_range exception if the textKey is not mapped to a Text object.

Parameters

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

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

6.11.2.56 SetConstants()

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

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

Parameters

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

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

- C:/Users/Work/Desktop/First Game Engine/First-Game-Engine/FA Rendering Engine/Header Files/FARenderScene.h

6.12 FARender::RenderTargetBuffer Class Reference

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

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

Public Member Functions

- **RenderTargetBuffer** (const RenderTargetBuffer &)=delete
- RenderTargetBuffer & **operator=** (const RenderTargetBuffer &)=delete
- RenderTargetBuffer ()

  Creates a render target buffer object. No memory is allocated. Called the CreateRenderTargetBufferAndView() function to allocate memory for the buffer.
- RenderTargetBuffer (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &rtvHeap, unsigned int index, unsigned int rtvSize, unsigned int width, unsigned int height, DXGI_FORMAT format=DXGI_FORMAT_R8G8B8A8_UNORM, unsigned int sampleCount=1)

  Creates the render target buffer and view.
- DXGI_FORMAT GetRenderTargetFormat () const

  Returns the format of the render target buffer.
- Microsoft::WRL::ComPtr< ID3D12Resource > & GetRenderTargetBuffer ()

  Returns a reference to the render target buffer.
- const Microsoft::WRL::ComPtr< ID3D12Resource > & GetRenderTargetBuffer () const

  Returns a constant reference to the render target buffer.

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

  Creates the render target buffer and view.
- void ReleaseBuffer ()

  Frees the memory of the buffer.
- void ClearRenderTargetBuffer (const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &rtvHeap, unsigned int indexOfView, unsigned int rtvSize, const float *clearValue)

  Clears the render target buffer with the specified clear value.

6.12.1 Detailed Description

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

6.12.2 Constructor & Destructor Documentation

6.12.2.1 RenderTargetBuffer() [1/2]

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

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

6.12.2.2 RenderTargetBuffer() [2/2]

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

Creates the render target buffer and view.

Parameters

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

6.12.3 Member Function Documentation

6.12.3.1 ClearRenderTargetBuffer()

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

Clears the render target buffer with the specified clear value.

Parameters

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

6.12.3.2 CreateRenderTargetBufferAndView()

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

Creates the render target buffer and view.

Parameters

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

6.12.3.3 GetRenderTargetBuffer() [1/2]

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

Returns a reference to the render target buffer.

6.12.3.4 GetRenderTargetBuffer() [2/2]

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

Returns a constant reference to the render target buffer.

6.12.3.5 GetRenderTargetFormat()

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

Returns the format of the render target buffer.

6.12.3.6 ReleaseBuffer()

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

Frees the memory of the buffer.

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

- C:/Users/Work/Desktop/First Game Engine/First-Game-Engine/FA Rendering Engine/Header Files/FABuffer.h

6.13 FARender::StaticBuffer Class Reference

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

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

Public Member Functions

- **StaticBuffer** (const StaticBuffer &)=delete
- StaticBuffer & **operator=** (const StaticBuffer &)=delete
- StaticBuffer ()

  Creates a static buffer object. No memory is allocated for the buffer. Call one of the CreateStaticBuffer() functions to allocate memory for the buffer and store data in the buffer.
- StaticBuffer (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const void *data, unsigned int numBytes, unsigned int stride)

  Creates a static vertex buffer and stores all of the specified data in the buffer.
- StaticBuffer (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const void *data, unsigned int numBytes, DXGI_FORMAT format)

  Creates a static index buffer and stores all of the specified data in the buffer.
- StaticBuffer (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const wchar_t *filename)

  Creates a static texture buffer and stores all of the data from the file in the buffer.
- void CreateStaticBuffer (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const void *data, unsigned int numBytes, unsigned int stride)

  Creates a static vertex buffer and stores all of the specified data in the buffer.
- void CreateStaticBuffer (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const void *data, unsigned int numBytes, DXGI_FORMAT format)

  Creates a static index buffer and stores all of the specified data in the buffer.
- void CreateStaticBuffer (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const wchar_t *filename)

  Creates a static texture buffer and stores all of the data from the file in the buffer.
- const D3D12_VERTEX_BUFFER_VIEW GetVertexBufferView () const

  Returns the vertex buffer view of the static buffer.
- const D3D12_INDEX_BUFFER_VIEW GetIndexBufferView () const

  Returns the index buffer view of the static buffers.
- void CreateTexture2DView (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &srvHeap, unsigned int srvSize, unsigned int index)

  Creates a 2D texture view and stores it in the specified heap.
- void CreateTexture2DMSView (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &srvHeap, unsigned int srvSize, unsigned int index)

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

  Frees the static buffer memory.

6.13.1 Detailed Description

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

6.13.2 Constructor & Destructor Documentation

6.13.2.1 StaticBuffer() [1/4]

```
FARender::StaticBuffer::StaticBuffer ( )
```

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

6.13.2.2 StaticBuffer() [2/4]

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

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

Parameters

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

6.13.2.3 StaticBuffer() [3/4]

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

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

Parameters

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

6.13.2.4 StaticBuffer() [4/4]

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

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

Parameters

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

6.13.3 Member Function Documentation

6.13.3.1 CreateStaticBuffer() [1/3]

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

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

Parameters

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

6.13.3.2 CreateStaticBuffer() [2/3]

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

```
            unsigned int numBytes,
            unsigned int stride )
```

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

Parameters

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

6.13.3.3 CreateStaticBuffer() [3/3]

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

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

Parameters

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

6.13.3.4 CreateTexture2DMSView()

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

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

Parameters

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

6.13.3.5 CreateTexture2DView()

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

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

Parameters

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

6.13.3.6 GetIndexBufferView()

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

Returns the index buffer view of the static buffers.

6.13.3.7 GetVertexBufferView()

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

Returns the vertex buffer view of the static buffer.

6.13.3.8 ReleaseBuffer()

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

Frees the static buffer memory.

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

- C:/Users/Work/Desktop/First Game Engine/First-Game-Engine/FA Rendering Engine/Header Files/FABuffer.h

6.14 FARender::SwapChain Class Reference

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

`#include "FASwapChain.h"`

Public Member Functions

- **SwapChain** (const SwapChain &)=delete
- SwapChain & **operator=** (const SwapChain &)=delete
- SwapChain ()

  Creates a swap chain object. Call the CreateSwapChain() function to create a swap chain.
- SwapChain (const Microsoft::WRL::ComPtr< IDXGIFactory4 > &dxgiFactory, const Microsoft::WRL::ComPtr< ID3D12CommandQueue > &commandQueue, HWND windowHandle, DXGI_FORMAT rtFormat=DXGI_FORMAT_R8G8B8A8_UNORM, DXGI_FORMAT dsFormat=DXGI_FORMAT_D24_UNORM_S8_UINT, unsigned int numRenderTargetBuffers=2)

  Creates a swap chain.
- void CreateSwapChain (const Microsoft::WRL::ComPtr< IDXGIFactory4 > &dxgiFactory, const Microsoft::WRL::ComPtr< ID3D12CommandQueue > &commandQueue, HWND windowHandle, DXGI_FORMAT rtFormat=DXGI_FORMAT_R8G8B8A8_UNORM, DXGI_FORMAT dsFormat=DXGI_FORMAT_D24_UNORM_S8_UINT, unsigned int numRenderTargetBuffers=2)

  Creates a swap chain.
- const Microsoft::WRL::ComPtr< ID3D12Resource > & GetCurrentBackBuffer () const

  Returns a constant reference to the current render target buffer.
- unsigned int GetNumRenderTargetBuffers () const

  Returns the number of swap chain buffers.
- unsigned int GetCurrentBackBufferIndex () const

  Returns the current back buffer index.
- DXGI_FORMAT GetBackBufferFormat () const

  Returns the format of the swap chain.
- DXGI_FORMAT GetDepthStencilFormat () const

  Returns the format of the depth stencil buffer.
- const std::vector< std::unique_ptr< RenderTargetBuffer > > & **GetRenderTargetBuffers** ()
- void ReleaseBuffers ()

  Frees the memory of the render target and depth stencil buffers.
- void CreateRenderTargetBuffersAndViews (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &rtvHeap, unsigned int index, unsigned int rtvSize, unsigned width, unsigned height)

  Creates the swap chains render target buffers and views to them.
- void CreateDepthStencilBufferAndView (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &dsvHeap, unsigned int index, unsigned int dsvSize, unsigned int width, unsigned int height)

  Creates the swap chains depth stencil buffer and view to it.
- void ClearCurrentBackBuffer (const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &rtvHeap, unsigned int indexOfFirstView, unsigned int rtvSize, const float *backBufferClearValue)

  Clears the current render target buffer.
- void ClearDepthStencilBuffer (const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &dsvHeap, unsigned int indexOfView, unsigned int dsvSize, float clearValue)

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

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

  Transitions the current render target buffer from the specified before state to the specified after state.
- void Present ()

  Swaps the front and back buffers.

6.14.1 Detailed Description

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

6.14.2 Constructor & Destructor Documentation

6.14.2.1 SwapChain() [1/2]

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

Creates a swap chain object. Call the CreateSwapChain() function to create a swap chain.

6.14.2.2 SwapChain() [2/2]

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

Creates a swap chain.

Parameters

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

6.14.3 Member Function Documentation

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

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

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

6.14.3.4 CreateRenderTargetBuffersAndViews()

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

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

Parameters

in	*device*	A Direct3D 12 device.
in	*rtvHeap*	A descriptor heap for storing render target descriptors.
in	*index*	The index of where to store the created descriptor in the descriptor heap.
in	*rtvSize*	The size of a render target descriptor.
in	*width*	The width of the render target buffers.
in	*height*	The height of the render target buffers.

