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

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Namespace Documentation

4.1 FACamera Namespace Reference

Has Camera class.

Classes

- class Camera

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

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

4.1.1 Detailed Description

Has Camera class.

4.2 FARender Namespace Reference

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

Classes

- class ConstantBuffer

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

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

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

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

- class IndexBuffer

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

  A wrapper for multisampling resources. Uses DirectD 12 API.
- class RenderScene

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

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

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

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

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

  This class stores vertices in a Direct3D 12 default buffer.

4.2.1 Detailed Description

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

4.3 FAWindow Namespace Reference

Has Window class.

Classes

- class Window

  The window class is used to make a Window using Windows API.

4.3.1 Detailed Description

Has Window class.

Chapter 5

Class Documentation

5.1 FACamera::Camera Class Reference

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

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

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

Public Member Functions

- Camera (vec3 cameraPosition=vec3(0.0f, 0.0f, 0.0f), vec3 x=vec3(1.0f, 0.0f, 0.0f), vec3 y=vec3(0.0f, 1.0f, 0.↩ 0f), vec3 z=vec3(0.0f, 0.0f, 1.0f), float znear=1.0f, float zfar=100.f, float aspectRatio=1.0f, float vFov=45.0f, float cameraVelocity=10.0f, float angularVelocity=0.25f)

  Constructor.
- const vec3 & GetCameraPosition () const

  Returns a constant reference to the position of the camera in world coordinates.
- const vec3 & GetX () const

  Returns a constant reference to the x-axis of the camera.
- const vec3 & GetY () const

  Returnsa constant reference to the y-axis of the camera.
- const vec3 & GetZ () const

  Returns a constant reference to the z-axis of the camera.
- const mat4 & GetViewTransformationMatrix () const

  Returns a constant reference to the view transformation matrix of this camera.
- float GetCameraVelocity () const

  Returns the camera's velocity.
- float GetAngularVelocity () const

  Returns the camera's angular velocity.
- void LookAt (vec3 cameraPosition, vec3 target, vec3 up)

  Defines the camera space using UVN.
- float GetZNear () const

  Returns the near value of the frustrum.
- float GetZFar () const

Returns the far value of the frustrum.

- float GetVerticalFov () const

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

- float GetAspectRatio () const

  Returns the aspect ratio of the frustrum.

- void SetCameraPosition (const vec3 &position)

  Sets the camera's position to the specified position.

- void SetX (const vec3 &x)

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

- void SetY (const vec3 &y)

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

- void SetZ (const vec3 &z)

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

- void SetCameraVelocity (float velocity)

  Sets the camera's velocity to the specified velocity.

- void SetAngularVelocity (float velcoity)

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

- void SetZNear (float znear)

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

- void SetZFar (float zfar)

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

- void SetVerticalFov (float fov)

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

- void SetAspectRatio (float ar)

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

- const mat4 & GetPerspectiveProjectionMatrix () const

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

- const mat4 & GetViewPerspectiveProjectionMatrix () const

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

- void UpdateViewMatrix ()

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

- void UpdatePerspectiveProjectionMatrix ()

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

- void UpdateViewPerspectiveProjectionMatrix ()

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

- void Left (float dt)

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

- void Right (float dt)

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

- void 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. 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.

- void MouseInput ()

  Rotates camera on mouse movement.

5.1.1 Detailed Description

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

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

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Camera()

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

Constructor.

Creates a new camera.
Sets the origin of the camera space to the given cameraPosition.
Sets the axis of the camera space to the given x, y and z vectors.
The origin and basis vectors of the camera space should be relative to world space.
Sets the frustum properties for perspective projection to the given znear, zar, aspectRatio and fov values.
vFov should be in degrees.
The constant velocity of the camera when moved is set to the given cameraVelocity; The angular velocity of the camera is set the to specified angularVelocity.

5.1.3 Member Function Documentation

5.1.3.1 Backward()

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

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

5.1.3.2 Down()

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

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

5.1.3.3 Foward()

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

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

5.1.3.4 GetAngularVelocity()

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

Returns the camera's angular velocity.

5.1.3.5 GetAspectRatio()

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

Returns the aspect ratio of the frustrum.

5.1.3.6 GetCameraPosition()

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

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

5.1.3.7 GetCameraVelocity()

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

Returns the camera's velocity.

5.1.3.8 GetPerspectiveProjectionMatrix()

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

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

5.1.3.9 GetVerticalFov()

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

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

5.1.3.10 GetViewPerspectiveProjectionMatrix()

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

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

5.1.3.11 GetViewTransformationMatrix()

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

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

5.1.3.12 GetX()

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

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

5.1.3.13 GetY()

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

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

5.1.3.14 GetZ()

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

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

5.1.3.15 GetZFar()

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

Returns the far value of the frustrum.

5.1.3.16 GetZNear()

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

Returns the near value of the frustrum.

5.1.3.17 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.

5.1.3.18 Left()

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

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

5.1.3.19 LookAt()

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

Defines the camera space using UVN.

5.1.3.20 MouseInput()

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

Rotates camera on mouse movement.

5.1.3.21 Right()

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

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

5.1.3.22 RotateCameraLeftRight()

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

Rotates the camera to look left and right.

5.1.3.23 RotateCameraUpDown()

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

Rotates the camera to look up and down.

5.1.3.24 SetAngularVelocity()

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

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

5.1.3.25 SetAspectRatio()

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

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

5.1.3.26 SetCameraPosition()

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

Sets the camera's position to the specified position.

5.1.3.27 SetCameraVelocity()

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

Sets the camera's velocity to the specified velocity.

5.1.3.28 SetVerticalFov()

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

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

5.1.3.29 SetX()

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

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

5.1.3.30 SetY()

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

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

5.1.3.31 SetZ()

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

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

5.1.3.32 SetZFar()

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

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

5.1.3.33 SetZNear()

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

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

5.1.3.34 Up()

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

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

5.1.3.35 UpdatePerspectiveProjectionMatrix()

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

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

5.1.3.36 UpdateViewMatrix()

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

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

5.1.3.37 UpdateViewPerspectiveProjectionMatrix()

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

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

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

- FACamera.h

5.2 FAColor::Color Class Reference

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

```
#include "FAColor.h"
```

Public Member Functions

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

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

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

  Returns the color.
- float GetRed () const

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

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

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

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

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

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

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

Sets the blue component to the specified float value.

- void SetAlpha (float a)

  Sets the alpha component to the specified float value.

- Color & operator+= (const Color &c)

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

- Color & operator-= (const Color &c)

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

- Color & operator*= (float k)

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

- Color & operator*= (const Color &c)

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

5.2.1 Detailed Description

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

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Color() [1/2]

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

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

5.2.2.2 Color() [2/2]

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

Overloaded Constructor. Initializes the color to the specified color.

5.2.3 Member Function Documentation

5.2.3.1 GetAlpha()

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

Returns the value of the alpha component.

5.2.3.2 GetBlue()

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

Returns the value of the green component.

5.2.3.3 GetColor()

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

Returns the color.

5.2.3.4 GetGreen()

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

Returns the value of the blue component.

5.2.3.5 GetRed()

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

Returns the value of the red component.

5.2.3.6 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.

5.2.3.7 operator*=() [2/2]

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

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

5.2.3.8 operator+=()

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

Adds this objects color to the specified color 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.

5.2.3.9 operator-=()

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

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

5.2.3.10 SetAlpha()

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

Sets the alpha component to the specified float value.

5.2.3.11 SetBlue()

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

Sets the blue component to the specified float value.

5.2.3.12 SetColor()

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

Sets the color to the specified color.

5.2.3.13 SetGreen()

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

Sets the green component to the specified float value.

5.2.3.14 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:

- FAColor.h

5.3 FARender::ConstantBuffer Class Reference

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

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

Public Member Functions

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

  Unmaps the pointer to the constant buffer.
- void CreateConstantBuffer (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const UINT &numOfBytes)

  Creates and maps the constant buffer. The number of bytes allocated should be a multiple of 256 bytes.
- void CreateConstantBufferView (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &cbvHeap, UINT cbvSize, UINT cBufferIndex, UINT cbvHeapIndex, UINT numBytes)

  Creates and maps the constant buffer view and stores it in the specified descriptor heap.
- void CopyData (UINT index, UINT byteSize, const void *data, UINT64 numOfBytes)

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

5.3.1 Detailed Description

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

5.3.2 Constructor & Destructor Documentation

5.3.2.1 ~ConstantBuffer()

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

Unmaps the pointer to the constant buffer.

5.3.3 Member Function Documentation

5.3.3.1 CopyData()

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

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

5.3.3.2 CreateConstantBuffer()

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

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

5.3.3.3 CreateConstantBufferView()

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

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

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

- FABuffer.h

5.4 FARender::DepthStencilBuffer Class Reference

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

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

Public Member Functions

- DepthStencilBuffer (DXGI_FORMAT format=DXGI_FORMAT_D24_UNORM_S8_UINT)

  Default Constructor.
- DXGI_FORMAT GetDepthStencilFormat () const

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

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

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

  Clears the depth stencil buffer with the specified clear value.

5.4.1 Detailed Description

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

5.4.2 Constructor & Destructor Documentation

5.4.2.1 DepthStencilBuffer()

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

Default Constructor.

5.4.3 Member Function Documentation

5.4.3.1 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.

