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 &num←OfBytes)

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

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

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

5.3.1 Detailed Description

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

5.3.2 Constructor & Destructor Documentation

5.3.2.1 ∼ConstantBuffer()

```
FARender::ConstantBuffer::∼ConstantBuffer ( )
```

Unmaps the pointer to the constant buffer.

5.3.3 Member Function Documentation

5.3.3.1 CopyData()

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

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

5.3.3.2 CreateConstantBuffer()

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

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

5.3.3.3 CreateConstantBufferView()

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

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

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

- FABuffer.h

5.4 DepthStencil Class Reference

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

`#include "FADepthStencil.h"`

5.4.1 Detailed Description

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

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

- FADepthStencil.h

5.5 FARender::DepthStencilBuffer Class Reference

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

`#include "FABuffer.h"`

Public Member Functions

- DepthStencilBuffer (DXGI_FORMAT format=DXGI_FORMAT_D24_UNORM_S8_UINT)

  Default Constructor.
- DXGI_FORMAT GetDepthStencilFormat () const

  Returns the format of the depth stencil buffer.
- void CreateDepthStencilBufferAndView (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &dsvHeap, unsigned int 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 > &command↩ List, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &dsvHeap, unsigned int indexOfView, unsigned int dsvSize, float clearValue)

  Clears the depth stencil buffer with the specified clear value.
- DepthStencilBuffer (DXGI_FORMAT format=DXGI_FORMAT_D24_UNORM_S8_UINT)

  Default Constructor.
- DXGI_FORMAT GetDepthStencilFormat () const

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

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

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

  Clears the depth stencil buffer with the specified clear value.

5.5.1 Detailed Description

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

5.5.2 Constructor & Destructor Documentation

5.5.2.1 DepthStencilBuffer() [1/2]

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

Default Constructor.

5.5.2.2 DepthStencilBuffer() [2/2]

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

Default Constructor.

5.5.3 Member Function Documentation

5.5.3.1 ClearDepthStencilBuffer() [1/2]

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

Clears the depth stencil buffer with the specified clear value.

5.5.3.2 ClearDepthStencilBuffer() [2/2]

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

Clears the depth stencil buffer with the specified clear value.

5.5.3.3 CreateDepthStencilBufferAndView() [1/2]

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

Creates the depth stencil buffer and view.

5.5.3.4 CreateDepthStencilBufferAndView() [2/2]

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

Creates the depth stencil buffer and view.

5.5.3.5 GetDepthStencilFormat() [1/2]

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

Returns the format of the depth stencil buffer.

5.5.3.6 GetDepthStencilFormat() [2/2]

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

Returns the format of the depth stencil buffer.

5.5.3.7 ResetBuffer() [1/2]

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

Resest the depth stencil buffer.

5.5.3.8 ResetBuffer() [2/2]

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

Resest the depth stencil buffer.

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

- FABuffer.h
- FADepthStencil.h

5.6 FARender::DeviceResources Class Reference

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

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

Public Member Functions

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

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

  Returns a constant reference to the ID3D12Device 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.
- UINT GetCBVSize () const

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

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

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

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

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

  Enables MSAA.
- void UpdateCurrentFrameFenceValue ()

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

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

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

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

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

  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)

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

Static Public Attributes

- static const unsigned int **NUM_OF_FRAMES** { 3 }

5.6.1 Detailed Description

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

5.6.2 Constructor & Destructor Documentation

5.6.2.1 ∼DeviceResources()

```
FARender::DeviceResources::∼DeviceResources ( )
```

Flushes the command queue.

5.6.3 Member Function Documentation

5.6.3.1 AfterTextDraw()

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

Executes the text commands.

5.6.3.2 BeforeTextDraw()

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

Prepares to render text.

5.6.3.3 DisableMSAA()

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

Disables MSAA.

5.6.3.4 EnableMSAA()

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

Enables MSAA.

5.6.3.5 Execute()

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

Executes the command list.

5.6.3.6 FlushCommandQueue()

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

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

5.6.3.7 GetBackBufferFormat()

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

Returns a constant reference to the back buffer format.

5.6.3.8 GetCBVSize()

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

The size of a constant buffer view.

5.6.3.9 GetCommandList()

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

Returns a constant reference to the ID3D12GraphicsCommandList objcet.

5.6.3.10 GetCurrentFrame()

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

Returns the current frame.

5.6.3.11 GetDepthStencilFormat()

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

Returns a constant reference to the depth stencil format.

5.6.3.12 GetDevice()

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

Returns a constant reference to the ID3D12Device objcet.

5.6.3.13 GetInstance()

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

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

5.6.3.14 GetTextResources()

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

Returns a constant reference to the TextResources object.

5.6.3.15 IsMSAAEnabled()

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

Returns true if MSAA is enabled, false otherwise.

5.6.3.16 NextFrame()

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

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

5.6.3.17 Present()

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

Swaps the front and back buffers.

5.6.3.18 Resize()

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

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

5.6.3.19 RTBufferTransition()

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

Transistions the render target buffer.

5.6.3.20 Signal()

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

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

5.6.3.21 UpdateCurrentFrameFenceValue()

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

Updates the current frames fence value.

5.6.3.22 WaitForGPU()

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

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

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

- FADeviceResources.h

5.7 DirectXException Class Reference

Public Member Functions

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

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

- FADirectXException.h

5.8 FARender::DrawSettings Struct Reference

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

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

Public Attributes

- Microsoft::WRL::ComPtr< ID3D12PipelineState > **pipelineState**
- Microsoft::WRL::ComPtr< ID3D12RootSignature > **rootSig**
- D3D_PRIMITIVE_TOPOLOGY **prim** = D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST
- std::vector< FAShapes::DrawArguments > **drawArgs**

5.8.1 Detailed Description

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

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

- FARenderScene.h

5.9 FARender::IndexBuffer Class Reference

This class stores indices in a Direct3D 12 default buffer.

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

Public Member Functions

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

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

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

  Creates the vertex buffer view and stores it.

5.9.1 Detailed Description

This class stores indices in a Direct3D 12 default buffer.

5.9.2 Member Function Documentation

5.9.2.1 CreateIndexBuffer()

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

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

5.9.2.2 CreateIndexBufferView()

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

Creates the vertex buffer view and stores it.

5.9.2.3 GetIndexBufferView()

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

Returns a constant reference to the vertex buffer view.

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

- FABuffer.h

5.10 FARender::MultiSampling Class Reference

A wrapper for multisampling resources. Uses DirectD 12 API.