6.14.3.5 CreateSwapChain()

```
void FARender::SwapChain::CreateSwapChain (
            const Microsoft::WRL::ComPtr< IDXGIFactory4 > & dxgiFactory,
```

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

Creates a swap chain.

Parameters

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

6.14.3.6 GetBackBufferFormat()

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

Returns the format of the swap chain.

6.14.3.7 GetCurrentBackBuffer()

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

Returns a constant reference to the current render target buffer.

6.14.3.8 GetCurrentBackBufferIndex()

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

Returns the current back buffer index.

6.14.3.9 GetDepthStencilFormat()

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

Returns the format of the depth stencil buffer.

6.14.3.10 GetNumRenderTargetBuffers()

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

Returns the number of swap chain buffers.

6.14.3.11 Present()

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

Swaps the front and back buffers.

6.14.3.12 ReleaseBuffers()

```
void FARender::SwapChain::ReleaseBuffers ( )
```

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

6.14.3.13 Transition()

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

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

Parameters

in	*commandList*	A Direct3D 12 graphics command list.

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

- C:/Users/Work/Desktop/First Game Engine/First-Game-Engine/FA Rendering Engine/Header Files/FASwap↩ Chain.h

6.15 FARender::Text Class Reference

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

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

Public Member Functions

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

  Constructs a Text object.
- const FAMath::Vector4D & GetTextLocation () const

  Returns a constant reference to the text location.
- const std::wstring & GetTextString () const

  Returns a constant reference to the text string.
- float GetTextSize () const

  Returns the text size.
- const FAColor::Color & GetTextColor () const

  Returns a constant reference to the text color.
- void SetTextSize (float textSize)

  Changes the text size to the specified textSize.
- void SetTextColor (const FAColor::Color &textColor)

  Changes the text color to the specified textColor.
- void SetTextString (const std::wstring &textString)

  Changes the text string to the specified textString.
- void SetTextLocation (const FAMath::Vector4D &textLocation)

  Changes the text location to the specified textLocation.

6.15.1 Detailed Description

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

6.15.2 Constructor & Destructor Documentation

6.15.2.1 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	*textLocation*	The location of the text on the window.
in	*textString*	The text to render.
in	*textSize*	How big the text is.
in	*textColor*	The color of the text.

6.15.3 Member Function Documentation

6.15.3.1 GetTextColor()

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

Returns a constant reference to the text color.

6.15.3.2 GetTextLocation()

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

Returns a constant reference to the text location.

6.15.3.3 GetTextSize()

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

Returns the text size.

6.15.3.4 GetTextString()

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

Returns a constant reference to the text string.

6.15.3.5 SetTextColor()

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

Changes the text color to the specified *textColor*.

6.15.3.6 SetTextLocation()

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

Changes the text location to the specified *textLocation*.

6.15.3.7 SetTextSize()

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

Changes the text size to the specified *textSize*.

6.15.3.8 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:

- C:/Users/Work/Desktop/First Game Engine/First-Game-Engine/FA Rendering Engine/Header Files/FAText.h

6.16 FARender::TextResources Class Reference

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

`#include "FATextResources.h"`

Public Member Functions

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

  Initializes the text resources.
- const Microsoft::WRL::ComPtr< ID2D1DeviceContext > & GetDirect2DDeviceContext () const

  Returns a constant reference to the direct 2D device context.
- const Microsoft::WRL::ComPtr< IDWriteFactory > & GetDirectWriteFactory () const

  Returns a constant reference to the direct direct write factory.
- void ResetBuffers ()

  Resets the text buffers.
- void ResizeBuffers (const std::vector< std::unique_ptr< RenderTargetBuffer > > &renderTargetBuffers, HWND windowHandle)

  Resizes the buffers.
- void BeforeRenderText (unsigned int currentBackBuffer)

  Prepares to render text.
- void AfterRenderText (unsigned int currentBackBuffer)

  Executes text commands.

6.16.1 Detailed Description

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

6.16.2 Constructor & Destructor Documentation

6.16.2.1 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	*device*	A Direct3D 12 device.
in	*commandQueue*	A Direct3D 12 command queue.
in	*numSwapChainBuffers*	The number of swap chain render target buffers.

6.16.3 Member Function Documentation

6.16.3.1 AfterRenderText()

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

Executes text commands.

Parameters

in	*currentBackBuffer*	The index of the current render target buffer.

6.16.3.2 BeforeRenderText()

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

Prepares to render text.

Parameters

in	*currentBackBuffer*	The index of the current render target buffer.

6.16.3.3 GetDirect2DDeviceContext()

```
const Microsoft::WRL::ComPtr< ID2D1DeviceContext > & FARender::TextResources::GetDirect2↩
DDeviceContext ( ) const
```

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

6.16.3.4 GetDirectWriteFactory()

```
const Microsoft::WRL::ComPtr< IDWriteFactory > & FARender::TextResources::GetDirectWrite↩
Factory ( ) const
```

Returns a constant reference to the direct direct write factory.

6.16.3.5 ResetBuffers()

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

Resets the text buffers.

6.16.3.6 ResizeBuffers()

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

Resizes the buffers.

Parameters

in	*renderTargetBuffers*	An array of render target buffers.
in	*windowHandle*	A handle to a window.

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

- C:/Users/Work/Desktop/First Game Engine/First-Game-Engine/FA Rendering Engine/Header Files/FAText↩ Resources.h

6.17 FATime::Time Class Reference

Public Member Functions

- float **GetPrevTime** () const
- float **GetDeltaTime** () const
- bool **GetIsTimeStopped** () const
- void **Reset** ()
- void **Tick** ()
- void **Start** ()
- void **Stop** ()

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

- C:/Users/Work/Desktop/First Game Engine/First-Game-Engine/FA Rendering Engine/Header Files/FATime.h

6.18 FAWindow::Window Class Reference

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

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

Public Member Functions

- **Window** (const Window &)=delete
- Window & **operator=** (const Window &)=delete
- Window ()

  Creates a Window object. No window is created. Call CreateParentWindow() or CreateChildWindow() or CreateControlWindow() to create a window.
- Window (const HINSTANCE &hInstance, WNDPROC windowProcedure, const FAColor::Color &background↩ Color, const std::wstring &windowClassName, const std::wstring &windowName, unsigned int styles, unsigned int x, unsigned int y, unsigned int width, unsigned int height, void ∗additionalData=nullptr)

  Creates a parent window.
- Window (const HINSTANCE &hInstance, HWND parent, unsigned int identifier, WNDPROC window↩ Procedure, const FAColor::Color &backgroundColor, const std::wstring &windowClassName, const std↩ ::wstring &windowName, unsigned int styles, unsigned int x, unsigned int y, unsigned int width, unsigned int height, void ∗additionalData=nullptr)

  Creates a non-control child window.
- Window (const HINSTANCE &hInstance, HWND parent, unsigned int identifier, const std::wstring &window↩ ClassName, const std::wstring &windowName, unsigned int styles, unsigned int x, unsigned int y, unsigned int width, unsigned int height, void ∗additionalData=nullptr)

  Creates a control window.

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

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

  Creates a non-control child window.
- void CreateControlWindow (const HINSTANCE &hInstance, HWND parent, unsigned long long int identifier, const std::wstring &windowClassName, const std::wstring &windowName, unsigned int styles, unsigned int x, unsigned int y, unsigned int width, unsigned int height, void *additionalData=nullptr)

  Creates a control window.
- HWND GetWindowHandle () const

  Returns the window handle.
- unsigned int GetWidth () const

  Returns the width of the window.
- unsigned int GetHeight () const

  Returns the height of the window.
- unsigned int GetX () const

  Returns the x position of the top left corner of the window.
- unsigned int GetY () const

  Returns the y position of the top left corner of the window.
- void SetWidth (unsigned int width)

  Sets the width of the window to the specified width.
- void SetHeight (unsigned int height)

  Sets the height of the window o the specified height.
- void SetX (unsigned int x)

  Sets the x position of the top left corner of the window.
- void SetY (unsigned int y)

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

6.18.1 Detailed Description

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

6.18.2 Constructor & Destructor Documentation

6.18.2.1 Window() [1/4]

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

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

6.18.2.2 Window() [2/4]

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

Creates a parent window.

The window gets displayed after it is created.

Parameters

in	*hInstance*	The handle to a module used to identify the executable.
in	*windowProcedure*	The window procedure that is called when an event occurs.
in	*backgroundColor*	The background color the window.
in	*windowClassName*	The name of the window class.
in	*windowName*	The name of the window.
in	*styles*	The style of the window. OR together the styles at https://learn.microsoft.↩ com/en-us/windows/win32/winmsg/window-styles
in	*The*	x position of the top left corner of the window from the desktops top left corner. Use CW_USEDEFAULT to let system select a default position for you.
in	*The*	y position of the top left corner of the windo from the desktops top left corner. Use CW_USEDEFAULT to let system select a default position for you.
in	*width*	The width of the client area of the window.
in	*height*	The height of the client area of the window.
	[in,optional]	parent a handle to a parent. Set to nullptr if it is not a child window.
	[in,optional]	additionalData A pointer to data to access in the window procedure.

6.18.2.3 Window() [3/4]

```
FAWindow::Window::Window (
            const HINSTANCE & hInstance,
            HWND parent,
            unsigned int identifier,
            WNDPROC windowProcedure,
            const FAColor::Color & backgroundColor,
            const std::wstring & windowClassName,
            const std::wstring & windowName,
            unsigned int styles,
            unsigned int x,
```

```
            unsigned int y,
            unsigned int width,
            unsigned int height,
            void * additionalData = nullptr )
```

Creates a non-control child window.

Parameters

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

6.18.2.4 Window() [4/4]

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

Creates a control window.

