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 FAColor Namespace Reference

Has the Color class.

Classes

- class Color

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

Functions

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

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

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

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

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

  Returns the result of c1 ∗ c2.

4.2.1 Detailed Description

Has the Color class.

4.2.2 Function Documentation

4.2.2.1 operator∗() [1/3]

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

Returns the result of *c* ∗ *k*.

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

4.2.2.2 operator∗() [2/3]

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

Returns the result of *c1* ∗ *c2*.

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

4.2.2.3 operator∗() [3/3]

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

Returns the result of $k * c$.

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

4.2.2.4 operator+()

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

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

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

4.2.2.5 operator-()

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

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

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

4.3 FARender Namespace Reference

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

Classes

- class DepthStencilBuffer
  A wrapper for depth stencil buffer resources. Uses DirectD 12 API.
- class DeviceResources
  A wrapper for resources that are needed to render objects and text using the Direct3D 12 API.
- class DynamicBuffer
  This class stores data in a Direct3D 12 upload buffer.
- class MultiSampling
  A wrapper for multisampling resources. Uses DirectD 12 API.
- class RenderScene
  This class is used to render a scene using Direct3D 12 API.
- class RenderTargetBuffer
  A wrapper for render target buffer resources. Uses DirectD 12 API.
- class StaticBuffer
  This class stores data in a Direct3D 12 default buffer.
- class SwapChain
  A wrapper for swap chain resources. Uses DirectD 12 API and DXGI API.
- class Text
  This class is used to help render text. Stores the location of the text, the text string, text size and the color of the text.
- class TextResources
  A wrapper for resources that are needed to render text using Direct3D 11on12, Direct2D and DirectWrite.

4.3.1 Detailed Description

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

4.4 FATime Namespace Reference

Has Time class.

Classes

- class Time

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

4.4.1 Detailed Description

Has Time class.

4.5 FAWindow Namespace Reference

Has Window class.

Classes

- class Window

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

4.5.1 Detailed Description

Has Window class.

Chapter 5

Class Documentation

5.1 FACamera::Camera Class Reference

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

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

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

Public Member Functions

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

  Creates a new camera.
- const vec3 & GetCameraPosition () const

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

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

  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 & GetViewMatrix () const

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

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

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

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

  Returns the near value of the 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 x.

- void SetY (const vec3 &y)

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

- void SetZ (const vec3 &z)

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

- void SetCameraVelocity (float velocity)

  Sets the camera's velocity to the specified velocity.

- void SetAngularVelocity (float velcoity)

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

- void SetZNear (float znear)

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

- void SetZFar (float zfar)

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

- void SetVerticalFov (float fov)

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

- void SetAspectRatio (float ar)

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

- const mat4 & GetPerspectiveProjectionMatrix () const

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

- const mat4 & GetViewPerspectiveProjectionMatrix () const

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

- void UpdateViewMatrix ()

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

- void UpdatePerspectiveProjectionMatrix ()

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

- void UpdateViewPerspectiveProjectionMatrix ()

  Updates the camera's view perspective projection matrix.

- void Left (float dt)

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

- void Right (float dt)

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

- void Foward (float dt)

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

- void Backward (float dt)

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

- void Up (float dt)

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

- void Down (float dt)

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

- void RotateCameraLeftRight (float xDiff)

  Rotates the camera to look left and right.

- void RotateCameraUpDown (float yDiff)

  Rotates the camera to look up and down.

- void KeyboardInput (float dt)

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

  Polls mouse input and rotates the camera.

5.1.1 Detailed Description

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

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

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Camera()

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

Creates a new camera.

Parameters

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

5.1.3 Member Function Documentation

5.1.3.1 Backward()

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

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

Parameters

in	*dt*	The time between frames.

5.1.3.2 Down()

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

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

Parameters

in	*dt*	The time between frames.

5.1.3.3 Foward()

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

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

Parameters

in	*dt*	The time between frames.

5.1.3.4 GetAngularVelocity()

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

Returns the camera's angular velocity.

5.1.3.5 GetAspectRatio()

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

Returns the aspect ratio of the 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 GetViewMatrix()

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

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

5.1.3.11 GetViewPerspectiveProjectionMatrix()

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

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

5.1.3.12 GetX()

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

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

5.1.3.13 GetY()

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

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.