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

Public Member Functions

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

  Constructor. Checks if the 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 index↩OfView, unsigned int rtvSize, const float ∗clearValue)

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

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

5.10.1 Detailed Description

A wrapper for multisampling resources. Uses DirectD 12 API.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 MultiSampling()

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

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

5.10.3 Member Function Documentation

5.10.3.1 ClearDepthStencilBuffer()

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

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

5.10.3.2 ClearRenderTargetBuffer()

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

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

5.10.3.3 CreateDepthStencilBufferAndView()

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

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

5.10.3.4 CreateRenderTargetBufferAndView()

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

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

5.10.3.5 GetRenderTargetBuffer()

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

Returns the MSAA render target buffer.

5.10.3.6 ResetBuffers()

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

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

5.10.3.7 Transition()

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

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

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

- FAMultiSampling.h

5.11 FARender::RenderScene Class Reference

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

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

Public Member Functions

- **RenderScene** (unsigned int width, unsigned int height, HWND windowHandle)
- **RenderScene** (const RenderScene &)=delete
- RenderScene & **operator=** (const RenderScene &)=delete
- const DeviceResources & **GetDeviceResources** () const
- const Microsoft::WRL::ComPtr< ID3DBlob > & **GetShader** (const std::wstring &name) const
- const std::vector< D3D12_INPUT_ELEMENT_DESC > & **GetInputElementLayout** (const std::wstring &name) const
- const D3D12_RASTERIZER_DESC & **GetRasterizationState** (const std::wstring &name) const
- const Microsoft::WRL::ComPtr< ID3D12PipelineState > & **GetPSO** (const std::wstring &drawSettingsName) const
- const Microsoft::WRL::ComPtr< ID3D12RootSignature > & **GetRootSignature** (const std::wstring &draw↩ SettingsName) const
- const D3D_PRIMITIVE_TOPOLOGY & **GetPrimitive** (const std::wstring &drawSettingsName) const
- FAShapes::DrawArguments & **GetDrawArguments** (const std::wstring &drawSettingsName, unsigned int index)
- const FAShapes::DrawArguments & **GetDrawArguments** (const std::wstring &drawSettingsName, unsigned int index) const
- FACamera::Camera & **GetCamera** ()
- const FACamera::Camera & **GetCamera** () const
- FARender::Text & **GetText** (std::wstring textName)
- const FARender::Text & **GetText** (std::wstring textName) const
- void **LoadShader** (const std::wstring &filename, const std::wstring &name)
- void **RemoveShader** (const std::wstring &shaderName)
- void **StoreInputElementDescriptions** (const std::wstring &name, const std::vector< D3D12_INPUT_↩ ELEMENT_DESC > &inputElementLayout)
- void **StoreInputElementDescriptions** (const std::wstring &name, const D3D12_INPUT_ELEMENT_DESC ∗inputElementLayout, UINT numElements)
- void **RemoveInputElementDescription** (const std::wstring &name)
- void **CreateRasterizationState** (D3D12_FILL_MODE fillMode, BOOL enableMultisample, const std::wstring &name)
- void **RemoveRasterizationState** (const std::wstring &name)
- void **CreatePSO** (const std::wstring &drawSettingsName, const std::wstring &rStateName, const std::wstring &vsName, const std::wstring &psName, const std::wstring &inputLayoutName, const D3D12_PRIMITIVE_↩ TOPOLOGY_TYPE &primitiveType, UINT sampleCount)
- void **CreateRootSignature** (const std::wstring &drawSettingsName)
- void **CreateVertexBuffer** ()
- void CreateIndexBuffer ()

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

  Creates the CBV heap.
- void CreateConstantBuffer (UINT 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 (const std::wstring &drawSettingsName, const Microsoft::WRL::ComPtr< ID3D12PipelineState > &pso)

  Sets the PSO in the specified DrawSettings structure to the specified pso. If the specifed DrawSettings structure does not exist an out_of_range exception is thrown.
- void SetRootSignature (const std::wstring &drawSettingsName, const Microsoft::WRL::ComPtr< ID3D12↩ RootSignature > &rootSignature)

Sets the root signature in the specified DrawSettings structure to the specified root signature. If the specifed DrawSettings structure does not exist an out_of_range exception is thrown.

- void SetPrimitive (const std::wstring &drawSettingsName, const D3D_PRIMITIVE_TOPOLOGY &primitive)

    Sets the Primitive in the specified DrawSettings structure to the specified primitive. If the specifed DrawSettings structure does not exist an out_of_range exception is thrown.

- void AddDrawArgument (const std::wstring &drawSettingsName, const FAShapes::DrawArguments &draw←Arg)

    Adds the specified draw argument structure to the DrawArguments vector of the specified DrawSettings structure. If the specifed DrawSettings structure does not exist an out_of_range exception is thrown.

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

    Adds the specified draw arguments to the DrawArguments vector of the specified DrawSettings structure. If the specifed DrawSettings structure does not exist an out_of_range exception is thrown.

- void RemoveDrawArgument (const std::wstring &drawSettingsName, unsigned int index)

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

- void CreateDrawSettings (const std::wstring &drawSettingsName)

    Creates a DrawSettings structure with the specified name.

- void RemoveDrawSettings (const std::wstring &drawSettingsName)

    Removes the specified DrawSettings structure. If the DrawSettings structure does not exist an out_of_range exception is thrown.

- void CreateText (const std::wstring &textName, FAMath::Vector4D textLocation, const std::wstring &text←String, float textSize, const FAColor::Color textColor)

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

- void RemoveText (const std::wstring &textName)

    Removes the specified text object with the specified name. If the Text object does not exist an out_of_range exception is thrown.

- void AddVertices (const std::vector< FAShapes::Vertex > &vertices)

    Adds the specified vertices to the vertex list.

- void AddVertices (const FAShapes::Vertex ∗vertices, unsigned int numVertices)

    Adds the specified vertices to the vertex list.

- void AddIndices (const std::vector< unsigned int > &indices)

    Adds the specified vertices to the index list.

- void AddIndices (const unsigned int ∗indices, unsigned int numIndices)

    Adds the specified vertices to the index list.

- void BeforeDrawObjects ()

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

- void DrawObjects (const std::wstring &drawSettingsName)

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

    .

- void AfterDrawObjects (bool renderText)

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

- void BeforeDrawText ()

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

- void RenderText (const std::wstring &textName)

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

    .

- void AfterDrawText ()

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

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

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

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

  Resizes the DeviceResources resources when the window gets resized.
- void CopyData (UINT index, UINT byteSize, const void ∗data, UINT64 numOfBytes)

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

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

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

  Enables MSAA.