Parameters

in	*hInstance*	The handle to a module used to identify the executable.
in	*parent*	A handle to a parent window.
in	*identifier*	An unsigned integer to identify the child window.
in	*windowClass*	The name of the window class.
in	*windowName*	The name of the window.

Parameters

in	*styles*	The style of the window. OR together the styles at https://learn.microsoft.↩ com/en-us/windows/win32/winmsg/window-styles
in	*The*	x position of the top left corner of the window from the parent window top left corner.
in	*The*	y position of the top left corner of the window from the parent window top left corner.
in	*width*	The width of the client area of the window.
in	*height*	The height of the client area of the window.
	[in,optional]	additionalData A pointer to data to access in the window procedure.

6.18.3 Member Function Documentation

6.18.3.1 CreateChildWindow()

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

Creates a non-control child window.

Parameters

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

6.18.3.2 CreateControlWindow()

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

Creates a control window.

Parameters

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

6.18.3.3 CreateParentWindow()

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

Creates a parent window.

The window gets displayed after it is created.

Parameters

in	*hInstance*	The handle to a module used to identify the executable.
in	*windowProcedure*	The window procedure that is called when an event occurs.
in	*backgroundColor*	The background color the window.
in	*windowClassName*	The name of the window class.
in	*windowName*	The name of the window.
in	*styles*	The style of the window. OR together the styles at https://learn.microsoft.↩ com/en-us/windows/win32/winmsg/window-styles
in	*The*	x position of the top left corner of the window from the desktops top left corner. Use CW_USEDEFAULT to let system select a default position for you.
in	*The*	y position of the top left corner of the windo from the desktops top left corner. Use CW_USEDEFAULT to let system select a default position for you.
in	*width*	The width of the client area of the window.
in	*height*	The height of the client area of the window.
	[in,optional]	parent a handle to a parent. Set to nullptr if it is not a child window.
	[in,optional]	additionalData A pointer to data to access in the window procedure.

6.18.3.4 GetHeight()

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

Returns the height of the window.

6.18.3.5 GetWidth()

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

Returns the width of the window.

6.18.3.6 GetWindowHandle()

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

Returns the window handle.

6.18.3.7 GetX()

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

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

6.18.3.8 GetY()

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

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

6.18.3.9 SetHeight()

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

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

6.18.3.10 SetWidth()

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

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

6.18.3.11 SetX()

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

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

6.18.3.12 SetY()

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

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

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

- C:/Users/Work/Desktop/First Game Engine/First-Game-Engine/FA Rendering Engine/Header Files/FAWindow.h

Chapter 7

File Documentation

7.1 DDSTextureLoader.h

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

```
59                                      _In_ ID3D12GraphicsCommandList* cmdList,
60                                      _In_reads_bytes_(ddsDataSize) const uint8_t* ddsData,
61                                      _In_ size_t ddsDataSize,
62                                      _Out_ Microsoft::WRL::ComPtr<ID3D12Resource>& texture,
63                                      _Out_ Microsoft::WRL::ComPtr<ID3D12Resource>& textureUploadHeap,
64                                      _In_ size_t maxsize = 0,
65                                      _Out_opt_ DDS_ALPHA_MODE* alphaMode = nullptr
66                                      );
67
68      HRESULT CreateDDSTextureFromFile12(_In_ ID3D12Device* device,
69          _In_ ID3D12GraphicsCommandList* cmdList,
70          _In_z_ const wchar_t* szFileName,
71          _Out_ Microsoft::WRL::ComPtr<ID3D12Resource>& texture,
72          _Out_ Microsoft::WRL::ComPtr<ID3D12Resource>& textureUploadHeap,
73          _In_ size_t maxsize = 0,
74          _Out_opt_ DDS_ALPHA_MODE* alphaMode = nullptr
75      );
76 }
```

7.2 Direct3DLink.h

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

7.3 C:/Users/Work/Desktop/First Game Engine/First-Game-Engine/FA Rendering Engine/Header Files/FABuffer.h File Reference

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

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

Classes

- class FARender::RenderTargetBuffer

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

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

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

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

Namespaces

- namespace FARender

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

Enumerations

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

7.3.1 Detailed Description

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

7.4 FABuffer.h

Go to the documentation of this file.

```
1 #pragma once
2
7 #include <wrl.h>
8 #include <d3d12.h>
9
13 namespace FARender
14 {
15     enum BufferTypes { VERTEX_BUFFER, INDEX_BUFFER, CONSTANT_BUFFER, TEXTURE_BUFFER };
16     enum TextureTypes { TEX2D, TEX2D_MS };
17
22     class RenderTargetBuffer
23     {
24     public:
25
26         //No copying
27         RenderTargetBuffer(const RenderTargetBuffer&) = delete;
28         RenderTargetBuffer& operator=(const RenderTargetBuffer&) = delete;
29
33         RenderTargetBuffer();
34
45         RenderTargetBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
46             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int index, unsigned int
    rtvSize,
47             unsigned int width, unsigned int height, DXGI_FORMAT format = DXGI_FORMAT_R8G8B8A8_UNORM,
    unsigned int sampleCount = 1);
48
51         DXGI_FORMAT GetRenderTargetFormat() const;
52
55         Microsoft::WRL::ComPtr<ID3D12Resource>& GetRenderTargetBuffer();
56
59         const Microsoft::WRL::ComPtr<ID3D12Resource>& GetRenderTargetBuffer() const;
60
71         void CreateRenderTargetBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
72             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int index, unsigned int
    rtvSize,
73             unsigned int width, unsigned int height, DXGI_FORMAT format = DXGI_FORMAT_R8G8B8A8_UNORM,
    unsigned int sampleCount = 1);
74
77         void ReleaseBuffer();
78
89         void ClearRenderTargetBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
    commandList,
90             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int indexOfView,
    unsigned int rtvSize,
91             const float* clearValue);
92
93     private:
94         Microsoft::WRL::ComPtr<ID3D12Resource> mRenderTargetBuffer;
95     };
96
101     class DepthStencilBuffer
102     {
103     public:
104
105         //No copying
106         DepthStencilBuffer(const DepthStencilBuffer&) = delete;
107         DepthStencilBuffer& operator=(const DepthStencilBuffer&) = delete;
108
112         DepthStencilBuffer();
113
```

```
124         DepthStencilBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
125             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int index, unsigned
    int dsvSize,
126             unsigned int width, unsigned int height, DXGI_FORMAT format = DXGI_FORMAT_D24_UNORM_S8_UINT,
    unsigned int sampleCount = 1);
127
130         DXGI_FORMAT GetDepthStencilFormat() const;
131
142         void CreateDepthStencilBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
143             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int index, unsigned
    int dsvSize,
144             unsigned int width, unsigned int height, DXGI_FORMAT format = DXGI_FORMAT_D24_UNORM_S8_UINT,
    unsigned int sampleCount = 1);
145
148         void ReleaseBuffer();
149
160         void ClearDepthStencilBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
    commandList,
161             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int indexOfView,
    unsigned int dsvSize,
162             float clearValue);
163
164     private:
165         Microsoft::WRL::ComPtr<ID3D12Resource> mDepthStencilBuffer;
166     };
167
168
173     class StaticBuffer
174     {
175     public:
176
177         //No copying
178         StaticBuffer(const StaticBuffer&) = delete;
179         StaticBuffer& operator=(const StaticBuffer&) = delete;
180
184         StaticBuffer();
185
198         StaticBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
199             const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const void* data,
200             unsigned int numBytes, unsigned int stride);
201
214         StaticBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
215             const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const void* data,
216             unsigned int numBytes, DXGI_FORMAT format);
217
230         StaticBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
231             const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const wchar_t*
    filename);
232
245         void CreateStaticBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
246             const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const void* data,
247             unsigned int numBytes, unsigned int stride);
248
261         void CreateStaticBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
262             const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const void* data,
263             unsigned int numBytes, DXGI_FORMAT format);
264
277         void CreateStaticBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
278             const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const wchar_t*
    filename);
279
282         const D3D12_VERTEX_BUFFER_VIEW GetVertexBufferView() const;
283
286         const D3D12_INDEX_BUFFER_VIEW GetIndexBufferView() const;
287
298         void CreateTexture2DView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
299             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& srvHeap, unsigned int srvSize, unsigned
    int index);
300
311         void CreateTexture2DMSView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
312             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& srvHeap, unsigned int srvSize, unsigned
    int index);
313
316         void ReleaseBuffer();
317
318     private:
319         Microsoft::WRL::ComPtr<ID3D12Resource> mStaticDefaultBuffer;
320         Microsoft::WRL::ComPtr<ID3D12Resource> mStaticUploadBuffer;
321
322         union
323         {
324             unsigned int mStride;
325             DXGI_FORMAT mFormat;
326         };
327     };
328
333     class DynamicBuffer
```

```
334     {
335     public:
336
337         //No copying
338         DynamicBuffer(const DynamicBuffer&) = delete;
339         DynamicBuffer& operator=(const DynamicBuffer&) = delete;
340
344         DynamicBuffer();
345
354         DynamicBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device, unsigned int numOfBytes,
    unsigned int stride);
355
364         DynamicBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device, unsigned int numOfBytes,
    DXGI_FORMAT format);
365
368         ~DynamicBuffer();
369
378         void CreateDynamicBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device, unsigned int
    numOfBytes, unsigned int stride);
379
388         void CreateDynamicBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device, unsigned int
    numOfBytes, DXGI_FORMAT format);
389
392         const D3D12_GPU_VIRTUAL_ADDRESS GetGPUAddress(unsigned int index) const;
393
402         void CreateConstantBufferView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
403             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& cbvHeap, unsigned int cbvSize, unsigned
    int cbvHeapIndex,
404             unsigned int cBufferIndex);
405
408         const D3D12_VERTEX_BUFFER_VIEW GetVertexBufferView();
409
412         const D3D12_INDEX_BUFFER_VIEW GetIndexBufferView();
413
420         void CopyData(unsigned int index, const void* data, unsigned long long numOfBytes);
421
424         void ReleaseBuffer();
425
426     private:
427         Microsoft::WRL::ComPtr<ID3D12Resource> mDynamicBuffer;
428         BYTE* mMappedData{ nullptr };
429
430         union
431         {
432             UINT mStride;
433             DXGI_FORMAT mFormat;
434         };
435     };
436 }
```

7.5 C:/Users/Work/Desktop/First Game Engine/First-Game-Engine/FA Rendering Engine/Header Files/FACamera.h File Reference

File that has namespace FACamera. Withn the namespace is the class Camera.