Parameters

in	*dt*	The time between frames.

5.1.3.18 Left()

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

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

5.1.3.19 LookAt()

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

Defines the camera space using UVN.

Parameters

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

5.1.3.20 MouseInput()

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

Polls mouse input and rotates the camera.

5.1.3.21 Right()

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

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

Parameters

in	*dt*	The time between frames.

5.1.3.22 RotateCameraLeftRight()

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

Rotates the camera to look left and right.

Parameters

in	*xDiff*	How many degrees to rotate.

5.1.3.23 RotateCameraUpDown()

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

Rotates the camera to look up and down.

Parameters

in	*yDiff*	How many degrees to rotate.

5.1.3.24 SetAngularVelocity()

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

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

5.1.3.25 SetAspectRatio()

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

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

5.1.3.26 SetCameraPosition()

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

Sets the camera's position to the specified position.

5.1.3.27 SetCameraVelocity()

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

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

5.1.3.28 SetVerticalFov()

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

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

5.1.3.29 SetX()

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

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

5.1.3.30 SetY()

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

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

5.1.3.31 SetZ()

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

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

5.1.3.32 SetZFar()

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

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

5.1.3.33 SetZNear()

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

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

5.1.3.34 Up()

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

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

Parameters

in	*dt*	The time between frames.

5.1.3.35 UpdatePerspectiveProjectionMatrix()

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

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

5.1.3.36 UpdateViewMatrix()

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

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

5.1.3.37 UpdateViewPerspectiveProjectionMatrix()

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

Updates the camera's view perspective projection matrix.

Call this to rebuild the view perspective projection transformation matrix.

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

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

5.2.1 Detailed Description

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

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Color() [1/2]

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

Initializes the color to the specified RGBA values.

5.2.2.2 Color() [2/2]

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

Initializes the color to the specified *color*.

5.2.3 Member Function Documentation

5.2.3.1 GetAlpha()

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

Returns the value of the alpha component.

5.2.3.2 GetBlue()

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

Returns the value of the green component.