5.11.1 Detailed Description

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

5.11.2 Member Function Documentation

5.11.2.1 AddDrawArgument() [1/2]

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

Adds the specified draw argument structure to the DrawArguments vector of the specified DrawSettings structure. If the specifed DrawSettings structure does not exist an out_of_range exception is thrown.

5.11.2.2 AddDrawArgument() [2/2]

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

Adds the specified draw arguments to the DrawArguments vector of the specified DrawSettings structure. If the specifed DrawSettings structure does not exist an out_of_range exception is thrown.

5.11.2.3 AddIndices() [1/2]

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

Adds the specified vertices to the index list.

5.11.2.4 AddIndices() [2/2]

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

Adds the specified vertices to the index list.

5.11.2.5 AddVertices() [1/2]

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

Adds the specified vertices to the vertex list.

5.11.2.6 AddVertices() [2/2]

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

Adds the specified vertices to the vertex list.

5.11.2.7 AfterDraw()

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

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

5.11.2.8 AfterDrawObjects()

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

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

5.11.2.9 AfterDrawText()

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

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

5.11.2.10 BeforeDrawObjects()

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

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

5.11.2.11 BeforeDrawText()

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

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

5.11.2.12 CopyData()

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

Copies the specified data into the constant buffer.

5.11.2.13 CreateCBVHeap()

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

Creates the CBV heap.

5.11.2.14 CreateConstantBuffer()

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

Creates a constant buffer for each frame.

5.11.2.15 CreateConstantBufferView()

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

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

5.11.2.16 CreateDrawSettings()

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

Creates a DrawSettings structure with the specified name.

5.11.2.17 CreateIndexBuffer()

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

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

5.11.2.18 CreateText()

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

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

5.11.2.19 DisableMSAA()

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

Disables MSAA.

5.11.2.20 DrawObjects()

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

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

Ex.
beforeDrawObjects()
drawObjects()
drawObjects()
afterDrawObjects()
Throws an out_of_range exception if the specified DrawSettings structure does not exist.

5.11.2.21 EnableMSAA()

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

Enables MSAA.

5.11.2.22 ExecuteAndFlush()

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

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

5.11.2.23 IsMSAAEnabled()

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

Returns true if MSAA is enabled, false otherwise.

5.11.2.24 NextFrame()

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

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

5.11.2.25 RemoveDrawArgument()

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

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

5.11.2.26 RemoveDrawSettings()

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

Removes the specified DrawSettings structure. If the DrawSettings structure does not exist an out_of_range exception is thrown.

5.11.2.27 RemoveText()

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

Removes the specified text object with the specified name. If the Text object does not exist an out_of_range exception is thrown.

5.11.2.28 RenderText()

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

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

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

5.11.2.29 Resize()

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

Resizes the DeviceResources resources when the window gets resized.

5.11.2.30 SetPrimitive()

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

Sets the Primitive in the specified DrawSettings structure to the specified primitive. If the specifed DrawSettings structure does not exist an out_of_range exception is thrown.

5.11.2.31 SetPSO()

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

Sets the PSO in the specified DrawSettings structure to the specified pso. If the specifed DrawSettings structure does not exist an out_of_range exception is thrown.

5.11.2.32 SetRootSignature()

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

Sets the root signature in the specified DrawSettings structure to the specified root signature. If the specifed DrawSettings structure does not exist an out_of_range exception is thrown.

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

- FARenderScene.h

5.12 FARender::RenderTargetBuffer Class Reference

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

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

Public Member Functions

- RenderTargetBuffer (DXGI_FORMAT format=DXGI_FORMAT_R8G8B8A8_UNORM)

  Default Constructor.
- DXGI_FORMAT GetRenderTargetFormat () const

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

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

  Returns a constant reference to the render target buffer.
- void CreateRenderTargetBufferAndView (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &rtvHeap, unsigned int 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 index↩ OfView, unsigned int rtvSize, const float ∗clearValue)

  Clears the render target buffer with the specified clear value.

5.12.1 Detailed Description

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

5.12.2 Constructor & Destructor Documentation

5.12.2.1 RenderTargetBuffer()

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

Default Constructor.

5.12.3 Member Function Documentation

5.12.3.1 ClearRenderTargetBuffer()

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

Clears the render target buffer with the specified clear value.

5.12.3.2 CreateRenderTargetBufferAndView()

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

Creates the render target buffer and view.

5.12.3.3 GetRenderTargetBuffer() [1/2]

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

Returns a reference to the render target buffer.

5.12.3.4 GetRenderTargetBuffer() [2/2]

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

Returns a constant reference to the render target buffer.

5.12.3.5 GetRenderTargetFormat()

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

Returns the format of the render target buffer.

5.12.3.6 ResetBuffer()

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

Resest the render target buffer.

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

- FABuffer.h

5.13 FARender::SwapChain Class Reference

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

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

Public Member Functions

- SwapChain (const Microsoft::WRL::ComPtr< IDXGIFactory4 > &dxgiFactory, const Microsoft::WRL↩ ::ComPtr< ID3D12CommandQueue > &commandQueue, HWND windowHandle, DXGI_FORMAT rtFormat=DXGI_FORMAT_R8G8B8A8_UNORM, DXGI_FORMAT dsFormat=DXGI_FORMAT_D24_↩ UNORM_S8_UINT, unsigned int numRenderTargetBuffers=2)

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

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

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

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

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

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

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

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

    Resizes the swap chain.
- void CreateRenderTargetBuffersAndViews (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &rtvHeap, unsigned int indexOfWhereToStoreFirst↩ View, 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 > &command↩ List, 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 > &command↩ List, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &dsvHeap, unsigned int indexOfView, unsigned int dsvSize, float clearValue)

    Clears the swap chains depth stencil buffer with the specified clear value.
- void Transition (const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, D3D12↩ _RESOURCE_STATES before, D3D12_RESOURCE_STATES after)

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

    Swaps the front and back buffers.

5.13.1 Detailed Description

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

5.13.2 Constructor & Destructor Documentation

5.13.2.1 SwapChain()

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

Constructor. Creates a swap chain.

5.13.3 Member Function Documentation

5.13.3.1 ClearCurrentBackBuffer()

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

Clears the current render target buffer.

5.13.3.2 ClearDepthStencilBuffer()

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

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

5.13.3.3 CreateDepthStencilBufferAndView()

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

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

5.13.3.4 CreateRenderTargetBuffersAndViews()

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

Creates the render target buffers and views to them.

5.13.3.5 GetBackBufferFormat()

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

Returns the format of the swap chain.