5.4.3.2 CreateDepthStencilBufferAndView()

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

Creates the depth stencil buffer and view.

5.4.3.3 GetDepthStencilFormat()

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

Returns the format of the depth stencil buffer.

5.4.3.4 ResetBuffer()

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

Resest the depth stencil buffer.

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

- FABuffer.h

5.5 FARender::DeviceResources Class Reference

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

```
#include "FADeviceResources.h"
```

Public Member Functions

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

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

  Returns a constant reference to the ID3D12Device 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 GetCBVSize () const

  The size of a constant buffer view.
- unsigned int GetNumFrames () const

  Returns the number of frames.
- unsigned int GetCurrentFrame () const

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

  Returns a constant reference to the TextResources object.
- bool IsMSAAEnabled () const

  Returns true if MSAA is enabled, false otherwise.

- void DisableMSAA (unsigned int width, unsigned int height, HWND windowHandle)

  Disables MSAA.
- void EnableMSAA (unsigned int width, unsigned int height, HWND windowHandle)

  Enables MSAA.
- bool IsTextEnabled () const

  Returns true if text is enabled, false otherwise.
- void DisableText (unsigned int width, unsigned int height, HWND windowHandle)

  Disables text.
- void EnableText (unsigned int width, unsigned int height, HWND windowHandle)

  Enables text.
- void UpdateCurrentFrameFenceValue ()

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

  Synchronizes the CPU and GPU. Use this function to make sure all of the commands in command list are executed by the GPU before the CPU writes in new commands.
- void WaitForGPU () const

  Waits for the GPU to execute all of the commands of the current frame. Signal should have been called before this function is called.
- void Signal ()

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

  Call when the window gets resized. Call when you initialize your program.
- void RTBufferTransition ()

  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** ()
- 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, unsigned int numFrames)

  Call to make an object of DeviceResources. This only allows one instance to exist.

5.5.1 Detailed Description

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

5.5.2 Constructor & Destructor Documentation

5.5.2.1 ~DeviceResources()

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

Flushes the command queue.

5.5.3 Member Function Documentation

5.5.3.1 AfterTextDraw()

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

Executes the text commands.

5.5.3.2 BeforeTextDraw()

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

Prepares to render text.

5.5.3.3 DisableMSAA()

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

Disables MSAA.

5.5.3.4 DisableText()

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

Disables text.

5.5.3.5 EnableMSAA()

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

Enables MSAA.

5.5.3.6 EnableText()

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

Enables text.

5.5.3.7 Execute()

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

Executes the command list.

5.5.3.8 FlushCommandQueue()

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

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

5.5.3.9 GetBackBufferFormat()

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

Returns a constant reference to the back buffer format.

5.5.3.10 GetCBVSize()

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

The size of a constant buffer view.

5.5.3.11 GetCommandList()

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

Returns a constant reference to the ID3D12GraphicsCommandList objcet.

5.5.3.12 GetCurrentFrame()

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

Returns the current frame.

5.5.3.13 GetDepthStencilFormat()

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

Returns a constant reference to the depth stencil format.

5.5.3.14 GetDevice()

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

Returns a constant reference to the ID3D12Device objcet.

5.5.3.15 GetInstance()

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

Call to make an object of DeviceResources. This only allows one instance to exist.

5.5.3.16 GetNumFrames()

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

Returns the number of frames.

5.5.3.17 GetTextResources()

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

Returns a constant reference to the TextResources object.

5.5.3.18 IsMSAAEnabled()

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

Returns true if MSAA is enabled, false otherwise.

5.5.3.19 IsTextEnabled()

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

Returns true if text is enabled, false otherwise.

5.5.3.20 NextFrame()

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

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

5.5.3.21 Present()

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

Swaps the front and back buffers.

5.5.3.22 Resize()

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

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

5.5.3.23 RTBufferTransition()

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

Transistions the render target buffer.

5.5.3.24 Signal()

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

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

5.5.3.25 UpdateCurrentFrameFenceValue()

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

Updates the current frames fence value.

5.5.3.26 WaitForGPU()

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

Waits for the GPU to execute all of the commands of the current frame. Signal should have been called before this function is called.

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

- FADeviceResources.h

5.6 DirectXException Class Reference

Public Member Functions

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

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

- FADirectXException.h

5.7 FARender::DrawSettings Struct Reference

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

`#include "FARenderScene.h"`

Public Attributes

- Microsoft::WRL::ComPtr< ID3D12PipelineState > **pipelineState**
- Microsoft::WRL::ComPtr< ID3D12RootSignature > **rootSig**
- D3D_PRIMITIVE_TOPOLOGY **prim**
- std::vector< FAShapes::DrawArguments > **drawArgs**

5.7.1 Detailed Description

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

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

- FARenderScene.h

5.8 FARender::IndexBuffer Class Reference

This class stores indices in a Direct3D 12 default buffer.

`#include "FABuffer.h"`

Public Member Functions

- **IndexBuffer** (const IndexBuffer &)=delete
- IndexBuffer & **operator=** (const IndexBuffer &)=delete
- const D3D12_INDEX_BUFFER_VIEW & GetIndexBufferView ()

  Returns a constant reference to the vertex buffer view.
- void CreateIndexBuffer (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const void *data, UINT numBytes)

  Creates the vertex buffer and stores all of the specified vertices in the vertex buffer.
- void CreateIndexBufferView (UINT numBytes, DXGI_FORMAT format)

  Creates the vertex buffer view and stores it.

5.8.1 Detailed Description

This class stores indices in a Direct3D 12 default buffer.

5.8.2 Member Function Documentation

5.8.2.1 CreateIndexBuffer()

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

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

5.8.2.2 CreateIndexBufferView()

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

Creates the vertex buffer view and stores it.

5.8.2.3 GetIndexBufferView()

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

Returns a constant reference to the vertex buffer view.

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

- FABuffer.h

5.9 FARender::MultiSampling Class Reference

A wrapper for multisampling resources. Uses DirectD 12 API.

`#include "FAMultiSampling.h"`

Public Member Functions

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

  Constructor. Checks if the specifed format and sample count are supported by the specified device for multi-sampling. Throws a runtime_error if they are not supproted.
- const Microsoft::WRL::ComPtr< ID3D12Resource > & GetRenderTargetBuffer ()

  Returns the MSAA render target buffer.
- DXGI_FORMAT **GetRenderTargetFormat** ()
- DXGI_FORMAT **GetDepthStencilFormat** ()
- void ResetBuffers ()

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

5.9.1 Detailed Description

A wrapper for multisampling resources. Uses DirectD 12 API.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 MultiSampling()

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

Constructor. Checks if the specifed format and sample count are supported by the specified device for multi-sampling. Throws a runtime_error if they are not supproted.

5.9.3 Member Function Documentation

5.9.3.1 ClearDepthStencilBuffer()

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

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

5.9.3.2 ClearRenderTargetBuffer()

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

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

5.9.3.3 CreateDepthStencilBufferAndView()

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

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

5.9.3.4 CreateRenderTargetBufferAndView()

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

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

5.9.3.5 GetRenderTargetBuffer()

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

Returns the MSAA render target buffer.

5.9.3.6 ResetBuffers()

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

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

5.9.3.7 Transition()

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

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

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

- FAMultiSampling.h

5.10 FARender::RenderScene Class Reference

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

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

Public Member Functions

- **RenderScene** (unsigned int width, unsigned int height, HWND windowHandle)
- **RenderScene** (const RenderScene &)=delete
- RenderScene & **operator=** (const RenderScene &)=delete
- const DeviceResources & **GetDeviceResources** () const
- FAShapes::DrawArguments & **GetDrawArguments** (unsigned int drawSettingsIndex, unsigned int drawArgsIndex)
- const FAShapes::DrawArguments & **GetDrawArguments** (unsigned int drawSettingsIndex, unsigned int drawArgsIndex) const
- FACamera::Camera & **GetCamera** ()
- const FACamera::Camera & **GetCamera** () const
- FARender::Text & **GetText** (unsigned int textIndex)
- const FARender::Text & **GetText** (unsigned int textIndex) const
- void **LoadShader** (const std::wstring &filename)
- void **LoadShaderAndCompile** (const std::wstring &filename, const std::string &entryPointName, const std::string &target)
- void **RemoveShader** (unsigned int index)
- void **CreateInputElementDescription** (const char *semanticName, unsigned int semanticIndex, DXGI_FORMAT format, unsigned int inputSlot, unsigned int byteOffset, D3D12_INPUT_CLASSIFICATION inputSlotClassifcation=D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA, unsigned int instanceStepRate=0)
- void **RemoveInputElementDescription** (unsigned int index)
- void **CreatePSO** (unsigned int drawSettingsIndex, D3D12_FILL_MODE fillMode, BOOL enableMultisample, unsigned int vsIndex, unsigned int psIndex, const D3D12_PRIMITIVE_TOPOLOGY_TYPE &primitiveType, UINT sampleCount)
- void **CreateRootSignature** (unsigned int drawSettingsIndex)
- void **CreateVertexBuffer** ()
- void CreateIndexBuffer ()

  Creates an index buffer with the specified name and stores all of the added indices. Also creates a view to the index buffer.
  Execute commands and flush the command queue after calling createVertexBuffer() and createIndexBuffer().
- void CreateCBVHeap (UINT numDescriptors, UINT shaderRegister)