5.2.3.3 GetColor()

```
const 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 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 *c* and stores the result in this object.

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

5.2.3.9 operator-=()

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

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

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

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::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 ()
  Resets 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.

5.3.1 Detailed Description

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

5.3.2 Constructor & Destructor Documentation

5.3.2.1 DepthStencilBuffer()

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

Default Constructor.

Parameters

in	*format*	The format of the depth stencil buffer.

5.3.3 Member Function Documentation

5.3.3.1 ClearDepthStencilBuffer()

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

Clears the depth stencil buffer with the specified clear value.

Parameters

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

5.3.3.2 CreateDepthStencilBufferAndView()

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

Creates the depth stencil buffer and view.

Parameters

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

Parameters

in	*dsvSize*	The size of a depth stenicl descriptor.
in	*width*	The width of the depth stenicl buffer.
in	*height*	The height of the depth stenicl buffer.
in	*sampleCount*	The sample count of the depth stenicl buffer.

5.3.3.3 GetDepthStencilFormat()

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

Returns the format of the depth stencil buffer.

5.3.3.4 ResetBuffer()

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

Resets the depth stencil buffer.

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

- FABuffer.h

5.4 FARender::DeviceResources Class Reference

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

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

Public Member Functions

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

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

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

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

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

  Returns a constant reference to the depth stencil format.

- unsigned int GetCBVSize () const

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

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

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

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

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

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

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

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

  Transistions the render target buffer.
- void BeforeTextDraw ()

  Prepares to render text.
- void AfterTextDraw ()

  Executes the text commands.
- void Execute () const

  Executes the command list.
- void Present ()

  Swaps the front and back buffers.
- void **Draw** (bool isMSAAEnabled)
- void NextFrame ()

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

Static Public Member Functions

- static DeviceResources & GetInstance (unsigned int width, unsigned int height, HWND windowHandle, bool isMSAAEnabled, bool isTextEnabled)

  Call to make an object of DeviceResources.

Static Public Attributes

- static const unsigned int NUM_OF_FRAMES { 3 }

  The number of frames in the ciruclar array.

5.4.1 Detailed Description

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

5.4.2 Constructor & Destructor Documentation

5.4.2.1 ~DeviceResources()

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

Flushes the command queue.

5.4.3 Member Function Documentation

5.4.3.1 AfterTextDraw()

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

Executes the text commands.

5.4.3.2 BeforeTextDraw()

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

Prepares to render text.

5.4.3.3 Execute()

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

Executes the command list.

5.4.3.4 FlushCommandQueue()

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

Synchronizes the CPU and GPU.

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

5.4.3.5 GetBackBufferFormat()

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

Returns a constant reference to the back buffer format.

5.4.3.6 GetCBVSize()

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

The size of a constant buffer view.

5.4.3.7 GetCommandList()

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

Returns a constant reference to the ID3D12GraphicsCommandList objcet.

5.4.3.8 GetCurrentFrame()

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

Returns the current frame.

5.4.3.9 GetDepthStencilFormat()

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

Returns a constant reference to the depth stencil format.

5.4.3.10 GetDevice()

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

Returns a constant reference to the ID3D12Device objcet.

5.4.3.11 GetInstance()

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

Call to make an object of DeviceResources.

Only one instance of DeviceResources can exist in a program.

Parameters

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

5.4.3.12 GetTextResources()

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

Returns a constant reference to the TextResources object.

5.4.3.13 NextFrame()

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

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

5.4.3.14 Present()

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

Swaps the front and back buffers.

5.4.3.15 Resize()

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

Call when the window gets resized.

Call when you initialize your program.

Parameters

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

5.4.3.16 RTBufferTransition()

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

Transistions the render target buffer.

Parameters

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

5.4.3.17 Signal()

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

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

5.4.3.18 UpdateCurrentFrameFenceValue()

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

Updates the current frames fence value.

5.4.3.19 WaitForGPU()

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

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

Signal should have been called before this function is called.

5.4.4 Member Data Documentation

5.4.4.1 NUM_OF_FRAMES

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

The number of frames in the ciruclar array.

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

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

- FADeviceResources.h

5.5 DirectXException Class Reference

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

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

Public Member Functions

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

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

  Returns a message describing the error.

5.5.1 Detailed Description

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

5.5.2 Constructor & Destructor Documentation

5.5.2.1 DirectXException()

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

Constructs a DirectXException object.

Parameters

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

5.5.3 Member Function Documentation

5.5.3.1 ErrorMsg()

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

Returns a message describing the error.

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

- FADirectXException.h

5.6 FARender::DynamicBuffer Class Reference

This class stores data in a Direct3D 12 upload buffer.

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

Public Member Functions

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

  Unmaps the pointer to the dynamic buffer.
- D3D12_GPU_VIRTUAL_ADDRESS GetGPUAddress () const

  Returns the GPU virtual address of the first byte of the dynamic buffer.
- const unsigned int & GetStride () const

  Returns the stride of the dymanic buffer.
- const DXGI_FORMAT & GetFormat () const

  Returns the format of the dymanic buffer.
- void CreateDynamicBuffer (const Microsoft::WRL::ComPtr< ID3D12Device > &device, UINT numOfBytes, UINT stride)

  Creates and maps the dynamic buffer.
- void CreateDynamicBuffer (const Microsoft::WRL::ComPtr< ID3D12Device > &device, UINT numOfBytes, DXGI_FORMAT format)

  Creates and maps the dynamic buffer.

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

  Creates the constant buffer view and stores it in the specified descriptor heap.
- void CreateVertexBufferView (UINT numBytes)

  Creates a vertex buffer view and stores it.
- void CreateIndexBufferView (UINT numBytes)

  Creates a index buffer view and stores it.
- const D3D12_VERTEX_BUFFER_VIEW & GetVertexBufferView ()

  Returns a constant reference to the vertex buffer view.
- const D3D12_INDEX_BUFFER_VIEW & GetIndexBufferView ()

  Returns a constant reference to the vertex buffer view.
- void CopyData (UINT index, const void *data, UINT64 numOfBytes)

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

5.6.1 Detailed Description

This class stores data in a Direct3D 12 upload buffer.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 ~DynamicBuffer()

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

Unmaps the pointer to the dynamic buffer.

5.6.3 Member Function Documentation

5.6.3.1 CopyData()

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

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

Parameters

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

5.6.3.2 CreateConstantBufferView()

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

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

Parameters

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

5.6.3.3 CreateDynamicBuffer() [1/2]

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

Creates and maps the dynamic buffer.

Call if you want to create a dynamic index buffer.

Parameters

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

5.6.3.4 CreateDynamicBuffer() [2/2]

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

Creates and maps the dynamic buffer.

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

Parameters

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

5.6.3.5 CreateIndexBufferView()

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

Creates a index buffer view and stores it.

Parameters

in	*numBytes*	The number of bytes in the dynamic buffer.

5.6.3.6 CreateVertexBufferView()

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

Creates a vertex buffer view and stores it.

Parameters

in	*numBytes*	The number of bytes in the dynamic buffer.

5.6.3.7 GetFormat()

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

Returns the format of the dymanic buffer.

5.6.3.8 GetGPUAddress()

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

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

5.6.3.9 GetIndexBufferView()

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

Returns a constant reference to the vertex buffer view.

5.6.3.10 GetStride()

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

Returns the stride of the dymanic buffer.

5.6.3.11 GetVertexBufferView()

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

Returns a constant reference to the vertex buffer view.

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

- FABuffer.h

5.7 FARender::MultiSampling Class Reference

A wrapper for multisampling resources. Uses DirectD 12 API.

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

Public Member Functions

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

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

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

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

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

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

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

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

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

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

5.7.1 Detailed Description

A wrapper for multisampling resources. Uses DirectD 12 API.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 MultiSampling()

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

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

Throws a runtime_error if they are not supproted.

Parameters

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

5.7.3 Member Function Documentation

5.7.3.1 ClearDepthStencilBuffer()

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

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

Parameters

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

5.7.3.2 ClearRenderTargetBuffer()

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

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

Parameters

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

5.7.3.3 CreateDepthStencilBufferAndView()