5.13.3.6 GetCurrentBackBuffer()

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

Returns a constant reference to the current render target buffer.

5.13.3.7 GetCurrentBackBufferIndex()

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

Returns the current back buffer index.

5.13.3.8 GetDepthStencilFormat()

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

Returns the format of the depth stencil buffer.

5.13.3.9 GetNumRenderTargetBuffers()

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

Returns the number of swap chain buffers.

5.13.3.10 GetRenderTargetBuffers()

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

Returns a constant pointer to the render target buffers.

5.13.3.11 Present()

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

Swaps the front and back buffers.

5.13.3.12 ResetBuffers()

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

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

5.13.3.13 ResizeSwapChain()

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

Resizes the swap chain.

5.13.3.14 Transition()

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

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

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

- FASwapChain.h

5.14 FARender::Text Class Reference

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

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

Public Member Functions

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

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

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

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

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

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

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

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

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

  Changes the text location to the specified location.

5.14.1 Detailed Description

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

5.14.2 Constructor & Destructor Documentation

5.14.2.1 Text()

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

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

5.14.3 Member Function Documentation

5.14.3.1 GetTextColor()

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

Returns a constant reference to the text color.

5.14.3.2 GetTextLocation()

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

Returns a constant reference to the text location.

5.14.3.3 GetTextSize()

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

Returns the text size.

5.14.3.4 GetTextString()

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

Returns a constant reference to the text string.

5.14.3.5 SetTextColor()

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

Changes the text color to the specified color.

5.14.3.6 SetTextLocation()

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

Changes the text location to the specified location.

5.14.3.7 SetTextSize()

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

Changes the text size to the specified size.

5.14.3.8 SetTextString()

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

Changes the text string to the specified string.

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

- FAText.h

5.15 FARender::TextResources Class Reference

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

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

Public Member Functions

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

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

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

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

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

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

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

  Executes text commands.

5.15.1 Detailed Description

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

5.15.2 Constructor & Destructor Documentation

5.15.2.1 TextResources()

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

Constructor. Initializes the text resources.

5.15.3 Member Function Documentation

5.15.3.1 AfterRenderText()

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

Executes text commands.

5.15.3.2 BeforeRenderText()

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

Prepares to render text.

5.15.3.3 GetDirect2DDeviceContext()

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

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

5.15.3.4 GetDirectWriteFactory()

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

Returns a constant reference to the direct direct write factory.

5.15.3.5 ResetBuffers()

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

Resets the text buffers.

5.15.3.6 ResizeBuffers()

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

Resizes the buffers.

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

- FATextResources.h

5.16 FATime::Time Class Reference

Public Member Functions

- Time ()

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

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

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

  Resets all time variables.
- void Stop ()

  Stops the timer.
- void Start ()

  Starts the timer.
- float TotalTime () const

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

5.16.1 Constructor & Destructor Documentation

5.16.1.1 Time()

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

Default Constructor. Gets and stores the seconds per count.

5.16.2 Member Function Documentation

5.16.2.1 DeltaTime()

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

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

5.16.2.2 Reset()

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

Resets all time variables.

5.16.2.3 Start()

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

Starts the timer.

5.16.2.4 Stop()

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

Stops the timer.

5.16.2.5 Tick()

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

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

5.16.2.6 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

5.17 Time Class Reference

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

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

5.17.1 Detailed Description

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

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

- FATime.h

5.18 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.

5.18.1 Detailed Description

This class stores vertices in a Direct3D 12 default buffer.

5.18.2 Member Function Documentation

5.18.2.1 CreateVertexBuffer()

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

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

5.18.2.2 CreateVertexBufferView()

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

Creates the vertex buffer view and stores it.

5.18.2.3 GetVertexBufferView()

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

Returns a constant reference to the vertex buffer view.

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

- FABuffer.h

5.19 FAWindow::Window Class Reference

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

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

Public Member Functions

- Window (const HINSTANCE &hInstance, const std::wstring &windowClassName, const std::wstring &windowName, WNDPROC winProcFunction, unsigned int width, unsigned int height, void ∗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.

5.19.1 Detailed Description

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

5.19.2 Constructor & Destructor Documentation

5.19.2.1 Window() [1/2]

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

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

5.19.2.2 Window() [2/2]

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

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

5.19.3 Member Function Documentation

5.19.3.1 GetHeight()

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

Returns the height of the window.

5.19.3.2 GetWidth()

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

Returns the width of the window.

5.19.3.3 GetWindowHandle()

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

Returns the window handle.

5.19.3.4 SetHeight()

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

Sets the height of the window o the specified height.

5.19.3.5 SetWidth()

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

Sets the width of the window to the specified width.

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

- FAWindow.h

Chapter 6

File Documentation

6.1 Direct3DLink.h

```
#pragma once

//Link necessary libraries.
#pragma comment(lib, "D3D12.lib")
#pragma comment(lib, "dxgi.lib")
#pragma comment(lib, "dxguid.lib")
#pragma comment(lib, "d3dcompiler.lib")
#pragma comment(lib, "D3D11.lib")
#pragma comment(lib, "D2D1.lib")
#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.

```cpp
#pragma once

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

namespace FARender
{
    class RenderTargetBuffer
    {
    public:
        RenderTargetBuffer(DXGI_FORMAT format = DXGI_FORMAT_R8G8B8A8_UNORM);

        DXGI_FORMAT GetRenderTargetFormat() const;

        Microsoft::WRL::ComPtr<ID3D12Resource>& GetRenderTargetBuffer();

        const Microsoft::WRL::ComPtr<ID3D12Resource>& GetRenderTargetBuffer() const;

        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);

        void ResetBuffer();

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

    private:
        Microsoft::WRL::ComPtr<ID3D12Resource> mRenderTargetBuffer;
        DXGI_FORMAT mRenderTargetFormat;

    };

    class DepthStencilBuffer
    {
    public:

        DepthStencilBuffer(DXGI_FORMAT format = DXGI_FORMAT_D24_UNORM_S8_UINT);

        DXGI_FORMAT GetDepthStencilFormat() const;

        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);

        void ResetBuffer();