  Creates the CBV heap.
- void CreateConstantBuffer (UINT numOfBytes)

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

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

  Sets the PSO in the specified DrawSettings structure to the specified pso. If the index to the specified DrawSettings structure is out of bounds an out_of_range exception is thrown.
- void SetRootSignature (unsigned int drawSettingsIndex, const Microsoft::WRL::ComPtr< ID3D12RootSignature > &rootSignature)

  Sets the root signature in the specified DrawSettings structure to the specified root signature. If the index to the specified DrawSettings structure is out of bounds an out_of_range exception is thrown.
- void SetPrimitive (unsigned int drawSettingsIndex, const D3D_PRIMITIVE_TOPOLOGY &primitive)

  Sets the Primitive in the specified DrawSettings structure to the specified primitive. If the index to the specified DrawSettings structure is out of bounds an out_of_range exception is thrown.
- void AddDrawArgument (unsigned int drawSettingsIndex, const FAShapes::DrawArguments &drawArg)

  Adds the specified draw argument structure to the DrawArguments vector of the specified DrawSettings structure. If the index to the specified DrawSettings structure is out of bounds an out_of_range exception is thrown.
- void CreateDrawArgument (unsigned int drawSettingsIndex, unsigned int indexCount, unsigned int locationOfFirstIndex, int indexOfFirstVertex, int indexOfConstantData)

Creates a DrawArgument structure with the specified values. The created DrawArgument structure is stored in the DrawArguments vector of the specified DrawSettings structure. If the index to the specified DrawSettings structure is out of bounds an out_of_range exception is thrown.

- void RemoveDrawArgument (unsigned int drawSettingsIndex, unsigned int drawArgIndex)

  Removes the specified DrawArgument structure in the DrawArguments vector of the specified DrawSettings structure. If the index to the specified DrawSettings structure or if the index to the specifed DrawArguments structure is out of bounds an out_of_range exception is thrown.
- void CreateDrawSettings ()

  Creates a DrawSettings.
- void RemoveDrawSettings (unsigned int drawSettingsIndex)

  Removes the specified DrawSettings structure. If the index to the specified DrawSettings structure is out of bounds an out_of_range exception is thrown.
- void CreateText (FAMath::Vector4D textLocation, const std::wstring &textString, float textSize, const FAColor::Color textColor)

  Creates a Text object with the specified properties and stores it. For text location the first two values in the vector is the top-left location of the rectangle and the last two values are the bottom-right location of the rectangle.
- void RemoveText (unsigned int textIndex)

  Removes the specified Text object. If the index to the specified DrawSettings structure is out of bounds an out_of_↩ range exception is thrown.
- void AddVertices (const std::vector< FAShapes::Vertex > &vertices)

  Adds the specified vertices to the vertex list.
- void AddVertices (const FAShapes::Vertex *vertices, unsigned int numVertices)

  Adds the specified vertices to the vertex list.
- void AddIndices (const std::vector< unsigned int > &indices)

  Adds the specified vertices to the index list.
- void AddIndices (const unsigned int *indices, unsigned int numIndices)

  Adds the specified vertices to the index list.
- void BeforeDrawObjects ()

  Puts all of the commands needed in the command list before drawing the objects of the scene. Call before calling the first drawObjects function.
- void DrawObjects (unsigned int drawSettingsIndex)

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

  .
- void AfterDrawObjects ()

  Transitions the render target buffer to the correct state and excutes commands.
- void BeforeDrawText ()

  Puts all of the commands needed in the command list before drawing the text of the scene. Call before calling the first RenderText function.
- void RenderText (unsigned int textIndex)

  Draws the specified Text object. Call in between a BeforeDrawText function and a AfterDrawText function.

  .
- void AfterDrawText ()

  Transitions the render target buffer and executes all of the text drawing commands. Call after calling all the RenderText functions.
- void AfterDraw ()

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

  Executes the commands to fill the vertex and index buffer with data and flushes the queue.
- void NextFrame ()

  Moves to next frame and waits for the GPU to finish executing the next frame's commands.
- void Resize (unsigned int width, unsigned int height, HWND windowHandle)

  Resizes the DeviceResources resources when the window gets resized.

- void CopyData (UINT index, UINT byteSize, const void *data, UINT64 numOfBytes)

  Copies the specified data into the constant buffer.
- bool IsMSAAEnabled () const

  Returns true if MSAA is enabled, false otherwise.
- void DisableMSAA (unsigned int width, unsigned int height, HWND windowHandle)

  Disables MSAA.
- void EnableMSAA (unsigned int width, unsigned int height, HWND windowHandle)

  Enables MSAA.
- bool IsTextEnabled () const

  Returns true if text is enabled, false otherwise.
- void DisableText (unsigned int width, unsigned int height, HWND windowHandle)

  Disables text.
- void EnableText (unsigned int width, unsigned int height, HWND windowHandle)

  Enables text.

5.10.1 Detailed Description

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

5.10.2 Member Function Documentation

5.10.2.1 AddDrawArgument()

```
void FARender::RenderScene::AddDrawArgument (
            unsigned int drawSettingsIndex,
            const FAShapes::DrawArguments & drawArg )
```

Adds the specified draw argument structure to the DrawArguments vector of the specified DrawSettings structure. If the index to the specified DrawSettings structure is out of bounds an out_of_range exception is thrown.

5.10.2.2 AddIndices() [1/2]

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

Adds the specified vertices to the index list.

5.10.2.3 AddIndices() [2/2]

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

Adds the specified vertices to the index list.

5.10.2.4 AddVertices() [1/2]

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

Adds the specified vertices to the vertex list.

5.10.2.5 AddVertices() [2/2]

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

Adds the specified vertices to the vertex list.

5.10.2.6 AfterDraw()

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

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

5.10.2.7 AfterDrawObjects()

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

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

5.10.2.8 AfterDrawText()

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

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

5.10.2.9 BeforeDrawObjects()

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

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

5.10.2.10 BeforeDrawText()

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

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

5.10.2.11 CopyData()

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

Copies the specified data into the constant buffer.

5.10.2.12 CreateCBVHeap()

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

Creates the CBV heap.

5.10.2.13 CreateConstantBuffer()

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

Creates a constant buffer for each frame.

5.10.2.14 CreateConstantBufferView()

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

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

5.10.2.15 CreateDrawArgument()

```
void FARender::RenderScene::CreateDrawArgument (
            unsigned int drawSettingsIndex,
            unsigned int indexCount,
            unsigned int locationOfFirstIndex,
            int indexOfFirstVertex,
            int indexOfConstantData )
```

Creates a DrawArgument structure with the specified values. The created DrawArgument structure is stored in the DrawArguments vector of the specified DrawSettings structure. If the index to the specified DrawSettings structure is out of bounds an out_of_range exception is thrown.

5.10.2.16 CreateDrawSettings()

```
void FARender::RenderScene::CreateDrawSettings ( )
```

Creates a DrawSettings.

5.10.2.17 CreateIndexBuffer()

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

Creates an index buffer with the specified name and stores all of the added indices. Also creates a view to the index buffer.
Execute commands and flush the command queue after calling createVertexBuffer() and createIndexBuffer().

5.10.2.18 CreateText()

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

Creates a Text object with the specified properties and stores it. For text location the first two values in the vector is the top-left location of the rectangle and the last two values are the bottom-right location of the rectangle.

5.10.2.19 DisableMSAA()

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

Disables MSAA.

5.10.2.20 DisableText()

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

Disables text.

5.10.2.21 DrawObjects()

```
void FARender::RenderScene::DrawObjects (
            unsigned int drawSettingsIndex )
```

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

Ex.
beforeDrawObjects()
drawObjects()
drawObjects()
afterDrawObjects()
Throws an out_of_range exception if the index of the specified DrawSettings structure is out of bounds.

5.10.2.22 EnableMSAA()

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

Enables MSAA.

5.10.2.23 EnableText()

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

Enables text.

5.10.2.24 ExecuteAndFlush()

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

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

5.10.2.25 IsMSAAEnabled()

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

Returns true if MSAA is enabled, false otherwise.

5.10.2.26 IsTextEnabled()

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

Returns true if text is enabled, false otherwise.

5.10.2.27 NextFrame()

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

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

5.10.2.28 RemoveDrawArgument()

```
void FARender::RenderScene::RemoveDrawArgument (
            unsigned int drawSettingsIndex,
            unsigned int drawArgIndex )
```

Removes the specified DrawArgument structure in the DrawArguments vector of the specified DrawSettings structure. If the index to the specified DrawSettings structure or if the index to the specifed DrawArguments structure is out of bounds an out_of_range exception is thrown.

5.10.2.29 RemoveDrawSettings()

```
void FARender::RenderScene::RemoveDrawSettings (
            unsigned int drawSettingsIndex )
```

Removes the specified DrawSettings structure. If the index to the specified DrawSettings structure is out of bounds an out_of_range exception is thrown.

5.10.2.30 RemoveText()

```
void FARender::RenderScene::RemoveText (
            unsigned int textIndex )
```

Removes the specified Text object. If the index to the specified DrawSettings structure is out of bounds an out_of↩
_range exception is thrown.