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

Classes

- class FACamera::Camera

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

Namespaces

- namespace FACamera

  Has Camera class.

Typedefs

- typedef FAMath::Vector2D vec2
- typedef FAMath::Vector3D **vec3**
- typedef FAMath::Vector4D **vec4**
- typedef FAMath::Matrix4x4 **mat4**

7.5.1 Detailed Description

File that has namespace FACamera. Withn the namespace is the class Camera.

7.5.2 Typedef Documentation

7.5.2.1 vec2

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

FACAMERA_H FILE

7.6 FACamera.h

Go to the documentation of this file.

```
1 #pragma once
2
12 #include "FAMathEngine.h"
13 #include <Windows.h>
14
15 typedef FAMath::Vector2D vec2;
16 typedef FAMath::Vector3D vec3;
17 typedef FAMath::Vector4D vec4;
18 typedef FAMath::Matrix4x4 mat4;
19
23 namespace FACamera
24 {
29     class Camera
30     {
31     public:
32
35         Camera();
36
46         Camera(vec4 cameraPosition, vec4 x, vec4 y, vec4 z, float cameraVelocity, float angularVelocity);
47
57         void SetProperties(vec4 cameraPosition, vec4 x, vec4 y, vec4 z, float cameraVelocity, float
      angularVelocity);
58
61         const vec4& GetCameraPosition() const;
62
65         const vec4& GetX() const;
66
69         const vec4& GetY() const;
70
73         const vec4& GetZ() const;
74
77         const mat4& GetViewMatrix() const;
78
81         float GetCameraVelocity() const;
82
85         float GetAngularVelocity() const;
86
93         void LookAt(vec4 cameraPosition, vec4 target, vec4 up);
94
```

```
97          void SetCameraPosition(const vec4& position);
98
101          void SetX(const vec4& x);
102
105          void SetY(const vec4& y);
106
109          void SetZ(const vec4& z);
110
113          void SetCameraVelocity(float velocity);
114
117          void SetAngularVelocity(float velcoity);
118
121          void UpdateViewMatrix();
122
125          void Left(float dt);
126
131          void Right(float dt);
132
137          void Foward(float dt);
138
143          void Backward(float dt);
144
149          void Up(float dt);
150
155          void Down(float dt);
156
161          void RotateCameraLeftRight(float xDiff);
162
167          void RotateCameraUpDown(float yDiff);
168
177          void KeyboardInput(float dt);
178
187          void KeyboardInputWASD(float dt);
188
197          void KeyboardInputArrow(float dt);
198
201          void MouseInput();
202
203      private:
204          //camera position in world coordinates
205          vec4 mCameraPosition;
206
207          //x-axis of the camera coordinate system
208          vec4 mX;
209
210          //y-axis of the camera coordinate system
211          vec4 mY;
212
213          //z-axis of the camera coordinate system
214          vec4 mZ;
215
216          //stores the world to camera transform
217          mat4 mViewMatrix;
218
219          float mCameraVelocity;
220          float mAngularVelocity;
221
222          vec2 mLastMousePosition;
223
224          bool mUpdateViewMatrix;
225      };
226  }
```

7.7 C:/Users/Work/Desktop/First Game Engine/First-Game-Engine/FA Rendering Engine/Header Files/FAColor.h File Reference

File has class Color under namespace FAColor.

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

Classes

- class FAColor::Color

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

Namespaces

- namespace FAColor

  Has the Color class.

Functions

- Color FAColor::operator+ (const Color &c1, const Color &c2)

  Returns the result of c1 + c2.
- Color FAColor::operator- (const Color &c1, const Color &c2)

  Returns the result of c1 - c2.
- Color FAColor::operator∗ (const Color &c, float k)

  Returns the result of c ∗ k.
- Color FAColor::operator∗ (float k, const Color &c)

  Returns the result of k ∗ c.
- Color FAColor::operator∗ (const Color &c1, const Color &c2)

  Returns the result of c1 ∗ c2.

7.7.1 Detailed Description

File has class Color under namespace FAColor.

7.8 FAColor.h

Go to the documentation of this file.

```
1 #pragma once
2
3 #include "FAMathEngine.h"
4
12 namespace FAColor
13 {
19     class Color
20     {
21     public:
22
25         Color(float r = 0.0f, float g = 0.0f, float b = 0.0f, float a = 1.0f);
26
29         Color(const FAMath::Vector4D& color);
30
33         const FAMath::Vector4D& GetColor() const;
34
37         float GetRed() const;
38
41         float GetGreen() const;
42
45         float GetBlue() const;
46
49         float GetAlpha() const;
50
53         void SetColor(const FAMath::Vector4D& color);
54
57         void SetRed(float r);
58
61         void SetGreen(float g);
62
65         void SetBlue(float b);
66
69         void SetAlpha(float a);
70
75         Color& operator+=(const Color& c);
76
81         Color& operator-=(const Color& c);
82
88         Color& operator*=(float k);
```

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

7.9 C:/Users/Work/Desktop/First Game Engine/First-Game-Engine/FA Rendering Engine/Header Files/FADeviceResources.h File Reference

File has class DeviceResources under namespace FARender.

```
#include <wrl.h>
#include <d3d12.h>
#include <dxgi1_4.h>
#include "FASwapChain.h"
#include "FAMultiSampling.h"
#include "FATextResources.h"
```

Classes

- class FARender::DeviceResources

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

Namespaces

- namespace FARender

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

7.9.1 Detailed Description

File has class DeviceResources under namespace FARender.

7.10 FADeviceResources.h

Go to the documentation of this file.

```
1 #pragma once
2
7 #include <wrl.h>
8 #include <d3d12.h>
9 #include <dxgi1_4.h>
10 #include "FASwapChain.h"
11 #include "FAMultiSampling.h"
12 #include "FATextResources.h"
13
14 namespace FARender
15 {
20     class DeviceResources
21     {
22     public:
23
24         //No copying
25         DeviceResources(const DeviceResources&) = delete;
26         DeviceResources& operator=(const DeviceResources&) = delete;
27
32         static const unsigned int NUM_OF_FRAMES{ 3 };
33
45         static DeviceResources& GetInstance(unsigned int width, unsigned int height, HWND windowHandle,
      bool isMSAAEnabled, bool isTextEnabled);
46
49         ~DeviceResources();
50
53         const Microsoft::WRL::ComPtr<ID3D12Device>& GetDevice() const;
54
57         const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& GetCommandList() const;
58
61         DXGI_FORMAT GetBackBufferFormat() const;
62
65         DXGI_FORMAT GetDepthStencilFormat() const;
66
69         unsigned int GetCBVSRVUAVSize() const;
70
73         unsigned int GetCurrentFrame() const;
74
77         const TextResources& GetTextResources() const;
78
81         void UpdateCurrentFrameFenceValue();
82
88         void FlushCommandQueue();
89
94         void WaitForGPU() const;
95
98         void Signal();
99
110         void Resize(int width, int height, const HWND& handle, bool isMSAAEnabled, bool isTextEnabled);
111
117         void RTBufferTransition(bool isMSAAEnabled, bool isTextEnabled);
118
121         void BeforeTextDraw();
122
125         void AfterTextDraw();
126
129         void Execute() const;
130
133         void Present();
134
135         /*@brief Calls the necessary functions to let the user draw their objects.
136 *
137 *  @param[in] isMSAAEnabled Pass in true if MSAA enabled, false otherwise.
138 */
139         void Draw(bool isMSAAEnabled);
140
143         void NextFrame();
144
145     private:
146
159         DeviceResources(unsigned int width, unsigned int height, HWND windowHandle,
160             bool isMSAAEnabled, bool isTextEnabled);
161
162         unsigned int mCurrentFrameIndex;
163
164         Microsoft::WRL::ComPtr<ID3D12Device> mDirect3DDevice;
165
166         Microsoft::WRL::ComPtr<IDXGIFactory4> mDXGIFactory;
167
168         Microsoft::WRL::ComPtr<ID3D12Fence> mFence;
169         UINT64 mFenceValue;
170         UINT64 mCurrentFrameFenceValue[NUM_OF_FRAMES];
```