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

    private:
        Microsoft::WRL::ComPtr<ID3D12Resource> mDepthStencilBuffer;
        DXGI_FORMAT mDepthStencilFormat;
    };


    class VertexBuffer
    {
    public:
        VertexBuffer() = default;
        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);

        void CreateVertexBufferView(UINT numBytes, UINT stride);

        const D3D12_VERTEX_BUFFER_VIEW& GetVertexBufferView();

    private:
        Microsoft::WRL::ComPtr<ID3D12Resource> mVertexDefaultBuffer;
        Microsoft::WRL::ComPtr<ID3D12Resource> mVertexUploadBuffer;
        D3D12_VERTEX_BUFFER_VIEW mVertexBufferView{};
    };

    class IndexBuffer
    {
    public:
        IndexBuffer() = default;
        IndexBuffer(const IndexBuffer&) = delete;
        IndexBuffer& operator=(const IndexBuffer&) = delete;

        const D3D12_INDEX_BUFFER_VIEW& GetIndexBufferView();

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

        void CreateIndexBufferView(UINT numBytes, DXGI_FORMAT format);

    private:
        Microsoft::WRL::ComPtr<ID3D12Resource> mIndexDefaultBuffer;
        Microsoft::WRL::ComPtr<ID3D12Resource> mIndexUploadBuffer;
        D3D12_INDEX_BUFFER_VIEW mIndexBufferView{};
    };

    class ConstantBuffer
    {
    public:
        ConstantBuffer() = default;

        ConstantBuffer(const ConstantBuffer&) = delete;
        ConstantBuffer& operator=(const ConstantBuffer&) = delete;

        ~ConstantBuffer();

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

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

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

    private:
        Microsoft::WRL::ComPtr<ID3D12Resource> mConstantBuffer;
        BYTE* mMappedData{ nullptr };
    };
}
```

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.

```
#pragma once

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

typedef FAMath::Vector2D vec2;
typedef FAMath::Vector3D vec3;
typedef FAMath::Vector4D vec4;
typedef FAMath::Matrix4x4 mat4;

namespace FACamera
{
    class Camera
    {
    public:
        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);

        const vec3& GetCameraPosition() const;

        const vec3& GetX() const;

        const vec3& GetY() const;

        const vec3& GetZ() const;

        const mat4& GetViewTransformationMatrix() const;

        float GetCameraVelocity() const;

        float GetAngularVelocity() const;

        void LookAt(vec3 cameraPosition, vec3 target, vec3 up);

        float GetZNear() const;

        float GetZFar() const;

        float GetVerticalFov() const;

        float GetAspectRatio() const;

        void SetCameraPosition(const vec3& position);

        void SetX(const vec3& x);

        void SetY(const vec3& y);

        void SetZ(const vec3& z);

        void SetCameraVelocity(float velocity);

        void SetAngularVelocity(float velcoity);

        void SetZNear(float znear);

        void SetZFar(float zfar);

        void SetVerticalFov(float fov);

        void SetAspectRatio(float ar);

        const mat4& GetPerspectiveProjectionMatrix() const;

        const mat4& GetViewPerspectiveProjectionMatrix() const;

        void UpdateViewMatrix();

        void UpdatePerspectiveProjectionMatrix();

        void UpdateViewPerspectiveProjectionMatrix();

        void Left(float dt);

        void Right(float dt);

        void Foward(float dt);

        void Backward(float dt);

        void Up(float dt);

        void Down(float dt);

        void RotateCameraLeftRight(float xDiff);

        void RotateCameraUpDown(float yDiff);

        void KeyboardInput(float dt);

        void MouseInput();

    private:
        //camera position in world coordinates
        vec3 mCameraPosition;

        //z-axis of the camera coordinate system
        vec3 mN;

        //y-axis of the camera coordinate system
        vec3 mV;

        //x-axis of the camera coordinate system
        vec3 mU;

        //stores the world to camera transform
        mat4 mViewMatrix;

        //frustrum properties
        float mNear;
        float mFar;
        float mVerticalFov;
        float mAspectRatio;
        mat4 mPerspectiveProjectionMatrix;

        mat4 mViewPerspectiveProjectionMatrix;

        float mCameraVelocity;
        float mAngularVelocity;

        vec2 mLastMousePosition;
    };
}
```

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.

```
#pragma once

#include "FAMathEngine.h"

namespace FAColor
{
    class Color
    {
    public:

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

        Color(const FAMath::Vector4D& color);

        const FAMath::Vector4D& GetColor() const;

        float GetRed() const;

        float GetGreen() const;

        float GetBlue() const;

        float GetAlpha() const;

        void SetColor(const FAMath::Vector4D& color);

        void SetRed(float r);

        void SetGreen(float g);

        void SetBlue(float b);

        void SetAlpha(float a);

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

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

        Color& operator*=(float k);

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

    private:
        FAMath::Vector4D mColor;
    };

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

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

    Color operator*(const Color& c, float k);

    Color operator*(float k, const Color& c);

    Color operator*(const Color& c1, const Color& c2);
}
```

6.8 FADepthStencil.h

```
#pragma once

#include <wrl.h>
#include "d3dx12.h"

namespace FARender
{
    class DepthStencilBuffer
    {
    public:

        DepthStencilBuffer(DXGI_FORMAT format = DXGI_FORMAT_D24_UNORM_S8_UINT);

        DXGI_FORMAT GetDepthStencilFormat() const;

        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);
        void ResetBuffer();

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

    private:
        Microsoft::WRL::ComPtr<ID3D12Resource> mDepthStencilBuffer;
        DXGI_FORMAT mDepthStencilFormat;
    };
}
```

6.9 FADeviceResources.h File Reference

File has class DeviceResources under namespace FARender.

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

Classes

- class FARender::DeviceResources

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

Namespaces

- namespace FARender

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

6.9.1 Detailed Description

File has class DeviceResources under namespace FARender.

6.10 FADeviceResources.h

Go to the documentation of this file.

```
#pragma once

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

namespace FARender
{
    class DeviceResources
    {
```

```
    public:
        static const unsigned int NUM_OF_FRAMES{ 3 };

        static DeviceResources& GetInstance(unsigned int width, unsigned int height, HWND windowHandle);

        DeviceResources(const DeviceResources&) = delete;
        DeviceResources& operator=(const DeviceResources&) = delete;

        ~DeviceResources();

        const Microsoft::WRL::ComPtr<ID3D12Device>& GetDevice() const;

        const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& GetCommandList() const;

        DXGI_FORMAT GetBackBufferFormat() const;

        DXGI_FORMAT GetDepthStencilFormat() const;

        UINT GetCBVSize() const;

        unsigned int GetCurrentFrame() const;

        const TextResources& GetTextResources() const;

        bool IsMSAAEnabled() const;

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

        void EnableMSAA(unsigned int width, unsigned int height, HWND windowHandle);

        void UpdateCurrentFrameFenceValue();

        void FlushCommandQueue();

        void WaitForGPU() const;

        void Signal();

        void Resize(int width, int height, const HWND& handle);

        void RTBufferTransition(bool renderText);

        void BeforeTextDraw();

        void AfterTextDraw();

        void Execute() const;

        void Present();