```
void FARender::MultiSampling::CreateDepthStencilBufferAndView (
            const Microsoft::WRL::ComPtr< ID3D12Device > & device,
            const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & dsvHeap,
```

```
            unsigned int indexOfWhereToStoreView,
            unsigned int dsvSize,
            unsigned int width,
            unsigned int height )
```

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

Parameters

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

5.7.3.4 CreateRenderTargetBufferAndView()

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

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

Parameters

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

5.7.3.5 GetRenderTargetBuffer()

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

Returns the MSAA render target buffer.

5.7.3.6 ResetBuffers()

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

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

5.7.3.7 Transition()

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

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

Parameters

in	*commandList*	A Direct3D 12 graphics command list.

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

- FAMultiSampling.h

5.8 FARender::RenderScene Class Reference

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

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

Public Member Functions

- **RenderScene** (unsigned int width, unsigned int height, HWND windowHandle, bool isMSAAEnabled=false, bool isTextEnabled=false)
- **RenderScene** (const RenderScene &)=delete
- RenderScene & **operator=** (const RenderScene &)=delete
- **RenderScene** (RenderScene &&)=default
- FACamera::Camera & GetCamera ()

  Returns a reference to the this scene's camera;.
- const FACamera::Camera & GetCamera () const

  Returns a constant reference to the this scene's camera;.
- FARender::Text & GetText (unsigned int textKey)

  Returns a reference to the specified Text object.
- const FARender::Text & GetText (unsigned int textKey) const

  Returns a constant reference to the specified Text object.
- void LoadShader (unsigned int shaderKey, const std::wstring &filename)

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

- void CompileShader (unsigned int shaderKey, const std::wstring &filename, const std::string &entryPoint↩ Name, const std::string &target)

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

  Removes the shader bytecode mapped to the specified shaderKey.
- void CreateStaticBuffer (unsigned int bufferType, unsigned int staticBufferKey, const void ∗data, unsigned numBytes, unsigned int stride=0, DXGI_FORMAT format=DXGI_FORMAT_R32_UINT)

  Creates a static buffer and stores the specified data in the buffer. The user cannot update/changed the data once it is stored in the buffer.
  Make sure to pass in different keys to store the static buffers with to prevent replacing the static buffer at that key with the newly created static buffer.
- void CreateDynamicBuffer (unsigned int bufferType, unsigned int dynamicBufferKey, unsigned numBytes, unsigned int stride=0, DXGI_FORMAT format=DXGI_FORMAT_R32_UINT)

  Creates a dynamic buffer. The user can update the data on a per-frame basis.
  Make sure to pass in different keys to store the dynamic buffers with to prevent replacing the dynamic buffer at that key with the newly created dynamic buffer.
- void CreateInputElementDescription (unsigned int key, const char ∗semanticName, unsigned int semantic↩ Index, DXGI_FORMAT format, unsigned int inputSlot, unsigned int byteOffset, D3D12_INPUT_↩ CLASSIFICATION inputSlotClass=D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA, unsigned int instanceStepRate=0)

  Creates an input element description and stores in an array mapped to the specified key.
- void CreateRootParameter (unsigned int rootParameterKey, unsigned int shaderRegister)

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

  Creates a root signature and maps it to the specified rootSigKey.
- void CreatePSO (unsigned int psoKey, D3D12_FILL_MODE fillMode, BOOL enableMultisample, unsigned int vsKey, unsigned int psKey, unsigned int inputElementDescriptionsKey, unsigned int rootSigKey, const D3↩ D12_PRIMITIVE_TOPOLOGY_TYPE &primitiveType, UINT sampleCount=1)

  Creates a PSO and maps it to the specified psoKey.
- void AddDrawArgument (unsigned int key, const FAShapes::DrawArguments &drawArg)

  Stores the specified DrawArgument structure in an array mapped to the specified key.
- void CreateDrawArgument (unsigned int key, unsigned int indexCount, unsigned int locationOfFirstIndex, int indexOfFirstVertex, int indexOfConstantData)

  Creates a DrawArgument structure with the specified values and stores it in an array mapped to the specified key.
- void RemoveDrawArgument (unsigned int key, unsigned int drawArgIndex)

  Removes the specified DrawArgument structure in the array mapped to the specified key.
- void CreateText (unsigned int textKey, FAMath::Vector4D textLocation, const std::wstring &textString, float textSize, const FAColor::Color textColor)

  Creates a Text object with the specified properties and maps it to the specified textKey.
- void RemoveText (unsigned int textKey)

  Removes the Text object mapped to the specified textKey.
- void SetPSOAndRootSignature (unsigned int psoKey, unsigned int rootSigKey)

  Sets the PSO and its associated root signature to indicate what settings you want to use to render objects. An out_of_range exception is thrown if any of the keys don't have a mapped values.
- void SetStaticBuffer (unsigned int bufferType, unsigned int staticBufferKey)

  Links the static buffer mapped to the static buffer key to the pipeline.
- void SetDynamicBuffer (unsigned int bufferType, unsigned int dynamicBufferKey, unsigned int root↩ ParameterIndex=0)

  Links the dynamic buffer mapped to the dynamic buffer key to the pipeline.
- void BeforeRenderObjects (bool isMSAAEnabled=false)

  Puts all of the commands needed in the command list before drawing the objects of the scene.
- void RenderObjects (unsigned int drawArgsKey, unsigned int objectConstantBufferKey, unsigned int root↩ ParamterIndex, D3D_PRIMITIVE_TOPOLOGY primitive)