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

7.11 FADirectXException.h

```
1 #pragma once
2
7 #include <wrl.h>
8 #include <dxgidebug.h>
9 #include <comdef.h>
10 #include <string>
11 #include <sstream>
12 #include <vector>
13
18 inline std::wstring AnsiToWString(const std::string& str)
19 {
20     WCHAR buffer[2048];
21     MultiByteToWideChar(CP_ACP, 0, str.c_str(), -1, buffer, 1024);
22     return std::wstring(buffer);
23 }
24
28 class DirectXException
29 {
30 public:
31
39     DirectXException(HRESULT hr, const std::wstring& functionName, const std::wstring& fileName, int
       lineNumber);
40
43     std::wstring ErrorMsg() const;
44
45 private:
46     HRESULT errorCode;
47     std::wstring functionName;
48     std::wstring fileName;
49     int lineNumber;
50     Microsoft::WRL::ComPtr<IDXGIInfoQueue> mInfoQueue;
51 };
52
55 #ifndef ThrowIfFailed
56 #define ThrowIfFailed(x)                                                    \
57 {                                                                           \
58 HRESULT hr = (x);                                                       \
59 std::wstring filename(AnsiToWString(__FILE__));                         \
60 if (FAILED(hr)) { throw DirectXException(hr, L#x, filename, __LINE__); }    \
61 }
62 #endif
63
64
67 inline void CreateInfoQueue(Microsoft::WRL::ComPtr<IDXGIInfoQueue>& infoQueue)
68 {
69 #if defined(_DEBUG) || defined(DEBUG)
70     //define function signature
```

```
71      typedef HRESULT(WINAPI* dxgiDebugInterface)(REFIID, void**);
72
73      //Get a handle to the dll file
74      HMODULE dxgiDebugHandle;
75      GetModuleHandleEx(GET_MODULE_HANDLE_EX_FLAG_UNCHANGED_REFCOUNT, L"Dxgidebug.dll", &dxgiDebugHandle);
76
77      //get the address of the function DXGIGetDebugInterface in the dll file
78      dxgiDebugInterface DXGIGetDebugInterface = (dxgiDebugInterface)GetProcAddress(dxgiDebugHandle,
    "DXGIGetDebugInterface");
79      if (DXGIGetDebugInterface == nullptr)
80      {
81          exit(-1);
82      }
83
84      //create a DXGIInfoQueue object.
85      DXGIGetDebugInterface(IID_PPV_ARGS(&infoQueue));
86  #endif
87  }
88
89
90
93  inline std::wstring ErrorMessage(HRESULT errorCode, const std::wstring& functionName, const std::wstring&
    filename, int lineNumber,
94      const Microsoft::WRL::ComPtr<IDXGIInfoQueue>& infoQueue)
95  {
96      //the _com_error class lets us retrieve the error message associated with the HRESULT error code
97      _com_error error(errorCode);
98      std::wstring msg = error.ErrorMessage();
99
100     //Get the hex value of the error code
101     std::stringstream ss;
102     ss « std::hex « errorCode;
103     std::wstring hrHex{ AnsiToWString(ss.str()) };
104
105     std::wstring eCode(std::to_wstring(errorCode));
106
107
108     std::wstring errorMessage{ L"File Name:  " + filename + L"\n\n" + L"Function Name:  " + functionName
    + L"\n\n" +
109         L"Line Number:  " + std::to_wstring(lineNumber) + L"\n\n" + L"Error Code:  " + eCode +
110         L"(0x" + hrHex + L")" + L"\n\n" + L"Error Code Description:  " + msg };
111
112     std::vector<std::wstring> messages;
113
114     if (infoQueue != nullptr)
115     {
116         //Get the number of messages in the queue.
117         UINT64 numOfMessages = infoQueue->GetNumStoredMessages(DXGI_DEBUG_ALL);
118
119         for (UINT64 i = 0; i < numOfMessages; ++i)
120         {
121             //Get the length of the current message.
122             SIZE_T messageLength{ 0 };
123             infoQueue->GetMessage(DXGI_DEBUG_ALL, i, nullptr, &messageLength);
124
125             //Allocate enough memory to store the message.
126             std::unique_ptr<unsigned char[]> bytes = std::make_unique<unsigned char[]>(messageLength);
127             DXGI_INFO_QUEUE_MESSAGE* pMsg = (DXGI_INFO_QUEUE_MESSAGE*)bytes.get();
128
129             //Retrieve the message.  It will be stored in pMsg.
130             infoQueue->GetMessage(DXGI_DEBUG_ALL, i, pMsg, &messageLength);
131
132             //Store the message.
133             std::string tempMessage{ pMsg->pDescription };
134             messages.emplace_back(AnsiToWString(tempMessage));
135         }
136     }
137
138     for (int i = 0; i < messages.size(); ++i)
139     {
140         errorMessage += L"\n";
141         errorMessage += messages[i];
142     }
143
144     return errorMessage;
145 }
146
149 #ifndef ExitIfFailed
150 #define ExitIfFailed(x)                                                                          \
151 {                                                                                                \
152 HRESULT hr = (x);                                                                          \
153 if (FAILED(hr))                                                                            \
154 {                                                                                          \
155 Microsoft::WRL::ComPtr<IDXGIInfoQueue> infoQueue;                                    \
156 CreateInfoQueue(infoQueue);                                                           \
157 std::wstring filename(AnsiToWString(__FILE__));                                       \
158 std::wstring errMsg = ErrorMessage(hr, L#x, filename, __LINE__, infoQueue);           \
```

```
159 MessageBox(nullptr, errMsg.c_str(), L"DirectX Error", MB_OK);                                    \
160         exit(-1);                                                                                \
161     }                                                                                            \
162 }
163 #endif
```

7.12 FADrawArgumentsStructure.h

```
1 #pragma once
2
3 #include <string>
4 #include <d3dcommon.h>
5
6 namespace FADrawArguments
7 {
11     struct DrawArguments
12     {
13         unsigned int indexCount = 0;
14         unsigned int locationOfFirstIndex = 0;
15         int indexOfFirstVertex = 0;
16         unsigned int indexOfConstantData = 0;
17         unsigned int rootParameterIndex = 0;
18         std::wstring constantBufferKey = L"";
19         D3D_PRIMITIVE_TOPOLOGY primtive = D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST;
20     };
21 }
```

7.13 FAMultiSampling.h

```
1 #pragma once
2
3 #include <wrl.h>
4 #include "d3dx12.h"
5 #include "FABuffer.h"
6
7 namespace FARender
8 {
13     class MultiSampling
14     {
15     public:
16
17         MultiSampling(const MultiSampling&) = delete;
18         MultiSampling& operator=(const MultiSampling&) = delete;
19
24         MultiSampling();
25
35         MultiSampling(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
36             DXGI_FORMAT rtFormat, unsigned int sampleCount);
37
47         void CheckMultiSamplingSupport(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
48             DXGI_FORMAT rtFormat, unsigned int sampleCount);
49
52         const Microsoft::WRL::ComPtr<ID3D12Resource>& GetRenderTargetBuffer();
53
56         DXGI_FORMAT GetRenderTargetFormat();
57
60         DXGI_FORMAT GetDepthStencilFormat();
61
64         void ReleaseBuffers();
65
75         void CreateRenderTargetBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
76             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int
    indexOfWhereToStoreView, unsigned int rtvSize,
77             unsigned int width, unsigned int height);
78
88         void CreateDepthStencilBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
89             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int
    indexOfWhereToStoreView, unsigned int dsvSize,
90             unsigned int width, unsigned int height);
91
96         void Transition(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList,
97             D3D12_RESOURCE_STATES before, D3D12_RESOURCE_STATES after);
98
107         void ClearRenderTargetBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
    commandList,
108             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int indexOfView,
    unsigned int rtvSize,
109             const float* clearValue);
110
```

```
119         void ClearDepthStencilBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
    commandList,
120             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int indexOfView,
    unsigned int dsvSize,
121             float clearValue);
122
123     private:
124         RenderTargetBuffer mMSAARenderTargetBuffer;
125         DepthStencilBuffer mMSAADepthStencilBuffer;
126         unsigned int mSampleCount;
127     };
128 }
```

7.14 C:/Users/Work/Desktop/First Game Engine/First-Game-Engine/FA Rendering Engine/Header Files/FAProjection.h File Reference

File that has namespace FAProjection. Within the namespace is the interface IProjection and class Perspective↩ Projection.

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

Classes

- class FAProjection::IProjection

  An interface for projections.
- class FAProjection::PerspectiveProjection

  A class for doing perspective projection.

Namespaces

- namespace FAProjection

  Within the namespace is the interface IProjection and class PerspectiveProjection.

7.14.1 Detailed Description

File that has namespace FAProjection. Within the namespace is the interface IProjection and class Perspective↩ Projection.

7.15 FAProjection.h

Go to the documentation of this file.

```
1 #pragma once
2
7 #include "FAMathEngine.h"
8
12 namespace FAProjection
13 {
17     class IProjection
18     {
19     public:
20
23         IProjection();
24
27         const FAMath::Matrix4x4& GetProjectionMatrix() const;
```

```
28
29          virtual void UpdateProjectionMatrix() = 0;
30
31      protected:
32          FAMath::Matrix4x4 mProjectionMatrix;
33          bool mUpdateProjectionMatrix;
34      };
35
39      class PerspectiveProjection :  public IProjection
40      {
41      public:
42
45          PerspectiveProjection();
46
54          PerspectiveProjection(float znear, float zfar, float vFov, float aspectRatio);
55
63          void SetProperties(float znear, float zfar, float vFov, float aspectRatio);
64
67          float GetNear() const;
68
71          float GetFar() const;
72
75          float GetVerticalFov() const;
76
79          float GetAspectRatio() const;
80
83          void SetNear(float znear);
84
87          void SetFar(float zfar);
88
91          void SetVerticalFov(float fov);
92
95          void SetAspectRatio(float ar);
96
99          void UpdateProjectionMatrix() override final;
100
101     private:
102         //frustrum properties
103         float mNear;
104         float mFar;
105         float mVerticalFov;
106         float mAspectRatio;
107     };
108 }
```

7.16 C:/Users/Work/Desktop/First Game Engine/First-Game-Engine/FA Rendering Engine/Header Files/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"
#include <string_view>
```

Classes

- class FARender::RenderScene

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

Namespaces

- namespace FARender

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

7.16.1 Detailed Description

File has class RenderScene under namespace FARender.

7.17 FARenderScene.h

Go to the documentation of this file.