        /*@brief Calls the necessary functions to let the user draw their objects.
*/
        void Draw();

        void NextFrame();

    private:

        DeviceResources(unsigned int width, unsigned int height, HWND windowHandle);

        unsigned int mCurrentFrameIndex{ 0 };

        Microsoft::WRL::ComPtr<ID3D12Device> mDirect3DDevice;

        Microsoft::WRL::ComPtr<IDXGIFactory4> mDXGIFactory;

        Microsoft::WRL::ComPtr<ID3D12Fence> mFence;
        UINT64 mFenceValue{ 0 };
        UINT64 mCurrentFrameFenceValue[NUM_OF_FRAMES];

        Microsoft::WRL::ComPtr<ID3D12CommandQueue> mCommandQueue;
        Microsoft::WRL::ComPtr<ID3D12CommandAllocator> mCommandAllocator[NUM_OF_FRAMES];
        Microsoft::WRL::ComPtr<ID3D12CommandAllocator> mDirectCommandAllocator;
        Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList> mCommandList;

        UINT mRTVSize;
        UINT mDSVSize;
        UINT mCBVSize;

        Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mRTVHeap;
        Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mDSVHeap;

        SwapChain mSwapChain;

        bool mIsMSAAEnabled{ false };
        MultiSampling mMultiSampling;

        D3D12_VIEWPORT mViewport{};
        D3D12_RECT mScissor{};

        TextResources mTextResources;

        //Call all of these functions to initialize Direct3D
        void mEnableDebugLayer();
        void mCreateDirect3DDevice();
        void mCreateDXGIFactory();
        void mCreateFence();
        void mQueryDescriptorSizes();
        void mCreateRTVHeap();
        void mCreateDSVHeap();
        void mCreateCommandObjects();
    };
}
```

6.11 FADirectXException.h

```
#pragma once

#include <wrl.h>
#include <dxgidebug.h>
#include <comdef.h>
#include <string>
#include <sstream>
#include <vector>

inline std::wstring AnsiToWString(const std::string& str)
{
    WCHAR buffer[1024];
    MultiByteToWideChar(CP_ACP, 0, str.c_str(), -1, buffer, 1024);
    return std::wstring(buffer);
}

class DirectXException
{
public:
    DirectXException(HRESULT hr, const std::wstring& functionName, const std::wstring& fileName, int
   lineNumber);

    std::wstring ErrorMsg() const;

private:
    HRESULT errorCode;
    std::wstring functionName;
    std::wstring fileName;
    int lineNumber;
    Microsoft::WRL::ComPtr<IDXGIInfoQueue> mInfoQueue;
};

//use when calling Direct3D or DXGI function to check if the function failed or not.
#ifndef ThrowIfFailed
#define ThrowIfFailed(x)                                                    \
{                                                                           \
HRESULT hr = (x);                                                       \
std::wstring filename(AnsiToWString(__FILE__));                         \
if (FAILED(hr)) { throw DirectXException(hr, L#x, filename, __LINE__); }    \
}
#endif
```

6.12 FAMultiSampling.h

```
#pragma once

#include <wrl.h>
#include "d3dx12.h"
#include "FABuffer.h"

namespace FARender
{
    class MultiSampling
    {
    public:

        MultiSampling() = default;

        MultiSampling(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
            DXGI_FORMAT rtFormat, DXGI_FORMAT dsFormat, unsigned int sampleCount);
        const Microsoft::WRL::ComPtr<ID3D12Resource>& GetRenderTargetBuffer();

        DXGI_FORMAT GetRenderTargetFormat();

        DXGI_FORMAT GetDepthStencilFormat();

        void ResetBuffers();

        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);

        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);

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

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

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

    private:
        RenderTargetBuffer mMSAARenderTargetBuffer;
        DepthStencilBuffer mMSAADepthStencilBuffer;
        unsigned int mSampleCount{ 0 };

        /*Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mMSAARTVDescriptorHeap;
Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mMSAADSVDescriptorHeap;
Microsoft::WRL::ComPtr<ID3D12Resource> mMSAARenderTargetBuffer;
Microsoft::WRL::ComPtr<ID3D12Resource> mMSAADepthStencilBuffer;*/

    };
}
```

6.13 FARenderScene.h File Reference

File has class RenderScene under namespace FARender.

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

Classes

- struct FARender::DrawSettings

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

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

Namespaces

- namespace FARender

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

6.13.1 Detailed Description

File has class RenderScene under namespace FARender.

6.14 FARenderScene.h

Go to the documentation of this file.

```cpp
#pragma once

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

namespace FARender
{
    struct DrawSettings
    {
        Microsoft::WRL::ComPtr<ID3D12PipelineState> pipelineState;
        Microsoft::WRL::ComPtr<ID3D12RootSignature> rootSig;
        D3D_PRIMITIVE_TOPOLOGY prim = D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST;
        std::vector<FAShapes::DrawArguments> drawArgs;
    };


    class RenderScene
    {
    public:

        RenderScene(unsigned int width, unsigned int height, HWND windowHandle);

        RenderScene(const RenderScene&) = delete;
        RenderScene& operator=(const RenderScene&) = delete;

        /*@brief Returns a constant reference to the device resources object.
*/
        const DeviceResources& GetDeviceResources() const;

        /*@brief Returns a constant reference to the shader with the specified name.
* Throws an out_of_range exception if the shader does not exist.
*/
        const Microsoft::WRL::ComPtr<ID3DBlob>& GetShader(const std::wstring& name) const;

        /*@brief Returns a constant reference to the specified array of input element layout descriptions.
* Throws an out_of_range exception if the array of input element layout descriptions does not exist.
*/
        const std::vector<D3D12_INPUT_ELEMENT_DESC>& GetInputElementLayout(const std::wstring& name) const;

        /*@brief Returns a constant reference to the specified rasterization description.
* Throws an out_of_range exception if the rasterization description does not exist.
*/
        const D3D12_RASTERIZER_DESC& GetRasterizationState(const std::wstring& name) const;

        /*@brief Returns a constant reference to the PSO in the specified DrawSettings.
* Throws an out_of_range exception if the DrawSettings does not exist.
*/
        const Microsoft::WRL::ComPtr<ID3D12PipelineState>& GetPSO(const std::wstring& drawSettingsName) const;

        /*@brief Returns a constant reference to the root signature in the specified DrawSettings structure.
* Throws an out_of_range exception if the DrawSettings does not exist.
*/
```