Renders the objects in the array mapped to the specified drawArgsKey.

- void AfterRenderObjects (bool isMSAAEnabled=false, bool isTextEnabled=false)

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

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

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

  .
- void AfterRenderText ()

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

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

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

  Resizes the window-dependent resources when the window gets resized.
- void CopyDataIntoDynamicBuffer (unsigned int dynamicBufferKey, unsigned int index, const void *data, UINT64 numOfBytes)

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

5.8.1 Detailed Description

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

5.8.2 Member Function Documentation

5.8.2.1 AddDrawArgument()

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

Stores the specified *DrawArgument* structure in an array mapped to the specified *key*.

5.8.2.2 AfterRender()

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

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

Call after rendering all your objects and text.

5.8.2.3 AfterRenderObjects()

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

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

Parameters

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

5.8.2.4 AfterRenderText()

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

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

Call after calling all the RenderText functions.

5.8.2.5 BeforeRenderObjects()

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

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

Call before calling the first RenderObjects function.

Parameters

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

5.8.2.6 BeforeRenderText()

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

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

5.8.2.7 CompileShader()

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

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

Parameters

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

5.8.2.8 CopyDataIntoDynamicBuffer()

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

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

Parameters

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

5.8.2.9 CreateDrawArgument()

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

Creates a DrawArgument structure with the specified values and stores it in an array mapped to the specified *key*.

Parameters

in	*key*	The key to a mappped array to store the created DrawArguments structure in.
in	*indexCount*	The number of indicies used to connect the vertices of an object in an index buffer.
in	*locationOfFirstIndex*	The location of the first index of the indices used to connect the vertices of an object in an index buffer.
in	*indexOfFirstVertex*	The index of the first vertex of the vertices of an object in a vertex buffer.
in	*indexOfConstantData*	The index of the objects constant data in a constant buffer.

5.8.2.10 CreateDynamicBuffer()

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

Creates a dynamic buffer. The user can update the data on a per-frame basis.
Make sure to pass in different keys to store the dynamic buffers with to prevent replacing the dynamic buffer at that key with the newly created dynamic buffer.

Parameters

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

5.8.2.11 CreateInputElementDescription()

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

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

Parameters

in	*key*	The key to a mapped array to store the created input element description.
in	*semanticName*	The name of the application variable linked to a shader variable.
in	*semanticIndex*	The index to attach to the semanticName.
in	*format*	The data type of input element being described.
in	*inputSlot*	The input slot the input element will come from.

Parameters

in	*byteOffset*	The offset in bytes to get to the input element being described.
	[in,optional]	inputSlotClass The data class for an input slot. Used for instancing.
	[in,optional]	instanceStepRate The number of instances to render. Used for instancing.

5.8.2.12 CreatePSO()

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

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

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

Parameters

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

5.8.2.13 CreateRootParameter()

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

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

Parameters

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

5.8.2.14 CreateRootSignature()

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

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

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

Parameters

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

5.8.2.15 CreateStaticBuffer()

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

Creates a static buffer and stores the specified *data* in the buffer. The user cannot update/changed the data once it is stored in the buffer.
Make sure to pass in different keys to store the static buffers with to prevent replacing the static buffer at that key with the newly created static buffer.

Parameters

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

5.8.2.16 CreateText()

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

Creates a Text object with the specified properties and maps it to the specified *textKey*.

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

Parameters

in	*textKey*	The key to a mapped Text object.
in	*textLocation*	The location of where to render the text on the window.
in	*textString*	The string to render.
in	*textSize*	How big the text is.
in	*textColor*	The color of the text.

5.8.2.17 ExecuteAndFlush()

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

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

5.8.2.18 GetCamera() [1/2]