5.10.2.31 RenderText()

```
void FARender::RenderScene::RenderText (
            unsigned int textIndex )
```

Draws the specified Text object. Call in between a BeforeDrawText function and a AfterDrawText function.
.

Ex.
beforeDrawText()
drawText()
drawText()
afterDrawText()
Throws an out_of_range exception if the specified Text object does not exist.

5.10.2.32 Resize()

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

Resizes the DeviceResources resources when the window gets resized.

5.10.2.33 SetPrimitive()

```
void FARender::RenderScene::SetPrimitive (
            unsigned int drawSettingsIndex,
            const D3D_PRIMITIVE_TOPOLOGY & primitive )
```

Sets the Primitive in the specified DrawSettings structure to the specified primitive. If the index to the specified DrawSettings structure is out of bounds an out_of_range exception is thrown.

5.10.2.34 SetPSO()

```
void FARender::RenderScene::SetPSO (
            unsigned int drawSettingsIndex,
            const Microsoft::WRL::ComPtr< ID3D12PipelineState > & pso )
```

Sets the PSO in the specified DrawSettings structure to the specified pso. If the index to the specified DrawSettings structure is out of bounds an out_of_range exception is thrown.

5.10.2.35 SetRootSignature()

```
void FARender::RenderScene::SetRootSignature (
            unsigned int drawSettingsIndex,
            const Microsoft::WRL::ComPtr< ID3D12RootSignature > & rootSignature )
```

Sets the root signature in the specified DrawSettings structure to the specified root signature. If the index to the specified DrawSettings structure is out of bounds an out_of_range exception is thrown.

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

- FARenderScene.h

5.11 FARender::RenderTargetBuffer Class Reference

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

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

Public Member Functions

- RenderTargetBuffer (DXGI_FORMAT format=DXGI_FORMAT_R8G8B8A8_UNORM)

  Default Constructor.
- DXGI_FORMAT GetRenderTargetFormat () const

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

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

  Returns a constant reference to the render target buffer.
- void CreateRenderTargetBufferAndView (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &rtvHeap, unsigned int indexOfWhereToStoreView, unsigned int rtvSize, unsigned int width, unsigned int height, unsigned int sampleCount=1)

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

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

  Clears the render target buffer with the specified clear value.

5.11.1 Detailed Description

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

5.11.2 Constructor & Destructor Documentation

5.11.2.1 RenderTargetBuffer()

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

Default Constructor.

5.11.3 Member Function Documentation

5.11.3.1 ClearRenderTargetBuffer()

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

Clears the render target buffer with the specified clear value.

5.11.3.2 CreateRenderTargetBufferAndView()

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

Creates the render target buffer and view.

5.11.3.3 GetRenderTargetBuffer() [1/2]

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

Returns a reference to the render target buffer.

5.11.3.4 GetRenderTargetBuffer() [2/2]

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

Returns a constant reference to the render target buffer.

5.11.3.5 GetRenderTargetFormat()

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

Returns the format of the render target buffer.

5.11.3.6 ResetBuffer()

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

Resest the render target buffer.

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

- FABuffer.h

5.12 FARender::SwapChain Class Reference

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

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

Public Member Functions

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

  Constructor. Creates a swap chain.
- const RenderTargetBuffer * GetRenderTargetBuffers () const

  Returns a constant pointer to the render target buffers.
- const Microsoft::WRL::ComPtr< ID3D12Resource > & GetCurrentBackBuffer () const

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

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

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

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

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

  The render target buffers no longer reference the swap chain buffers after this function is executed.
- void ResizeSwapChain (unsigned width, unsigned height)

  Resizes the swap chain.
- void CreateRenderTargetBuffersAndViews (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &rtvHeap, unsigned int indexOfWhereToStoreFirstView, unsigned int rtvSize)

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

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

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

  Clears the swap chains depth stencil buffer with the specified clear value.
- void Transition (const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, D3D12_RESOURCE_STATES before, D3D12_RESOURCE_STATES after)

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

  Swaps the front and back buffers.

5.12.1 Detailed Description

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

5.12.2 Constructor & Destructor Documentation

5.12.2.1 SwapChain()

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

Constructor. Creates a swap chain.

5.12.3 Member Function Documentation

5.12.3.1 ClearCurrentBackBuffer()

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

Clears the current render target buffer.

5.12.3.2 ClearDepthStencilBuffer()

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

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

5.12.3.3 CreateDepthStencilBufferAndView()

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

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

5.12.3.4 CreateRenderTargetBuffersAndViews()

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

Creates the render target buffers and views to them.

5.12.3.5 GetBackBufferFormat()

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

Returns the format of the swap chain.

5.12.3.6 GetCurrentBackBuffer()

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

Returns a constant reference to the current render target buffer.

5.12.3.7 GetCurrentBackBufferIndex()

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

Returns the current back buffer index.

5.12.3.8 GetDepthStencilFormat()

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

Returns the format of the depth stencil buffer.

5.12.3.9 GetNumRenderTargetBuffers()

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

Returns the number of swap chain buffers.

5.12.3.10 GetRenderTargetBuffers()

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

Returns a constant pointer to the render target buffers.

5.12.3.11 Present()

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

Swaps the front and back buffers.

5.12.3.12 ResetBuffers()

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

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

5.12.3.13 ResizeSwapChain()

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

Resizes the swap chain.

5.12.3.14 Transition()

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

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

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

- FASwapChain.h

5.13 FARender::Text Class Reference

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

`#include "FAText.h"`

Public Member Functions

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

  Overloaded Constructor. Initializes the format of the text.
  For text location the first two values in the vector is the top-left location of the rectangle and the last two values are the bottom-right location of the rectangle.
- const FAMath::Vector4D & GetTextLocation () const

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

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

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

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

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

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

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

  Changes the text location to the specified location.

5.13.1 Detailed Description

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

5.13.2 Constructor & Destructor Documentation

5.13.2.1 Text()

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

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

5.13.3 Member Function Documentation

5.13.3.1 GetTextColor()

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

Returns a constant reference to the text color.

5.13.3.2 GetTextLocation()

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

Returns a constant reference to the text location.

5.13.3.3 GetTextSize()

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

Returns the text size.

5.13.3.4 GetTextString()

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

Returns a constant reference to the text string.

5.13.3.5 SetTextColor()

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

Changes the text color to the specified color.

5.13.3.6 SetTextLocation()

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

Changes the text location to the specified location.

5.13.3.7 SetTextSize()

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

Changes the text size to the specified size.

5.13.3.8 SetTextString()

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

Changes the text string to the specified string.

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

- FAText.h

5.14 FARender::TextResources Class Reference

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

`#include "FATextResources.h"`

Public Member Functions

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

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

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

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

  Resets the text buffers.
- void ResizeBuffers (const RenderTargetBuffer *renderTargetBuffers, HWND windowHandle)

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

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

  Executes text commands.

5.14.1 Detailed Description

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

5.14.2 Constructor & Destructor Documentation

5.14.2.1 TextResources()

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

Constructor. Initializes the text resources.

5.14.3 Member Function Documentation

5.14.3.1 AfterRenderText()

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

Executes text commands.

5.14.3.2 BeforeRenderText()

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

Prepares to render text.

5.14.3.3 GetDirect2DDeviceContext()

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

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

5.14.3.4 GetDirectWriteFactory()

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

Returns a constant reference to the direct direct write factory.

5.14.3.5 ResetBuffers()

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

Resets the text buffers.

5.14.3.6 ResizeBuffers()

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

Resizes the buffers.

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

- FATextResources.h

5.15 FATime::Time Class Reference

Public Member Functions

- Time ()

  Default Constructor. Gets and stores the seconds per count.
- void Tick ()

  Stores the difference between the current time and the previous time.
- float DeltaTime () const

  Returns the difference between the current time and the previous time.
- void Reset ()

  Resets all time variables.
- void Stop ()

  Stops the timer.
- void Start ()

  Starts the timer.
- float TotalTime () const

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

5.15.1 Constructor & Destructor Documentation

5.15.1.1 Time()

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

Default Constructor. Gets and stores the seconds per count.

5.15.2 Member Function Documentation

5.15.2.1 DeltaTime()

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

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

5.15.2.2 Reset()

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

Resets all time variables.

5.15.2.3 Start()

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

Starts the timer.

5.15.2.4 Stop()

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

Stops the timer.

5.15.2.5 Tick()

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

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

5.15.2.6 TotalTime()

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

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

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

- FATime.h

5.16 Time Class Reference

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

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

5.16.1 Detailed Description

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

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

- FATime.h

5.17 FARender::VertexBuffer Class Reference

This class stores vertices in a Direct3D 12 default buffer.

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

Public Member Functions

- **VertexBuffer** (const VertexBuffer &)=delete
- VertexBuffer & **operator=** (const VertexBuffer &)=delete
- void CreateVertexBuffer (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL↩ ::ComPtr< ID3D12GraphicsCommandList > &commandList, const void *data, UINT numBytes)

  Creates the vertex buffer and stores all of the specified vertices in the vertex buffer.
- void CreateVertexBufferView (UINT numBytes, UINT stride)

  Creates the vertex buffer view and stores it.
- const D3D12_VERTEX_BUFFER_VIEW & GetVertexBufferView ()

  Returns a constant reference to the vertex buffer view.

5.17.1 Detailed Description

This class stores vertices in a Direct3D 12 default buffer.