```
        const Microsoft::WRL::ComPtr<ID3D12RootSignature>& GetRootSignature(const std::wstring& drawSettingsName) const;

        /*@brief Returns a constant reference to the primitive in the specified DrawSettings structure.
* Throws an out_of_range exception if the DrawSettings does not exist.
*/
        const D3D_PRIMITIVE_TOPOLOGY& GetPrimitive(const std::wstring& drawSettingsName) const;

        /*@brief Returns a reference to the specified DrawArguments object in the specified DrawSettings structure.
* Throws an out_of_range exception if the DrawSettings does not exist or if the index is out of range.
*/
        FAShapes::DrawArguments& GetDrawArguments(const std::wstring& drawSettingsName, unsigned int index);

        /*@brief Returns a constant reference to the specified DrawArguments object in the specified DrawSettings structure.
* Throws an out_of_range exception if the DrawSettings does not exist or if the index is out of range.
*/
        const FAShapes::DrawArguments& GetDrawArguments(const std::wstring& drawSettingsName, unsigned int index) const;

        /*@brief Returns a reference to the this scene's camera;
*/
        FACamera::Camera& GetCamera();

        /*@brief Returns a constant reference to the this scene's camera;
*/
        const FACamera::Camera& GetCamera() const;

        /*@brief Returns a reference to the specified Text object.
* If the Text object does not exist an out_of_range exception is thrown.
*/
        FARender::Text& GetText(std::wstring textName);

        /*@brief Returns a constant reference to the specified Text object.
* If the Text object does not exist an out_of_range exception is thrown.
*/
        const FARender::Text& GetText(std::wstring textName) const;

        /*@brief Loads a shader's bytecode and stores it with the specified name.
*/
        void LoadShader(const std::wstring& filename, const std::wstring& name);

        /*@brief Removes the specified shader.
* If the specified shader does not exist an out_of_range exception is thrown.
*/
        void RemoveShader(const std::wstring& shaderName);

        /*@brief Stores an array of input element descriptions with the specified name.
*/
        void StoreInputElementDescriptions(const std::wstring& name, const std::vector<D3D12_INPUT_ELEMENT_DESC>& inputElementLayout);

        /*@brief Stores an array of input element descriptions with the specified name.
*/
        void StoreInputElementDescriptions(const std::wstring& name, const D3D12_INPUT_ELEMENT_DESC* inputElementLayout,
                UINT numElements);

        /*@brief Removes the specified input element description.
* If the specified input element description does not exist an out_of_range exception is thrown.
*/
        void RemoveInputElementDescription(const std::wstring& name);

        /*@brief Creates a rasterization state description and stores it with the specified name.
*/
        void CreateRasterizationState(D3D12_FILL_MODE fillMode, BOOL enableMultisample, const std::wstring& name);

        /*@brief Removes the specified rasterization state.
* If the specified rasterization state does not exist an out_of_range exception is thrown.
*/
        void RemoveRasterizationState(const std::wstring& name);

        /*@brief Creates a PSO and stores it in the specified DrawSettings structure.
* If the specifed DrawSettings structure, Rasterization State, Vertex Shader, Pixel Shader or Input Layout
* does not exist an out_of_range exception is thrown.
*/
        void CreatePSO(const std::wstring& drawSettingsName, const std::wstring& rStateName,
            const std::wstring& vsName, const std::wstring& psName, const std::wstring& inputLayoutName,
            const D3D12_PRIMITIVE_TOPOLOGY_TYPE& primitiveType, UINT sampleCount);

        /*@brief Creates a root signature and stores it with the specified name.
* If the specifed DrawSettings structure does not exist an out_of_range exception is thrown.
*/
```

```
        void CreateRootSignature(const std::wstring& drawSettingsName);

        /*@brief Creates a vertex buffer with the specified name and stores all of the added vertices.
 * Also creates a view to the vertex buffer.\n
 * Execute commands and the flush command queue after calling createVertexBuffer() and
   createIndexBuffer().
 */
        void CreateVertexBuffer();

        void CreateIndexBuffer();

        void CreateCBVHeap(UINT numDescriptors, UINT shaderRegister);

        void CreateConstantBuffer(UINT numOfBytes);

        void CreateConstantBufferView(UINT index, UINT numBytes);

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

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

        void SetPrimitive(const std::wstring& drawSettingsName, const D3D_PRIMITIVE_TOPOLOGY&
   primitive);

        void AddDrawArgument(const std::wstring& drawSettingsName, const FAShapes::DrawArguments&
   drawArg);

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

        void RemoveDrawArgument(const std::wstring& drawSettingsName, unsigned int index);

        void CreateDrawSettings(const std::wstring& drawSettingsName);

        void RemoveDrawSettings(const std::wstring& drawSettingsName);

        void CreateText(const std::wstring& textName, FAMath::Vector4D textLocation, const std::wstring&
   textString,
            float textSize, const FAColor::Color textColor);

        void RemoveText(const std::wstring& textName);

        void AddVertices(const std::vector<FAShapes::Vertex>& vertices);

        void AddVertices(const FAShapes::Vertex* vertices, unsigned int numVertices);

        void AddIndices(const std::vector<unsigned int>& indices);

        void AddIndices(const unsigned int* indices, unsigned int numIndices);

        void BeforeDrawObjects();

        void DrawObjects(const std::wstring& drawSettingsName);

        void AfterDrawObjects(bool renderText);

        void BeforeDrawText();

        void RenderText(const std::wstring& textName);

        void AfterDrawText();

        void AfterDraw();

        void ExecuteAndFlush();

        void NextFrame();

        void Resize(unsigned int width, unsigned int height, HWND windowHandle);

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

        bool IsMSAAEnabled() const;

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

        void EnableMSAA(unsigned int width, unsigned int height, HWND windowHandle);

    private:

        //The device resources object that all RenderScene objects share.
        DeviceResources& mDeviceResources;

        //Stores all of the shaders and input element descriptions for this scene.
        std::unordered_map<std::wstring, Microsoft::WRL::ComPtr<ID3DBlob» mShaders;
        std::unordered_map < std::wstring, std::vector<D3D12_INPUT_ELEMENT_DESC»
    mInputElementDescriptions;

        //Stores all of the rasterization states.
        std::unordered_map <std::wstring, D3D12_RASTERIZER_DESC> mRasterizationStates;

        //Stores all of the possible draw settings that the scene uses.
        std::unordered_map <std::wstring, DrawSettings> mSceneObjects;