```
1 #pragma once
2
7 #include <d3dcompiler.h>
8 #include <unordered_map>
9 #include "FADeviceResources.h"
10 #include "FABuffer.h"
11 #include "FAColor.h"
12 #include <string_view>
13
14 namespace FARender
15 {
20     class RenderScene
21     {
22     public:
23
24
      //-------------------------------------------------------------------------------------------------------------------------------
25         //CONSTRUCTORS
26
27         //No copying
28         RenderScene(const RenderScene&) = delete;
29         RenderScene operator=(const RenderScene&) = delete;
30
31         /*@brief Creates a RenderScene object.  Does not create the necessary resources to render a
     scene.
32 * Call the function CreateDeviceResources() to initialize all necessary resources.
33 */
34         RenderScene();
35
36         /*@brief Initializes all necessary resources.
37 *
38 * @param[in] width The width of a window.
39 * @param[in] height The height of a window.
40 * @param[in] windowHandle A handle to a window.
41 * @param[in, optional] isMSAAEnabled Pass in true if you want to have MSAA enabled, false otherwise.
42 * @param[in, optional] isTextEnabled Pass in true if you want to have text enabled, false otherwise.
43 */
44         RenderScene(unsigned int width, unsigned int height, HWND windowHandle,
45             bool isMSAAEnabled = false, bool isTextEnabled = false);
46
47         /*@brief Initializes all necessary resources.
48 *
49 * @param[in] width The width of a window.
50 * @param[in] height The height of a window.
51 * @param[in] windowHandle A handle to a window.
52 * @param[in, optional] isMSAAEnabled Pass in true if you want to have MSAA enabled, false otherwise.
53 * @param[in, optional] isTextEnabled Pass in true if you want to have text enabled, false otherwise.
54 */
55         void CreateDeviceResources(unsigned int width, unsigned int height, HWND windowHandle,
56             bool isMSAAEnabled = false, bool isTextEnabled = false);
57
58
      //-------------------------------------------------------------------------------------------------------------------------------
59
60
61
62
63
64
      //-------------------------------------------------------------------------------------------------------------------------------
65         //SHADER FUNCTIONS
66
72         void LoadShader(unsigned int shaderKey, std::wstring_view filename);
73
81         void CompileShader(unsigned int shaderKey, std::wstring_view filename,
82             std::string_view entryPointName, std::string_view target);
83
88         void RemoveShader(unsigned int shaderKey);
89
90
91
97         void LoadShader(std::wstring_view shaderKey, std::wstring_view filename);
```

```
98
106         void CompileShader(std::wstring_view shaderKey, std::wstring_view filename,
107             std::string_view entryPointName, std::string_view target);
108
113         void RemoveShader(std::wstring_view shaderKey);
114
115
    //------------------------------------------------------------------------------------------------------------------
116
117
118
119
120
121
    //------------------------------------------------------------------------------------------------------------------
122         //INPUT ELEMENT DESCRIPTION FUNCTIONS
123
135         void CreateInputElementDescription(unsigned int key, const char* semanticName, unsigned int
    semanticIndex,
136             DXGI_FORMAT format, unsigned int inputSlot, unsigned int byteOffset,
137             D3D12_INPUT_CLASSIFICATION inputSlotClass = D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA,
138             unsigned int instanceStepRate = 0);
139
140
141
153         void CreateInputElementDescription(std::wstring_view key, const char* semanticName, unsigned int
    semanticIndex,
154             DXGI_FORMAT format, unsigned int inputSlot, unsigned int byteOffset,
155             D3D12_INPUT_CLASSIFICATION inputSlotClass = D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA,
156             unsigned int instanceStepRate = 0);
157
158
    //------------------------------------------------------------------------------------------------------------------
159
160
161
162
163
164
    //------------------------------------------------------------------------------------------------------------------
165         //ROOT PARAMETER FUNCTIONS
166
172         void CreateRootDescriptor(unsigned int rootParameterKey, unsigned int shaderRegister);
173
188         void CreateDescriptorRange(unsigned int descriptorRangeKey,
189             D3D12_DESCRIPTOR_RANGE_TYPE type, unsigned int numDescriptors, unsigned int shaderRegister,
    unsigned int registerSpace,
190             unsigned int offset);
191
197         void CreateDescriptorTable(unsigned int rootParameterKey, unsigned int descriptorRangeKey);
198
205         void CreateRootConstants(unsigned int rootParameterKey, unsigned int shaderRegister, unsigned
    int numValues);
206
207
208
214         void CreateRootDescriptor(std::wstring_view rootParameterKey, unsigned int shaderRegister);
215
230         void CreateDescriptorRange(std::wstring_view descriptorRangeKey,
231             D3D12_DESCRIPTOR_RANGE_TYPE type, unsigned int numDescriptors, unsigned int shaderRegister,
    unsigned int registerSpace,
232             unsigned int offset);
233
239         void CreateDescriptorTable(std::wstring_view rootParameterKey, unsigned int descriptorRangeKey);
240
247         void CreateRootConstants(std::wstring_view rootParameterKey, unsigned int shaderRegister,
    unsigned int numValues);
248
249
    //------------------------------------------------------------------------------------------------------------------
250
251
252
253
254
255
    //------------------------------------------------------------------------------------------------------------------
256         //ROOT SIGNATURE FUNCTIONS
257
266         void CreateRootSignature(unsigned int rootSigKey, unsigned int rootParametersKey);
267
278         void CreateRootSignature(unsigned int rootSigKey, unsigned int rootParametersKey,
279             unsigned int staticsSamplerKey);
280
290         void CreateStaticSampler(unsigned int staticSamplerKey, D3D12_FILTER filter,
291             D3D12_TEXTURE_ADDRESS_MODE u, D3D12_TEXTURE_ADDRESS_MODE v, D3D12_TEXTURE_ADDRESS_MODE w,
    unsigned int shaderRegister);
```

```
292
293
294
303         void CreateRootSignature(std::wstring_view rootSigKey, std::wstring_view rootParametersKey);
304
315         void CreateRootSignature(std::wstring_view rootSigKey, std::wstring_view rootParametersKey,
316             std::wstring_view staticsSamplerKey);
317
327         void CreateStaticSampler(std::wstring_view staticSamplerKey, D3D12_FILTER filter,
328             D3D12_TEXTURE_ADDRESS_MODE u, D3D12_TEXTURE_ADDRESS_MODE v, D3D12_TEXTURE_ADDRESS_MODE w,
    unsigned int shaderRegister);
329
330
    //-----------------------------------------------------------------------------------------------------------------
331
332
333
334
335
336
    //-----------------------------------------------------------------------------------------------------------------
337         //PIPELINE STATE OBJECT FUNCTIONS
338
360         void CreatePSO(unsigned int psoKey, D3D12_FILL_MODE fillMode, BOOL enableMultisample,
361             unsigned int vsKey, unsigned int psKey, unsigned int inputElementDescriptionsKey,
362             unsigned int rootSigKey,
363             const D3D12_PRIMITIVE_TOPOLOGY_TYPE& primitiveType, UINT sampleCount = 1);
364
371         void LinkPSOAndRootSignature(unsigned int psoKey, unsigned int rootSigKey);
372
373
374
396         void CreatePSO(std::wstring_view psoKey, D3D12_FILL_MODE fillMode, BOOL enableMultisample,
397             std::wstring_view vsKey, std::wstring_view psKey, std::wstring_view
    inputElementDescriptionsKey,
398             std::wstring_view rootSigKey,
399             const D3D12_PRIMITIVE_TOPOLOGY_TYPE& primitiveType, UINT sampleCount = 1);
400
407         void LinkPSOAndRootSignature(std::wstring_view psoKey, std::wstring_view rootSigKey);
408
409
    //-----------------------------------------------------------------------------------------------------------------
410
411
412
413
414
415
    //-----------------------------------------------------------------------------------------------------------------
416         //STATIC BUFFER FUNCTIONS
417
429         void CreateStaticBuffer(unsigned int staticBufferKey, const void* data, unsigned numBytes,
    unsigned int stride);
430
442         void CreateStaticBuffer(unsigned int staticBufferKey, const void* data, unsigned numBytes,
    DXGI_FORMAT format);
443
460         void CreateStaticBuffer(unsigned int staticBufferKey, const wchar_t* filename, unsigned int
    texType, unsigned int index);
461
472         void LinkStaticBuffer(unsigned int bufferType, unsigned int staticBufferKey);
473
474
475
487         void CreateStaticBuffer(std::wstring_view staticBufferKey, const void* data, unsigned numBytes,
    unsigned int stride);
488
500         void CreateStaticBuffer(std::wstring_view staticBufferKey, const void* data, unsigned numBytes,
    DXGI_FORMAT format);
501
518         void CreateStaticBuffer(std::wstring_view staticBufferKey, const wchar_t* filename, unsigned int
    texType, unsigned int index);
519
530         void LinkStaticBuffer(unsigned int bufferType, std::wstring_view staticBufferKey);
531
532
    //-----------------------------------------------------------------------------------------------------------------
533
534
535
536
537
538
    //-----------------------------------------------------------------------------------------------------------------
539         //DYNAMIC BUFFER FUNCTIONS
540
556         void CreateDynamicBuffer(unsigned int dynamicBufferKey, unsigned numBytes, const void* data,
```