5.17.2 Member Function Documentation

5.17.2.1 CreateVertexBuffer()

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

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

5.17.2.2 CreateVertexBufferView()

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

Creates the vertex buffer view and stores it.

5.17.2.3 GetVertexBufferView()

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

Returns a constant reference to the vertex buffer view.

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

- FABuffer.h

5.18 FAWindow::Window Class Reference

The window class is used to make a Window using Windows API.

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

Public Member Functions

- Window (const HINSTANCE &hInstance, const std::wstring &windowClassName, const std::wstring &windowName, WNDPROC winProcFunction, unsigned int width, unsigned int height, void *additional↩ Data=nullptr)

  Creates and displays a window. Registers a default window class with the OS with the specified instance, class name and window procdure.
- Window (const HINSTANCE &hInstance, const WNDCLASSEX &windowClass, const std::wstring &window↩ Name, unsigned int width, unsigned int height, void *additionalData=nullptr)

  Creates and displays a window. Registers the specified window class with the OS.
- HWND GetWindowHandle () const

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

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

  Returns the height of the window.
- void SetWidth (unsigned int width)

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

  Sets the height of the window o the specified height.

5.18.1 Detailed Description

The window class is used to make a Window using Windows API.

5.18.2 Constructor & Destructor Documentation

5.18.2.1 Window() [1/2]

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

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

5.18.2.2 Window() [2/2]

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

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

5.18.3 Member Function Documentation

5.18.3.1 GetHeight()

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

Returns the height of the window.

5.18.3.2 GetWidth()

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

Returns the width of the window.

5.18.3.3 GetWindowHandle()

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

Returns the window handle.

5.18.3.4 SetHeight()

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

Sets the height of the window o the specified height.

5.18.3.5 SetWidth()

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

Sets the width of the window to the specified width.

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

- FAWindow.h

Chapter 6

File Documentation

6.1 Direct3DLink.h

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

6.2 FABuffer.h File Reference

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

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

Classes

- class FARender::RenderTargetBuffer

  A wrapper for render target buffer resources. Uses DirectD 12 API.
- class FARender::DepthStencilBuffer

  A wrapper for depth stencil buffer resources. Uses DirectD 12 API.
- class FARender::VertexBuffer

  This class stores vertices in a Direct3D 12 default buffer.
- class FARender::IndexBuffer

  This class stores indices in a Direct3D 12 default buffer.
- class FARender::ConstantBuffer

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

Namespaces

- namespace FARender

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

6.2.1 Detailed Description

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

6.3 FABuffer.h

Go to the documentation of this file.

```
1 #pragma once
2
7 #include <wrl.h>
8 #include <d3d12.h>
9
13 namespace FARender
14 {
18     class RenderTargetBuffer
19     {
20     public:
23         RenderTargetBuffer(DXGI_FORMAT format = DXGI_FORMAT_R8G8B8A8_UNORM);
24
27         DXGI_FORMAT GetRenderTargetFormat() const;
28
31         Microsoft::WRL::ComPtr<ID3D12Resource>& GetRenderTargetBuffer();
32
35         const Microsoft::WRL::ComPtr<ID3D12Resource>& GetRenderTargetBuffer() const;
36
39         void CreateRenderTargetBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
40             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int
    indexOfWhereToStoreView, unsigned int rtvSize,
41             unsigned int width, unsigned int height, unsigned int sampleCount = 1);
42
45         void ResetBuffer();
46
49         void ClearRenderTargetBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
    commandList,
50             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int indexOfView,
    unsigned int rtvSize,
51             const float* clearValue);
52
53     private:
54         Microsoft::WRL::ComPtr<ID3D12Resource> mRenderTargetBuffer;
55         DXGI_FORMAT mRenderTargetFormat;
56
57     };
58
62     class DepthStencilBuffer
63     {
64     public:
65
68         DepthStencilBuffer(DXGI_FORMAT format = DXGI_FORMAT_D24_UNORM_S8_UINT);
69
72         DXGI_FORMAT GetDepthStencilFormat() const;
73
76         void CreateDepthStencilBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
77             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int
    indexOfWhereToStoreView, unsigned int dsvSize,
78             unsigned int width, unsigned int height, unsigned int sampleCount = 1);
79
82         void ResetBuffer();
83
86         void ClearDepthStencilBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
    commandList,
87             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int indexOfView,
    unsigned int dsvSize,
88             float clearValue);
89
90     private:
91         Microsoft::WRL::ComPtr<ID3D12Resource> mDepthStencilBuffer;
92         DXGI_FORMAT mDepthStencilFormat;
93     };
94
95
100     class VertexBuffer
101     {
102     public:
103         VertexBuffer() = default;
104         VertexBuffer(const VertexBuffer&) = delete;
105         VertexBuffer& operator=(const VertexBuffer&) = delete;
106
109         void CreateVertexBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
```

```
110             const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const void* data, UINT
    numBytes);
111
114         void CreateVertexBufferView(UINT numBytes, UINT stride);
115
118         const D3D12_VERTEX_BUFFER_VIEW& GetVertexBufferView();
119
120     private:
121         Microsoft::WRL::ComPtr<ID3D12Resource> mVertexDefaultBuffer;
122         Microsoft::WRL::ComPtr<ID3D12Resource> mVertexUploadBuffer;
123         D3D12_VERTEX_BUFFER_VIEW mVertexBufferView{};
124     };
125
130     class IndexBuffer
131     {
132     public:
133         IndexBuffer() = default;
134         IndexBuffer(const IndexBuffer&) = delete;
135         IndexBuffer& operator=(const IndexBuffer&) = delete;
136
139         const D3D12_INDEX_BUFFER_VIEW& GetIndexBufferView();
140
143         void CreateIndexBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
144             const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const void* data, UINT
    numBytes);
145
148         void CreateIndexBufferView(UINT numBytes, DXGI_FORMAT format);
149
150     private:
151         Microsoft::WRL::ComPtr<ID3D12Resource> mIndexDefaultBuffer;
152         Microsoft::WRL::ComPtr<ID3D12Resource> mIndexUploadBuffer;
153         D3D12_INDEX_BUFFER_VIEW mIndexBufferView{};
154     };
155
160     class ConstantBuffer
161     {
162     public:
163         ConstantBuffer() = default;
164
165         ConstantBuffer(const ConstantBuffer&) = delete;
166         ConstantBuffer& operator=(const ConstantBuffer&) = delete;
167
170         ~ConstantBuffer();
171
175         void CreateConstantBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device, const UINT&
    numOfBytes);
176
179         void CreateConstantBufferView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
180             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& cbvHeap, UINT cbvSize, UINT
    cBufferIndex,
181             UINT cbvHeapIndex, UINT numBytes);
182
186         void CopyData(UINT index, UINT byteSize, const void* data, UINT64 numOfBytes);
187
188     private:
189         Microsoft::WRL::ComPtr<ID3D12Resource> mConstantBuffer;
190         BYTE* mMappedData{ nullptr };
191     };
192 }
```

6.4 FACamera.h File Reference

File that has namespace FACamera. Withn the namespace is the class Camera.

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

Classes

- class FACamera::Camera

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

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

Namespaces

- namespace FACamera

  Has Camera class.

Typedefs

- typedef FAMath::Vector2D vec2
- typedef FAMath::Vector3D **vec3**
- typedef FAMath::Vector4D **vec4**
- typedef FAMath::Matrix4x4 **mat4**

6.4.1 Detailed Description

File that has namespace FACamera. Withn the namespace is the class Camera.

6.4.2 Typedef Documentation

6.4.2.1 vec2

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

FACAMERA_H FILE

6.5 FACamera.h

Go to the documentation of this file.

```
1 #pragma once
2
12 #include "FAMathEngine.h"
13 #include <Windows.h>
14
15 typedef FAMath::Vector2D vec2;
16 typedef FAMath::Vector3D vec3;
17 typedef FAMath::Vector4D vec4;
18 typedef FAMath::Matrix4x4 mat4;
19
23 namespace FACamera
24 {
30     class Camera
31     {
32     public:
44         Camera(vec3 cameraPosition = vec3(0.0f, 0.0f, 0.0f),
45             vec3 x = vec3(1.0f, 0.0f, 0.0f), vec3 y = vec3(0.0f, 1.0f, 0.0f), vec3 z = vec3(0.0f, 0.0f,
    1.0f),
46             float znear = 1.0f, float zfar = 100.f, float aspectRatio = 1.0f, float vFov = 45.0f,
47             float cameraVelocity = 10.0f, float angularVelocity = 0.25f);
48
51         const vec3& GetCameraPosition() const;
52
55         const vec3& GetX() const;
56
59         const vec3& GetY() const;
60
63         const vec3& GetZ() const;
64
```

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

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

6.6 FAColor.h File Reference

File has class Color under namespace FAColor.

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

Classes

- class FAColor::Color

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

Functions

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

  Returns the result of c1 + c2. Does component-wise addtion. If any of the resultant components are $>$ 1.0f, they are set to 1.0f.
- Color FAColor::operator- (const Color &c1, const Color &c2)

  Returns the result of c1 - c2. Does component-wise subtraction. If any of the resultant components are $<$ 0.0f, they are set to 0.0f.
- Color FAColor::operator$*$ (const Color &c, float k)

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

  Returns the result of k $*$ c. If k $<$ 0.0f, no multiplication happens and Color c is returned.
  If any of the resultant components are $>$ 1.0f, they are set to 1.0f.
  .
- Color FAColor::operator$*$ (const Color &c1, const Color &c2)

  Returns the result of c1 $*$ c2. If any of the resultant components are $>$ 1.0f, they are set to 1.0f.
  .