        //Each scene gets a CBV heap.
        Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mCBVHeap;
        D3D12_DESCRIPTOR_RANGE mCBVHeapDescription{};
        D3D12_ROOT_PARAMETER mCBVHeapRootParameter;

        //Stores all of the constant buffers this scene uses.  We can't update a constant buffer until
    the GPU
        //is done executing all the commands that reference it, so each frame needs its own constant
    buffer.
        ConstantBuffer mConstantBuffer[DeviceResources::NUM_OF_FRAMES];

        //The vertices and indicies for the scene.
        std::vector<FAShapes::Vertex> mVertexList;
        std::vector<unsigned int> mIndexList;

        //The vertex and index buffer for the scene.
        VertexBuffer mVertexBuffer;
        IndexBuffer mIndexBuffer;

        //All of the text that is rendered with the scene.
        std::unordered_map <std::wstring, Text> mTexts;

        //The camera for the scene.
        FACamera::Camera mCamera;
    };
}
```

6.15 FASwapChain.h

```
#pragma once

#include <wrl.h>
#include "d3dx12.h"
#include <dxgi1_4.h>
#include <vector>
#include "FABuffer.h"

namespace FARender
{
    class SwapChain
    {
    public:

        SwapChain() = default;

        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);

        const RenderTargetBuffer* GetRenderTargetBuffers() const;

        const Microsoft::WRL::ComPtr<ID3D12Resource>& GetCurrentBackBuffer() const;

        unsigned int GetNumRenderTargetBuffers() const;

        unsigned int GetCurrentBackBufferIndex() const;

        DXGI_FORMAT GetBackBufferFormat() const;

        DXGI_FORMAT GetDepthStencilFormat() const;

        void ResetBuffers();

        void ResizeSwapChain(unsigned width, unsigned height);

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

        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);

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

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

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

        void Present();

    private:
        unsigned int mNumRenderTargetBuffers = 0;
        unsigned int mCurrentBackBufferIndex = 0;

        Microsoft::WRL::ComPtr<IDXGISwapChain1> mSwapChain;
        std::vector<RenderTargetBuffer> mRenderTargetBuffers;

        DepthStencilBuffer mDepthStencilBuffer;
    };
}
```

6.16 FAText.h File Reference

File has class Text under namespace FARender.

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

Classes

- class FARender::Text

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

Namespaces

- namespace FARender

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

6.16.1 Detailed Description

File has class Text under namespace FARender.

6.17 FAText.h

Go to the documentation of this file.

```
#pragma once

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

namespace FARender
{
    class Text
    {
    public:

        Text() = default;

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

        const FAMath::Vector4D& GetTextLocation() const;

        const std::wstring& GetTextString() const;

        float GetTextSize() const;

        const FAColor::Color& GetTextColor() const;

        void SetTextSize(float textSize);

        void SetTextColor(const FAColor::Color& textColor);

        void SetTextString(const std::wstring& textString);

        void SetTextLocation(const FAMath::Vector4D& textLocation);

    private:

        FAMath::Vector4D mTextLocation;
        std::wstring mText;
        float mTextSize{ 0.0f };
        FAColor::Color mTextColor;
    };
}
```

6.18 FATextResources.h

```
#pragma once

#include <wrl.h>
#include <d3d11.h>
#include <d3d11on12.h>
#include <d2d1_3.h>
#include <dwrite.h>
#include <vector>
#include "FABuffer.h"

namespace FARender
{
    class TextResources
    {
    public:
        TextResources() = default;

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

        const Microsoft::WRL::ComPtr<ID2D1DeviceContext>& GetDirect2DDeviceContext() const;

        const Microsoft::WRL::ComPtr<IDWriteFactory>& GetDirectWriteFactory() const;

        void ResetBuffers();

        void ResizeBuffers(const RenderTargetBuffer* renderTargetBuffers, HWND windowHandle);

        void BeforeRenderText(unsigned int currentBackBuffer);

        void AfterRenderText(unsigned int currentBackBuffer);

    private:
        Microsoft::WRL::ComPtr<ID3D11Device> mDevice11;
        Microsoft::WRL::ComPtr<ID3D11DeviceContext> mDevice11Context;
        Microsoft::WRL::ComPtr<ID3D11On12Device> mDevice11on12;

        Microsoft::WRL::ComPtr<ID2D1Device2> mDirect2DDevice;
        Microsoft::WRL::ComPtr<ID2D1Factory3> mDirect2DFactory;
        Microsoft::WRL::ComPtr<ID2D1DeviceContext> mDirect2DDeviceContext;

        Microsoft::WRL::ComPtr<IDWriteFactory> mDirectWriteFactory;

        std::vector<Microsoft::WRL::ComPtr<ID3D11Resource» mWrappedBuffers;
        std::vector<Microsoft::WRL::ComPtr<ID2D1Bitmap1» mDirect2DBuffers;
        std::vector<Microsoft::WRL::ComPtr<IDXGISurface» mSurfaces;
    };
}
```

6.19 FATime.h File Reference

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

`#include <Windows.h>`

Classes

- class FATime::Time

6.19.1 Detailed Description

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

6.20 FATime.h

Go to the documentation of this file.

```
#pragma once

#include <Windows.h>

namespace FATime
{
    class Time
    {
    public:
        Time();

        void Tick();

        float DeltaTime() const;

        void Reset();

        void Stop();

        void Start();

        float TotalTime() const;

    private:
        __int64 mCurrTime; //holds current time stamp ti
        __int64 mPrevTime; //holds previous time stamp ti-1
        __int64 mStopTime; //holds the time we stopped the game/animation
        __int64 mPausedTime; //holds how long the game/animation was paused for
        __int64 mBaseTime; //holds the time we started / resetted

        double mSecondsPerCount;
        double mDeltaTime; //time elapsed btw frames change in t = ti - ti-1

        bool mStopped; //flag to indicate if the game/animation is paused or not

    };
}
```

6.21 FAWindow.h File Reference

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

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

Classes

- class FAWindow::Window

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

Namespaces

- namespace FAWindow

    Has Window class.

6.21.1 Detailed Description

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

6.22 FAWindow.h

Go to the documentation of this file.

```
#pragma once

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

namespace FAWindow
{
    class Window
    {
    public:
        //Window();

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

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

        HWND GetWindowHandle() const;

        unsigned int GetWidth() const ;

        unsigned int GetHeight() const;

        void SetWidth(unsigned int width);

        void SetHeight(unsigned int height);

    private:
        HWND mWindowHandle;

        WNDCLASSEX mWindowClass;
        std::wstring mWindowClassName;

        unsigned int mWidth;
        unsigned int mHeight;
    };
}
```

Index