```
FACamera::Camera & FARender::RenderScene::GetCamera ( )
```

Returns a reference to the this scene's camera;.

5.8.2.19 GetCamera() [2/2]

```
const FACamera::Camera & FARender::RenderScene::GetCamera ( ) const
```

Returns a constant reference to the this scene's camera;.

5.8.2.20 GetText() [1/2]

```
FARender::Text & FARender::RenderScene::GetText (
            unsigned int textKey )
```

Returns a reference to the specified Text object.

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

5.8.2.21 GetText() [2/2]

```
const FARender::Text & FARender::RenderScene::GetText (
            unsigned int textKey ) const
```

Returns a constant reference to the specified Text object.

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

5.8.2.22 LoadShader()

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

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

Parameters

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

5.8.2.23 RemoveDrawArgument()

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

Removes the specified DrawArgument structure in the array mapped to the specified key.

If the *key* to the specified array of DrawArgument structure or if the index to the specifed DrawArguments structure in the array is out of bounds an out_of_range exception is thrown.

Parameters

in	*key*	The key to a mapped array of where the DrawArguments structure is stored.
in	*drawArgIndex*	The index of where the DrawArguments structure is stored.

5.8.2.24 RemoveShader()

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

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

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

5.8.2.25 RemoveText()

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

Removes the Text object mapped to the specified *textKey*.

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

Parameters

in	*textKey*	The key to a mapped Text object.

5.8.2.26 RenderObjects()