6.6.1 Detailed Description

File has class Color under namespace FAColor.

6.6.2 Function Documentation

6.6.2.1 operator∗() [1/3]

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

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

6.6.2.2 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.
.

6.6.2.3 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.
.

6.6.2.4 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.

6.6.2.5 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.

6.7 FAColor.h

Go to the documentation of this file.

```
1 #pragma once
2
3 #include "FAMathEngine.h"
4
9 namespace FAColor
10 {
16     class Color
17     {
18     public:
19
23         Color(float r = 0.0f, float g = 0.0f, float b = 0.0f, float a = 1.0f);
24
28         Color(const FAMath::Vector4D& color);
29
32         const FAMath::Vector4D& GetColor() const;
33
36         float GetRed() const;
37
40         float GetGreen() const;
41
44         float GetBlue() const;
45
48         float GetAlpha() const;
49
52         void SetColor(const FAMath::Vector4D& color);
53
56         void SetRed(float r);
57
60         void SetGreen(float g);
61
64         void SetBlue(float b);
65
68         void SetAlpha(float a);
69
73         Color& operator+=(const Color& c);
74
78         Color& operator-=(const Color& c);
79
84         Color& operator*=(float k);
85
90         Color& operator*=(const Color& c);
91
92     private:
93         FAMath::Vector4D mColor;
94     };
95
99     Color operator+(const Color& c1, const Color& c2);
100
104     Color operator-(const Color& c1, const Color& c2);
105
110     Color operator*(const Color& c, float k);
111
116     Color operator*(float k, const Color& c);
117
121     Color operator*(const Color& c1, const Color& c2);
122 }
```

6.8 FADeviceResources.h File Reference

File has class DeviceResources under namespace FARender.

```
#include <wrl.h>
#include <d3d12.h>
#include <dxgi1_4.h>
#include "FASwapChain.h"
#include "FAMultiSampling.h"
#include "FATextResources.h"
```

Classes

- class FARender::DeviceResources

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

Namespaces

- namespace FARender

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

6.8.1 Detailed Description

File has class DeviceResources under namespace FARender.

6.9 FADeviceResources.h

Go to the documentation of this file.

```
1 #pragma once
2
7 #include <wrl.h>
8 #include <d3d12.h>
9 #include <dxgi1_4.h>
10 #include "FASwapChain.h"
11 #include "FAMultiSampling.h"
12 #include "FATextResources.h"
13
14 namespace FARender
15 {
19     class DeviceResources
20     {
21     public:
22
26         static DeviceResources& GetInstance(unsigned int width, unsigned int height, HWND windowHandle,
    unsigned int numFrames);
27
28         DeviceResources(const DeviceResources&) = delete;
29         DeviceResources& operator=(const DeviceResources&) = delete;
30
33         ~DeviceResources();
34
37         const Microsoft::WRL::ComPtr<ID3D12Device>& GetDevice() const;
38
41         const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& GetCommandList() const;
42
45         DXGI_FORMAT GetBackBufferFormat() const;
46
49         DXGI_FORMAT GetDepthStencilFormat() const;
50
53         unsigned int GetCBVSize() const;
54
57         unsigned int GetNumFrames() const;
58
61         unsigned int GetCurrentFrame() const;
62
65         const TextResources& GetTextResources() const;
66
69         bool IsMSAAEnabled() const;
70
73         void DisableMSAA(unsigned int width, unsigned int height, HWND windowHandle);
74
77         void EnableMSAA(unsigned int width, unsigned int height, HWND windowHandle);
78
81         bool IsTextEnabled() const;
82
85         void DisableText(unsigned int width, unsigned int height, HWND windowHandle);
86
89         void EnableText(unsigned int width, unsigned int height, HWND windowHandle);
90
93         void UpdateCurrentFrameFenceValue();
94
```

```
99          void FlushCommandQueue();
100
104         void WaitForGPU() const;
105
108         void Signal();
109
113         void Resize(int width, int height, const HWND& handle);
114
117         void RTBufferTransition();
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         void Draw();
138
141         void NextFrame();
142
143     private:
144
156         DeviceResources(unsigned int width, unsigned int height, HWND windowHandle, unsigned int
    numFrames);
157
158         unsigned int mNumFrames;
159         unsigned int mCurrentFrameIndex;
160
161         Microsoft::WRL::ComPtr<ID3D12Device> mDirect3DDevice;
162
163         Microsoft::WRL::ComPtr<IDXGIFactory4> mDXGIFactory;
164
165         Microsoft::WRL::ComPtr<ID3D12Fence> mFence;
166         UINT64 mFenceValue;
167         std::vector<UINT64> mCurrentFrameFenceValue;
168
169         Microsoft::WRL::ComPtr<ID3D12CommandQueue> mCommandQueue;
170         std::vector<Microsoft::WRL::ComPtr<ID3D12CommandAllocator» mCommandAllocators;
171         Microsoft::WRL::ComPtr<ID3D12CommandAllocator> mDirectCommandAllocator;
172         Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList> mCommandList;
173
174         UINT mRTVSize;
175         UINT mDSVSize;
176         UINT mCBVSize;
177
178         Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mRTVHeap;
179         Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mDSVHeap;
180
181         SwapChain mSwapChain;
182
183         bool mIsMSAAEnabled;
184         MultiSampling mMultiSampling;
185
186         D3D12_VIEWPORT mViewport{};
187         D3D12_RECT mScissor{};
188
189         bool mIsTextEnabled;
190         TextResources mTextResources;
191
192         //Call all of these functions to initialize Direct3D
193         void mEnableDebugLayer();
194         void mCreateDirect3DDevice();
195         void mCreateDXGIFactory();
196         void mCreateFence();
197         void mQueryDescriptorSizes();
198         void mCreateRTVHeap();
199         void mCreateDSVHeap();
200         void mCreateCommandObjects();
201     };
202 }
```

6.10 FADirectXException.h

```
1 #pragma once
2
3 #include <wrl.h>
4 #include <dxgidebug.h>
5 #include <comdef.h>
6 #include <string>
```

```
7 #include <sstream>
8 #include <vector>
9
10 inline std::wstring AnsiToWString(const std::string& str)
11 {
12     WCHAR buffer[1024];
13     MultiByteToWideChar(CP_ACP, 0, str.c_str(), -1, buffer, 1024);
14     return std::wstring(buffer);
15 }
16
17 class DirectXException
18 {
19 public:
20     DirectXException(HRESULT hr, const std::wstring& functionName, const std::wstring& fileName, int
      lineNumber);
21
22     std::wstring ErrorMsg() const;
23
24 private:
25     HRESULT errorCode;
26     std::wstring functionName;
27     std::wstring fileName;
28     int lineNumber;
29     Microsoft::WRL::ComPtr<IDXGIInfoQueue> mInfoQueue;
30 };
31
32 //use when calling Direct3D or DXGI function to check if the function failed or not.
33 #ifndef ThrowIfFailed
34 #define ThrowIfFailed(x)                                                                    \
35 {                                                                                           \
36 HRESULT hr = (x);                                                                   \
37 std::wstring filename(AnsiToWString(__FILE__));                                     \
38 if (FAILED(hr)) { throw DirectXException(hr, L#x, filename, __LINE__); }    \
39 }
40 #endif
```

6.11 FAMultiSampling.h

```
1 #pragma once
2
3 #include <wrl.h>
4 #include "d3dx12.h"
5 #include "FABuffer.h"
6
7 namespace FARender
8 {
12     class MultiSampling
13     {
14     public:
15
16         MultiSampling() = default;
17
22         MultiSampling(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
23             DXGI_FORMAT rtFormat, DXGI_FORMAT dsFormat, unsigned int sampleCount);
24
27         const Microsoft::WRL::ComPtr<ID3D12Resource>& GetRenderTargetBuffer();
28
29         DXGI_FORMAT GetRenderTargetFormat();
30
31         DXGI_FORMAT GetDepthStencilFormat();
32
35         void ResetBuffers();
36
39         void CreateRenderTargetBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
40             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int
      indexOfWhereToStoreView, unsigned int rtvSize,
41             unsigned int width, unsigned int height);
42
45         void CreateDepthStencilBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
46             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int
      indexOfWhereToStoreView, unsigned int dsvSize,
47             unsigned int width, unsigned int height);
48
51         void Transition(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList,
52             D3D12_RESOURCE_STATES before, D3D12_RESOURCE_STATES after);
53
56         void ClearRenderTargetBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
      commandList,
57             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int indexOfView,
      unsigned int rtvSize,
58             const float* clearValue);
59
62         void ClearDepthStencilBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
      commandList,
```

```
63             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int indexOfView,
       unsigned int dsvSize,
64             float clearValue);
65
66     private:
67         RenderTargetBuffer mMSAARenderTargetBuffer;
68         DepthStencilBuffer mMSAADepthStencilBuffer;
69         unsigned int mSampleCount{ 0 };
70
71         /*Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mMSAARTVDescriptorHeap;
72 Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mMSAADSVDescriptorHeap;
73 Microsoft::WRL::ComPtr<ID3D12Resource> mMSAARenderTargetBuffer;
74 Microsoft::WRL::ComPtr<ID3D12Resource> mMSAADepthStencilBuffer;*/
75
76     };
77 }
```

6.12 FARenderScene.h File Reference

File has class RenderScene under namespace FARender.