```
    unsigned int stride);
557
573         void CreateDynamicBuffer(unsigned int dynamicBufferKey, unsigned numBytes, const void* data,
    DXGI_FORMAT format);
574
593         void LinkDynamicBuffer(unsigned int bufferType, unsigned int dynamicBufferKey, unsigned int
    indexConstantData = 0,
594             unsigned int rootParameterIndex = 0);
595
603         void CopyDataIntoDynamicBuffer(unsigned int dynamicBufferKey, unsigned int index, const void*
    data, UINT64 numOfBytes);
604
605
606
622         void CreateDynamicBuffer(std::wstring_view dynamicBufferKey, unsigned numBytes, const void*
    data, unsigned int stride);
623
639         void CreateDynamicBuffer(std::wstring_view dynamicBufferKey, unsigned numBytes, const void*
    data, DXGI_FORMAT format);
640
659         void LinkDynamicBuffer(unsigned int bufferType, std::wstring_view dynamicBufferKey, unsigned int
    indexConstantData = 0,
660             unsigned int rootParameterIndex = 0);
661
669         void CopyDataIntoDynamicBuffer(std::wstring_view dynamicBufferKey, unsigned int index, const
    void* data, UINT64 numOfBytes);
670
671
    //------------------------------------------------------------------------------------------------------------------------
672
673
674
675
676
677
    //------------------------------------------------------------------------------------------------------------------------
678         //TEXTURE FUNCTIONS
679
683         void CreateTextureViewHeap(unsigned int numDescriptors);
684
687         void LinkTextureViewHeap();
688
694         void LinkTexture(unsigned int rootParameterIndex);
695
703         void LinkTexture(unsigned int rootParameterIndex, unsigned int textureViewIndex);
704
705
    //------------------------------------------------------------------------------------------------------------------------
706
707
    //------------------------------------------------------------------------------------------------------------------------
708         //RENDER OBJECTS FUNCTONS
709
716         void BeforeRenderObjects(bool isMSAAEnabled = false);
717
736         void RenderObject(unsigned int indexCount, unsigned int locationFirstIndex, int
    indexOfFirstVertex,
737             D3D_PRIMITIVE_TOPOLOGY primitive);
738
744         void AfterRenderObjects(bool isMSAAEnabled = false, bool isTextEnabled = false);
745
746
    //------------------------------------------------------------------------------------------------------------------------
747
748
749
750
751
752
    //------------------------------------------------------------------------------------------------------------------------
753         //RENDER TEXT FUNCTIONS
754
758         void BeforeRenderText();
759
782         void RenderText(const FAMath::Vector4D& textLocation, const FAColor::Color& textColor, float
    textSize,
783             const std::wstring& textString, DWRITE_PARAGRAPH_ALIGNMENT alignment =
    DWRITE_PARAGRAPH_ALIGNMENT_CENTER);
784
789         void AfterRenderText();
790
791
    //------------------------------------------------------------------------------------------------------------------------
792
793
794
795
```

```
796
797
    //-----------------------------------------------------------------------------------------------------------------
798         //MISCELLANEOUS FUNCTIONs
799
804         void AfterRender();
805
808         void ExecuteAndFlush();
809
818         void Resize(unsigned int width, unsigned int height, HWND windowHandle, bool isMSAAEnabled =
    false, bool isTextEnabled = false);
819
831         void SetConstants(unsigned int rootParameterIndex, unsigned int numValues, void* data, unsigned
    int index);
832
833
    //-----------------------------------------------------------------------------------------------------------------
834
835       private:
836
837         //The device resources object that all RenderScene objects share.
838         DeviceResources* mDeviceResources;
839
840
841
    //-----------------------------------------------------------------------------------------------------------------
842         //SHADER HASH MAPS
843
844         //Stores all of the shaders for this scene.
845         std::unordered_map<unsigned int, Microsoft::WRL::ComPtr<ID3DBlob» mShaders;
846
847         //Stores all of the shaders for this scene.
848         std::unordered_map<std::wstring_view, Microsoft::WRL::ComPtr<ID3DBlob» mShadersStr;
849
850
    //-----------------------------------------------------------------------------------------------------------------
851
852
853
854
855
856
    //-----------------------------------------------------------------------------------------------------------------
857         //INPUT ELEMENT DESCRIPTION HASH MAPS
858
859         //Stores input element descriptions for a set of shaders.
860         std::unordered_map<unsigned int, std::vector<D3D12_INPUT_ELEMENT_DESC»
    mInputElementDescriptions;
861
862         //Stores input element descriptions for a set of shaders.
863         std::unordered_map<std::wstring_view, std::vector<D3D12_INPUT_ELEMENT_DESC»
    mInputElementDescriptionsStr;
864
865
    //-----------------------------------------------------------------------------------------------------------------
866
867
868
869
870
871
    //-----------------------------------------------------------------------------------------------------------------
872         //ROOT PARAMETER HASH MAPS
873
874         //Stores root parameters for root signatures.
875         std::unordered_map<unsigned int, std::vector<D3D12_ROOT_PARAMETER» mRootParameters;
876
877         //Stores descriptor ranges for descriptor tables.
878         std::unordered_map<unsigned int, std::vector<D3D12_DESCRIPTOR_RANGE» mDescriptorRanges;
879
880         //Stores root parameters for root signatures.
881         std::unordered_map<std::wstring_view, std::vector<D3D12_ROOT_PARAMETER» mRootParametersStr;
882
883         //Stores descriptor ranges for descriptor tables.
884         std::unordered_map<std::wstring_view, std::vector<D3D12_DESCRIPTOR_RANGE» mDescriptorRangesStr;
885
886
    //-----------------------------------------------------------------------------------------------------------------
887
888
889
890
891
892
    //-----------------------------------------------------------------------------------------------------------------
893         //ROOT SIGNATURE HASH MAPS
894
```

```
895         //The root signatures for the scene.
896         //Describes all of the constant data that is expected in a set of shaders.
897         //Microsoft::WRL::ComPtr<ID3D12RootSignature> mRootSignature;
898         std::unordered_map<unsigned int, Microsoft::WRL::ComPtr<ID3D12RootSignature» mRootSignatures;
899 
900         //Stores static samplers.
901         std::unordered_map<unsigned int, std::vector<D3D12_STATIC_SAMPLER_DESC» mStaticSamplers;
902 
903         //The root signatures for the scene.
904         //Describes all of the constant data that is expected in a set of shaders.
905         //Microsoft::WRL::ComPtr<ID3D12RootSignature> mRootSignature;
906         std::unordered_map<std::wstring_view, Microsoft::WRL::ComPtr<ID3D12RootSignature»
    mRootSignaturesStr;
907 
908         //Stores static samplers.
909         std::unordered_map<std::wstring_view, std::vector<D3D12_STATIC_SAMPLER_DESC» mStaticSamplersStr;
910 
911 
    //-----------------------------------------------------------------------------------------------------------------------
912 
913 
914 
915 
916 
917 
    //-----------------------------------------------------------------------------------------------------------------------
918         //PIPELINE STATE OBJECT HASH MAPS
919 
920         //Stores pipeline state objects.
921         std::unordered_map<unsigned int, Microsoft::WRL::ComPtr<ID3D12PipelineState» mPSOs;
922 
923         //Stores pipeline state objects.
924         std::unordered_map<std::wstring_view, Microsoft::WRL::ComPtr<ID3D12PipelineState» mPSOsStr;
925 
    //-----------------------------------------------------------------------------------------------------------------------
926 
927 
928 
929 
930 
931 
    //-----------------------------------------------------------------------------------------------------------------------
932         //STATIC BUFFER HASH MAPS
933 
934         //Stores data that will not be updated on a per-frame basis.
935         std::unordered_map<unsigned int, StaticBuffer> mStaticBuffers;
936 
937         //Stores data that will not be updated on a per-frame basis.
938         std::unordered_map < std::wstring_view, StaticBuffer> mStaticBuffersStr;
939 
940 
    //-----------------------------------------------------------------------------------------------------------------------
941 
942 
943 
944 
945 
946 
    //-----------------------------------------------------------------------------------------------------------------------
947         //DYNAMIC BUFFER HASH MAPS
948 
949         //Stores data that will be updated on a per-frame basis.
950         //We can't update a dynamic buffer until the GPU
951         //is done executing all the commands that reference it, so each frame needs its own dynamic
    buffer.
952         std::unordered_map<unsigned int, DynamicBuffer[DeviceResources::NUM_OF_FRAMES]> mDynamicBuffers;
953 
954         //Stores data that will be updated on a per-frame basis.
955         //We can't update a dynamic buffer until the GPU
956         //is done executing all the commands that reference it, so each frame needs its own dynamic
    buffer.
957         std::unordered_map<std::wstring_view, DynamicBuffer[DeviceResources::NUM_OF_FRAMES]>
    mDynamicBuffersStr;
958 
959 
    //-----------------------------------------------------------------------------------------------------------------------
960 
961 
962         //Used to store descriptors of textures.
963         Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mTextureViewHeap;
964 
965     };
966 }
```

7.18 FASwapChain.h