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

Classes

- struct FARender::DrawSettings

  Holds a array of objects that use the same PSO, root signature and primitive.
- class FARender::RenderScene

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

Namespaces

- namespace FARender

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

6.12.1 Detailed Description

File has class RenderScene under namespace FARender.

6.13 FARenderScene.h

Go to the documentation of this file.

```
1 #pragma once
2
7 #include <d3dcompiler.h>
8 #include <unordered_map>
9 #include <string>
10 #include "FADeviceResources.h"
11 #include "FABuffer.h"
12 #include "FACamera.h"
13 #include "FAText.h"
14 #include "FAShapesUtility.h"
15
16 namespace FARender
17 {
21     struct DrawSettings
22     {
23         Microsoft::WRL::ComPtr<ID3D12PipelineState> pipelineState;
24         Microsoft::WRL::ComPtr<ID3D12RootSignature> rootSig;
25         D3D_PRIMITIVE_TOPOLOGY prim;
26         std::vector<FAShapes::DrawArguments> drawArgs;
27     };
28
29
33     class RenderScene
34     {
35     public:
36
37         RenderScene(unsigned int width, unsigned int height, HWND windowHandle);
38
39         RenderScene(const RenderScene&) = delete;
40         RenderScene& operator=(const RenderScene&) = delete;
41
42         /*@brief Returns a constant reference to the device resources object.
43 */
44         const DeviceResources& GetDeviceResources() const;
45
46         /*@brief Returns a reference to the specified DrawArguments structure in the specified
    DrawSettings structure.
47 * Throws an out_of_range exception if the index to DrawSettings structure or index to the DrawArguments
    structure
48 * is out of bounds.
49 */
50         FAShapes::DrawArguments& GetDrawArguments(unsigned int drawSettingsIndex, unsigned int
    drawArgsIndex);
51
52         /*@brief Returns a constant reference to the specified DrawArguments object in the specified
    DrawSettings structure.
53 * Throws an out_of_range exception if the index to DrawSettings structure or index to the DrawArguments
    structure
54 * is out of bounds.
55 */
56         const FAShapes::DrawArguments& GetDrawArguments(unsigned int drawSettingsIndex, unsigned int
    drawArgsIndex) const;
57
58         /*@brief Returns a reference to the this scene's camera;
59 */
60         FACamera::Camera& GetCamera();
61
62         /*@brief Returns a constant reference to the this scene's camera;
63 */
64         const FACamera::Camera& GetCamera() const;
65
66         /*@brief Returns a reference to the specified Text object.
67 * If the index of the Text object is out of bounds an out_of_range exception is thrown.
68 */
69         FARender::Text& GetText(unsigned int textIndex);
70
71         /*@brief Returns a constant reference to the specified Text object.
72 * If the index of the Text object is out of bounds an out_of_range exception is thrown.
73 */
74         const FARender::Text& GetText(unsigned int textIndex) const;
75
76         /*@brief Loads a shader's bytecode and stores it.
77 */
78         void LoadShader(const std::wstring& filename);
79
80         /*@brief Loads a shaders file, compiles it into bytecode and stores the bytecode.
81 */
82         void LoadShaderAndCompile(const std::wstring& filename, const std::string& entryPointName, const
    std::string& target);
83
84         /*@brief Removes the Shader structue at the specified index.
85 * If the index to the specifed Shader structure is out of bounds, an out_of_range exception is thrown.
```

```
86 */
87         void RemoveShader(unsigned int index);
88 
89         /*@brief Creates an input element description and stores it.
90 */
91         void CreateInputElementDescription(const char* semanticName, unsigned int semanticIndex,
92             DXGI_FORMAT format, unsigned int inputSlot, unsigned int byteOffset,
93             D3D12_INPUT_CLASSIFICATION inputSlotClassifcation =
    D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA,
94             unsigned int instanceStepRate = 0);
95 
96         /*@brief Removes the specified input element description.
97 * If the index to the input element description is out of bounds an out_of_range exceptio is thrown.
98 */
99         void RemoveInputElementDescription(unsigned int index);
100 
101         /*@brief Creates a PSO and stores it in the specified DrawSettings structure.
102 * If the indices to the specified DrawSettings structure, Vertex Shader or Pixel Shader
103 * is out of bounds an out_of_range exception is thrown.
104 */
105         void CreatePSO(unsigned int drawSettingsIndex, D3D12_FILL_MODE fillMode, BOOL enableMultisample,
106             unsigned int vsIndex, unsigned int psIndex,
107             const D3D12_PRIMITIVE_TOPOLOGY_TYPE& primitiveType, UINT sampleCount);
108 
109         /*@brief Creates a root signature and stores it in the specified DrawSettings structure.
110 * If the index to the specified DrawSettings structure is out of bounds an out_of_range exception is
    thrown.
111 */
112         void CreateRootSignature(unsigned int drawSettingsIndex);
113 
114         /*@brief Creates a vertex buffer with the specified name and stores all of the added vertices.
115 * Also creates a view to the vertex buffer.\n
116 * Execute commands and the flush command queue after calling createVertexBuffer() and
    createIndexBuffer().
117 */
118         void CreateVertexBuffer();
119 
124         void CreateIndexBuffer();
125 
128         void CreateCBVHeap(UINT numDescriptors, UINT shaderRegister);
129 
132         void CreateConstantBuffer(UINT numOfBytes);
133 
136         void CreateConstantBufferView(UINT index, UINT numBytes);
137 
141         void SetPSO(unsigned int drawSettingsIndex, const Microsoft::WRL::ComPtr<ID3D12PipelineState>&
    pso);
142 
146         void SetRootSignature(unsigned int drawSettingsIndex,
147             const Microsoft::WRL::ComPtr<ID3D12RootSignature>& rootSignature);
148 
152         void SetPrimitive(unsigned int drawSettingsIndex, const D3D_PRIMITIVE_TOPOLOGY& primitive);
153 
157         void AddDrawArgument(unsigned int drawSettingsIndex, const FAShapes::DrawArguments& drawArg);
158 
163         void CreateDrawArgument(unsigned int drawSettingsIndex,
164             unsigned int indexCount, unsigned int locationOfFirstIndex, int indexOfFirstVertex, int
    indexOfConstantData);
165 
170         void RemoveDrawArgument(unsigned int drawSettingsIndex, unsigned int drawArgIndex);
171 
174         void CreateDrawSettings();
175 
179         void RemoveDrawSettings(unsigned int drawSettingsIndex);
180 
185         void CreateText(FAMath::Vector4D textLocation, const std::wstring& textString,
186             float textSize, const FAColor::Color textColor);
187 
191         void RemoveText(unsigned int textIndex);
192 
195         void AddVertices(const std::vector<FAShapes::Vertex>& vertices);
196 
199         void AddVertices(const FAShapes::Vertex* vertices, unsigned int numVertices);
200 
203         void AddIndices(const std::vector<unsigned int>& indices);
204 
207         void AddIndices(const unsigned int* indices, unsigned int numIndices);
208 
212         void BeforeDrawObjects();
213 
225         void DrawObjects(unsigned int drawSettingsIndex);
226 
229         void AfterDrawObjects();
230 
234         void BeforeDrawText();
235 
247         void RenderText(unsigned int textIndex);
```

```
248
252         void AfterDrawText();
253
257         void AfterDraw();
258
261         void ExecuteAndFlush();
262
265         void NextFrame();
266
269         void Resize(unsigned int width, unsigned int height, HWND windowHandle);
270
273         void CopyData(UINT index, UINT byteSize, const void* data, UINT64 numOfBytes);
274
277         bool IsMSAAEnabled() const;
278
281         void DisableMSAA(unsigned int width, unsigned int height, HWND windowHandle);
282
285         void EnableMSAA(unsigned int width, unsigned int height, HWND windowHandle);
286
289         bool IsTextEnabled() const;
290
293         void DisableText(unsigned int width, unsigned int height, HWND windowHandle);
294
297         void EnableText(unsigned int width, unsigned int height, HWND windowHandle);
298
299     private:
300
301         static const unsigned int NUM_OF_FRAMES{ 3 };
302
303         //The device resources object that all RenderScene objects share.
304         DeviceResources& mDeviceResources;
305
306         //Stores all of the shaders for this scene.
307         std::vector<Microsoft::WRL::ComPtr<ID3DBlob» mShaders;
308
309         //Stores input element descriptions for the shaders.
310         std::vector<D3D12_INPUT_ELEMENT_DESC> mInputElementDescriptions;
311
312         //Stores draw settings that the scene uses.
313         std::vector<DrawSettings> mSceneObjects;
314
315         //Each scene gets a CBV heap.
316         Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mCBVHeap;
317         D3D12_DESCRIPTOR_RANGE mCBVHeapDescription{};
318         D3D12_ROOT_PARAMETER mCBVHeapRootParameter;
319
320         //Stores all of the constant buffers this scene uses.  We can't update a constant buffer until
    the GPU
321         //is done executing all the commands that reference it, so each frame needs its own constant
    buffer.
322         ConstantBuffer mConstantBuffer[NUM_OF_FRAMES];
323
324         //The vertices and indicies for the scene.
325         std::vector<FAShapes::Vertex> mVertexList;
326         std::vector<unsigned int> mIndexList;
327
328         //The vertex and index buffer for the scene.
329         VertexBuffer mVertexBuffer;
330         IndexBuffer mIndexBuffer;
331
332         //All of the text that is rendered with the scene.
333         std::vector<Text> mTexts;
334
335         //The camera for the scene.
336         FACamera::Camera mCamera;
337     };
338 }
```

6.14 FASwapChain.h

```
1 #pragma once
2
3 #include <wrl.h>
4 #include "d3dx12.h"
5 #include <dxgi1_4.h>
6 #include <vector>
7 #include "FABuffer.h"
8
9 namespace FARender
10 {
14     class SwapChain
15     {
16     public:
```

```
17
18          SwapChain() = default;
19
23          SwapChain(const Microsoft::WRL::ComPtr<IDXGIFactory4>& dxgiFactory,
24              const Microsoft::WRL::ComPtr<ID3D12CommandQueue>& commandQueue, HWND windowHandle,
25              DXGI_FORMAT rtFormat = DXGI_FORMAT_R8G8B8A8_UNORM, DXGI_FORMAT dsFormat =
    DXGI_FORMAT_D24_UNORM_S8_UINT,
26              unsigned int numRenderTargetBuffers = 2);
27
30          const RenderTargetBuffer* GetRenderTargetBuffers() const;
31
34          const Microsoft::WRL::ComPtr<ID3D12Resource>& GetCurrentBackBuffer() const;
35
38          unsigned int GetNumRenderTargetBuffers() const;
39
42          unsigned int GetCurrentBackBufferIndex() const;
43
46          DXGI_FORMAT GetBackBufferFormat() const;
47
50          DXGI_FORMAT GetDepthStencilFormat() const;
51
54          void ResetBuffers();
55
58          void ResizeSwapChain(unsigned width, unsigned height);
59
62          void CreateRenderTargetBuffersAndViews(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
63              const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int
    indexOfWhereToStoreFirstView,
64              unsigned int rtvSize);
65
68          void CreateDepthStencilBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
69              const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int index, unsigned int
    dsvSize,
70              unsigned int width, unsigned int height);
71
74          void ClearCurrentBackBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList,
75              const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int indexOfFirstView,
    unsigned int rtvSize,
76              const float* backBufferClearValue);
77
80          void ClearDepthStencilBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
    commandList,
81              const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int indexOfView,
    unsigned int dsvSize,
82              float clearValue);
83
86          void Transition(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList,
87              D3D12_RESOURCE_STATES before, D3D12_RESOURCE_STATES after);
88
91          void Present();
92
93      private:
94          unsigned int mNumRenderTargetBuffers = 0;
95          unsigned int mCurrentBackBufferIndex = 0;
96
97          Microsoft::WRL::ComPtr<IDXGISwapChain1> mSwapChain;
98          std::vector<RenderTargetBuffer> mRenderTargetBuffers;
99
100         DepthStencilBuffer mDepthStencilBuffer;
101     };
102 }
```

6.15 FAText.h File Reference

File has class Text under namespace FARender.

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

Classes

- class FARender::Text

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

Namespaces

- namespace FARender

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

6.15.1 Detailed Description

File has class Text under namespace FARender.

6.16 FAText.h

Go to the documentation of this file.

```
1 #pragma once
2
7 #include <string>
8 #include "FAColor.h"
9
10 namespace FARender
11 {
16     class Text
17     {
18     public:
19
20         Text() = default;
21
27         Text(const FAMath::Vector4D& textLocation, const std::wstring& textString, float textSize, const
      FAColor::Color& textColor);
28
31         const FAMath::Vector4D& GetTextLocation() const;
32
35         const std::wstring& GetTextString() const;
36
39         float GetTextSize() const;
40
43         const FAColor::Color& GetTextColor() const;
44
47         void SetTextSize(float textSize);
48
51         void SetTextColor(const FAColor::Color& textColor);
52
55         void SetTextString(const std::wstring& textString);
56
59         void SetTextLocation(const FAMath::Vector4D& textLocation);
60
61     private:
62
63         FAMath::Vector4D mTextLocation;
64         std::wstring mText;
65         float mTextSize{ 0.0f };
66         FAColor::Color mTextColor;
67     };
68 }
```

6.17 FATextResources.h

```
1 #pragma once
2
3 #include <wrl.h>
4 #include <d3d11.h>
5 #include <d3d11on12.h>
6 #include <d2d1_3.h>
7 #include <dwrite.h>
8 #include <vector>
9 #include "FABuffer.h"
10
11 namespace FARender
12 {
16     class TextResources
17     {
18     public:
```

```
19          TextResources() = default;
20
24          TextResources(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
25              const Microsoft::WRL::ComPtr<ID3D12CommandQueue>& commandQueue, unsigned int
    numSwapChainBuffers);
26
29          const Microsoft::WRL::ComPtr<ID2D1DeviceContext>& GetDirect2DDeviceContext() const;
30
33          const Microsoft::WRL::ComPtr<IDWriteFactory>& GetDirectWriteFactory() const;
34
37          void ResetBuffers();
38
41          void ResizeBuffers(const RenderTargetBuffer* renderTargetBuffers, HWND windowHandle);
42
45          void BeforeRenderText(unsigned int currentBackBuffer);
46
49          void AfterRenderText(unsigned int currentBackBuffer);
50
51      private:
52          Microsoft::WRL::ComPtr<ID3D11Device> mDevice11;
53          Microsoft::WRL::ComPtr<ID3D11DeviceContext> mDevice11Context;
54          Microsoft::WRL::ComPtr<ID3D11On12Device> mDevice11on12;
55
56          Microsoft::WRL::ComPtr<ID2D1Device2> mDirect2DDevice;
57          Microsoft::WRL::ComPtr<ID2D1Factory3> mDirect2DFactory;
58          Microsoft::WRL::ComPtr<ID2D1DeviceContext> mDirect2DDeviceContext;
59
60          Microsoft::WRL::ComPtr<IDWriteFactory> mDirectWriteFactory;
61
62          std::vector<Microsoft::WRL::ComPtr<ID3D11Resource» mWrappedBuffers;
63          std::vector<Microsoft::WRL::ComPtr<ID2D1Bitmap1» mDirect2DBuffers;
64          std::vector<Microsoft::WRL::ComPtr<IDXGISurface» mSurfaces;
65      };
66  }
```

6.18 FATime.h File Reference

File that has namespace FATime. Withn the namespace is the class Time.

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

Classes

- class FATime::Time

6.18.1 Detailed Description

File that has namespace FATime. Withn the namespace is the class Time.

6.19 FATime.h

Go to the documentation of this file.

```
1  #pragma once
2
7  #include <Windows.h>
8
12 namespace FATime
13 {
14      class Time
15      {
16      public:
20          Time();
21
24          void Tick();
```

```
25
28          float DeltaTime() const;
29
32          void Reset();
33
36          void Stop();
37
40          void Start();
41
44          float TotalTime() const;
45
46      private:
47          __int64 mCurrTime; //holds current time stamp ti
48          __int64 mPrevTime; //holds previous time stamp ti-1
49          __int64 mStopTime; //holds the time we stopped the game/animation
50          __int64 mPausedTime; //holds how long the game/animation was paused for
51          __int64 mBaseTime; //holds the time we started / resetted
52
53          double mSecondsPerCount;
54          double mDeltaTime; //time elapsed btw frames change in t = ti - ti-1
55
56          bool mStopped; //flag to indicate if the game/animation is paused or not
57
58      };
59  }
```

6.20 FAWindow.h File Reference

File that has namespace FAWindow. Withn the namespace is the class Window.

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

Classes

- class FAWindow::Window

  The window class is used to make a Window using Windows API.

Namespaces

- namespace FAWindow

  Has Window class.

6.20.1 Detailed Description

File that has namespace FAWindow. Withn the namespace is the class Window.

6.21 FAWindow.h

Go to the documentation of this file.

```
1 #pragma once
2
7 #include <Windows.h>
8 #include <string>
9 #include <stdexcept>
10
14 namespace FAWindow
15 {
19     class Window
20     {
21     public:
22         //Window();
23
27         Window(const HINSTANCE& hInstance, const std::wstring& windowClassName, const std::wstring&
       windowName,
28             WNDPROC winProcFunction, unsigned int width, unsigned int height, void* additionalData =
       nullptr);
29
33         Window(const HINSTANCE& hInstance, const WNDCLASSEX& windowClass, const std::wstring& windowName,
34             unsigned int width, unsigned int height, void* additionalData = nullptr);
35
38         HWND GetWindowHandle() const;
39
42         unsigned int GetWidth() const ;
43
46         unsigned int GetHeight() const;
47
50         void SetWidth(unsigned int width);
51
54         void SetHeight(unsigned int height);
55
56     private:
57         HWND mWindowHandle;
58
59         WNDCLASSEX mWindowClass;
60         std::wstring mWindowClassName;
61
62         unsigned int mWidth;
63         unsigned int mHeight;
64     };
65 }
```

Index