```
1 #pragma once
2
3 #include <wrl.h>
4 #include "d3dx12.h"
5 #include <dxgi1_4.h>
6 #include <vector>
7 #include <memory>
8 #include "FABuffer.h"
9
10 namespace FARender
11 {
16     class SwapChain
17     {
18     public:
19
20         //No copying
21         SwapChain(const SwapChain&) = delete;
22         SwapChain& operator=(const SwapChain&) = delete;
23
27         SwapChain();
28
38         SwapChain(const Microsoft::WRL::ComPtr<IDXGIFactory4>& dxgiFactory,
39             const Microsoft::WRL::ComPtr<ID3D12CommandQueue>& commandQueue, HWND windowHandle,
40             DXGI_FORMAT rtFormat = DXGI_FORMAT_R8G8B8A8_UNORM, DXGI_FORMAT dsFormat =
    DXGI_FORMAT_D24_UNORM_S8_UINT,
41             unsigned int numRenderTargetBuffers = 2);
42
52         void CreateSwapChain(const Microsoft::WRL::ComPtr<IDXGIFactory4>& dxgiFactory,
53             const Microsoft::WRL::ComPtr<ID3D12CommandQueue>& commandQueue, HWND windowHandle,
54             DXGI_FORMAT rtFormat = DXGI_FORMAT_R8G8B8A8_UNORM, DXGI_FORMAT dsFormat =
    DXGI_FORMAT_D24_UNORM_S8_UINT,
55             unsigned int numRenderTargetBuffers = 2);
56
59         const Microsoft::WRL::ComPtr<ID3D12Resource>& GetCurrentBackBuffer() const;
60
63         unsigned int GetNumRenderTargetBuffers() const;
64
67         unsigned int GetCurrentBackBufferIndex() const;
68
71         DXGI_FORMAT GetBackBufferFormat() const;
72
75         DXGI_FORMAT GetDepthStencilFormat() const;
76
77         const std::vector<std::unique_ptr<RenderTargetBuffer>>& GetRenderTargetBuffers();
78
81         void ReleaseBuffers();
82
92         void CreateRenderTargetBuffersAndViews(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
93             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int index,
94             unsigned int rtvSize, unsigned width, unsigned height);
95
105         void CreateDepthStencilBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
106             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int index, unsigned
    int dsvSize,
107             unsigned int width, unsigned int height);
108
119         void ClearCurrentBackBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
    commandList,
120             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int indexOfFirstView,
    unsigned int rtvSize,
121             const float* backBufferClearValue);
122
133         void ClearDepthStencilBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
    commandList,
134             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int indexOfView,
    unsigned int dsvSize,
135             float clearValue);
136
141         void Transition(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList,
142             D3D12_RESOURCE_STATES before, D3D12_RESOURCE_STATES after);
143
146         void Present();
147
148     private:
149         unsigned int mNumRenderTargetBuffers;
150         unsigned int mCurrentBackBufferIndex;
151
152         Microsoft::WRL::ComPtr<IDXGISwapChain1> mSwapChain;
153         std::vector<std::unique_ptr<RenderTargetBuffer>> mRenderTargetBuffers;
154
155         DepthStencilBuffer mDepthStencilBuffer;
156     };
157 }
```

7.19 C:/Users/Work/Desktop/First Game Engine/First-Game-Engine/FA Rendering Engine/Header Files/FAText.h File Reference

File has class Text under namespace FARender.

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

Classes

- class FARender::Text

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

Namespaces

- namespace FARender

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

7.19.1 Detailed Description

File has class Text under namespace FARender.

7.20 FAText.h

Go to the documentation of this file.

```
1 #pragma once
2
7 #include <string>
8 #include "FAColor.h"
9
10 namespace FARender
11 {
16     class Text
17     {
18     public:
19
20         Text() = default;
21
32         Text(const FAMath::Vector4D& textLocation, const std::wstring& textString, float textSize, const
      FAColor::Color& textColor);
33
36         const FAMath::Vector4D& GetTextLocation() const;
37
40         const std::wstring& GetTextString() const;
41
44         float GetTextSize() const;
45
48         const FAColor::Color& GetTextColor() const;
49
52         void SetTextSize(float textSize);
53
56         void SetTextColor(const FAColor::Color& textColor);
57
60         void SetTextString(const std::wstring& textString);
61
64         void SetTextLocation(const FAMath::Vector4D& textLocation);
65
66     private:
67
68         FAMath::Vector4D mTextLocation;
69         std::wstring mText;
70         float mTextSize{ 0.0f };
71         FAColor::Color mTextColor;
72     };
73 }
```

7.21 FATextResources.h

```
1 #pragma once
2
3 #include <wrl.h>
4 #include <d3d11.h>
5 #include <d3d11on12.h>
6 #include <d2d1_3.h>
7 #include <dwrite.h>
8 #include <vector>
9 #include <memory>
10 #include "FABuffer.h"
11
12 namespace FARender
13 {
17     class TextResources
18     {
19     public:
20         TextResources() = default;
21
28         TextResources(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
29             const Microsoft::WRL::ComPtr<ID3D12CommandQueue>& commandQueue, unsigned int
    numSwapChainBuffers);
30
33         const Microsoft::WRL::ComPtr<ID2D1DeviceContext>& GetDirect2DDeviceContext() const;
34
37         const Microsoft::WRL::ComPtr<IDWriteFactory>& GetDirectWriteFactory() const;
38
41         void ResetBuffers();
42
48         void ResizeBuffers(const std::vector<std::unique_ptr<RenderTargetBuffer»& renderTargetBuffers,
    HWND windowHandle);
49
54         void BeforeRenderText(unsigned int currentBackBuffer);
55
60         void AfterRenderText(unsigned int currentBackBuffer);
61
62     private:
63         Microsoft::WRL::ComPtr<ID3D11Device> mDevice11;
64         Microsoft::WRL::ComPtr<ID3D11DeviceContext> mDevice11Context;
65         Microsoft::WRL::ComPtr<ID3D11On12Device> mDevice11on12;
66
67         Microsoft::WRL::ComPtr<ID2D1Device2> mDirect2DDevice;
68         Microsoft::WRL::ComPtr<ID2D1Factory3> mDirect2DFactory;
69         Microsoft::WRL::ComPtr<ID2D1DeviceContext> mDirect2DDeviceContext;
70
71         Microsoft::WRL::ComPtr<IDWriteFactory> mDirectWriteFactory;
72
73         std::vector<Microsoft::WRL::ComPtr<ID3D11Resource» mWrappedBuffers;
74         std::vector<Microsoft::WRL::ComPtr<ID2D1Bitmap1» mDirect2DBuffers;
75         std::vector<Microsoft::WRL::ComPtr<IDXGISurface» mSurfaces;
76     };
77 }
```

7.22 C:/Users/Work/Desktop/First Game Engine/First-Game-Engine/FA Rendering Engine/Header Files/FATime.h File Reference

File that has namespace FATime. Withn the namespace is the class Time.

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

Classes

- class FATime::Time

Namespaces

- namespace FATime

  Has Time class.

7.22.1 Detailed Description

File that has namespace FATime. Withn the namespace is the class Time.

7.23 FATime.h

Go to the documentation of this file.

```
1 #pragma once
2
7 #include <Windows.h>
8
12 namespace FATime
13 {
14     class Time
15     {
16     public:
17         Time();
18
19         float GetPrevTime() const;
20         float GetDeltaTime() const;
21         bool GetIsTimeStopped()const ;
22
23         void Reset();
24         void Tick();
25
26         void Start();
27         void Stop();
28
29     private:
30         __int64 mPrevTime;              //t0
31         __int64 mCurrentTime;           //t1
32         double mDeltaTime;              //t1 - t0
33         double mSecondsPerCount;
34         bool mStopped;
35     };
36 }
```

7.24 C:/Users/Work/Desktop/First Game Engine/First-Game-Engine/FA Rendering Engine/Header Files/FAWindow.h File Reference

File that has namespace FAWindow. Withn the namespace is the class Window.

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

Classes

- class FAWindow::Window

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

Namespaces

- namespace FAWindow

  Has Window class.

7.24.1 Detailed Description

File that has namespace FAWindow. Withn the namespace is the class Window.

7.25 FAWindow.h

Go to the documentation of this file.

```
1 #pragma once
2
7 #include <Windows.h>
8 #include <string>
9 #include <stdexcept>
10 #include "FAColor.h"
11
15 namespace FAWindow
16 {
22     class Window
23     {
24     public:
25
26         Window(const Window&) = delete;
27         Window& operator=(const Window&) = delete;
28
32         Window();
33
66         Window(const HINSTANCE& hInstance, WNDPROC windowProcedure, const FAColor::Color&
    backgroundColor,
67             const std::wstring& windowClassName, const std::wstring& windowName, unsigned int styles,
68             unsigned int x, unsigned int y, unsigned int width, unsigned int height, void* additionalData
    = nullptr);
69
100         Window(const HINSTANCE& hInstance, HWND parent, unsigned int identifier,
101             WNDPROC windowProcedure, const FAColor::Color& backgroundColor,
102             const std::wstring& windowClassName, const std::wstring& windowName, unsigned int styles,
103             unsigned int x, unsigned int y, unsigned int width, unsigned int height, void*
    additionalData = nullptr);
104
131         Window(const HINSTANCE& hInstance, HWND parent, unsigned int identifier,
132             const std::wstring& windowClassName,
133             const std::wstring& windowName, unsigned int styles,
134             unsigned int x, unsigned int y, unsigned int width, unsigned int height, void*
    additionalData = nullptr);
135
168         void CreateParentWindow(const HINSTANCE& hInstance, WNDPROC windowProcedure, const
    FAColor::Color& backgroundColor,
169             const std::wstring& windowClassName, const std::wstring& windowName, unsigned int styles,
170             unsigned int x, unsigned int y, unsigned int width, unsigned int height, void*
    additionalData = nullptr);
171
202         void CreateChildWindow(const HINSTANCE& hInstance, HWND parent, unsigned long long int
    identifier,
203             WNDPROC windowProcedure, const FAColor::Color& backgroundColor,
204             const std::wstring& windowClassName, const std::wstring& windowName, unsigned int styles,
205             unsigned int x, unsigned int y, unsigned int width, unsigned int height, void*
    additionalData = nullptr);
206
233         void CreateControlWindow(const HINSTANCE& hInstance, HWND parent, unsigned long long int
    identifier,
234             const std::wstring& windowClassName,
235             const std::wstring& windowName, unsigned int styles,
236             unsigned int x, unsigned int y, unsigned int width, unsigned int height, void*
    additionalData = nullptr);
237
240         HWND GetWindowHandle() const;
241
244         unsigned int GetWidth() const ;
245
248         unsigned int GetHeight() const;
249
252         unsigned int GetX() const;
253
256         unsigned int GetY() const;
257
260         void SetWidth(unsigned int width);
261
264         void SetHeight(unsigned int height);
265
268         void SetX(unsigned int x);
269
```

```
272         void SetY(unsigned int y);
273
274     private:
275         HWND mWindowHandle;
276
277         WNDCLASSEX mWindowClass;
278
279         unsigned int mX;
280         unsigned int mY;
281         unsigned int mWidth;
282         unsigned int mHeight;
283     };
284 }
```

Index