```
void FARender::RenderScene::RenderObjects (
            unsigned int drawArgsKey,
            unsigned int objectConstantBufferKey,
            unsigned int rootParamterIndex,
            D3D_PRIMITIVE_TOPOLOGY primitive )
```

Renders the objects in the array mapped to the specified *drawArgsKey*.

Call in between a BeforeRenderObjects function and a AfterRenderObjects function.

Ex.
BeforeRenderObjects()
RenderObjects()
RenderObjects()
AfterRenderObjects()

Throws an out_of_range exception if any of the keys don't have mapped values.

Parameters

in	*objectConstantBufferKey*	The key mapped to a constant buffer that has the constant data for the specified objects.
in	*rootParameterIndex*	The index of the root parameter in the root signature that has the register the constant data in the dynamic constant buffer will be stored in.
in	*primitive*	The primitve used to render the objects.

5.8.2.27 RenderText()

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

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

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

Parameters

in	*textKey*	The key mapped to a Text object.

5.8.2.28 Resize()

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

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

Parameters

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

5.8.2.29 SetDynamicBuffer()

```
void FARender::RenderScene::SetDynamicBuffer (
            unsigned int bufferType,
```

```
            unsigned int dynamicBufferKey,
            unsigned int rootParameterIndex = 0 )
```

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

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

Parameters

in		

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

Parameters

in	*dynamicBufferKey*	The key mapped to a dynamic buffer.
	[in,optional]	rootParameterIndex The index of the root parameter in the root signature that has the register the constant data in the dynamic constant buffer will be stored in. Used if the dynamic buffer is a constant buffer.

5.8.2.30 SetPSOAndRootSignature()

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

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

Parameters

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

5.8.2.31 SetStaticBuffer()

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

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

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

Parameters

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

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

- FARenderScene.h

5.9 FARender::RenderTargetBuffer Class Reference

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

`#include "FABuffer.h"`

Public Member Functions

- RenderTargetBuffer (DXGI_FORMAT format=DXGI_FORMAT_R8G8B8A8_UNORM)

  Default Constructor.
- DXGI_FORMAT GetRenderTargetFormat () const

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

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

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

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

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

  Clears the render target buffer with the specified clear value.

5.9.1 Detailed Description

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

5.9.2 Constructor & Destructor Documentation

5.9.2.1 RenderTargetBuffer()

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

Default Constructor.

Parameters

in	*format*	The format of the render target buffer.

5.9.3 Member Function Documentation

5.9.3.1 ClearRenderTargetBuffer()

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

Clears the render target buffer with the specified clear value.

Parameters

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

5.9.3.2 CreateRenderTargetBufferAndView()

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

Creates the render target buffer and view.

Parameters

in	*device*	A Direct3D 12 device.
in	*rtvHeap*	A descriptor heap for storing render target descriptors.
in	*indexOfWhereToStoreView*	The index of where to store the created descriptor in the descriptor heap.

Parameters

in	*rtvSize*	The size of a render target descriptor.
in	*width*	The width of the render target buffer.
in	*height*	The height of the render target buffer.
in	*sampleCount*	The sample count of the render target buffer.

5.9.3.3 GetRenderTargetBuffer() [1/2]

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

Returns a reference to the render target buffer.

5.9.3.4 GetRenderTargetBuffer() [2/2]

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

Returns a constant reference to the render target buffer.

5.9.3.5 GetRenderTargetFormat()

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

Returns the format of the render target buffer.

5.9.3.6 ResetBuffer()

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

Resets the render target buffer.

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

- FABuffer.h

5.10 FARender::StaticBuffer Class Reference

This class stores data in a Direct3D 12 default buffer.

`#include "FABuffer.h"`

Public Member Functions

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

  Creates the static buffer and stores all of the specified data.
- void CreateVertexBufferView (UINT numBytes, UINT stride)

  Creates a vertex buffer view and stores it.
- void CreateIndexBufferView (UINT numBytes, DXGI_FORMAT format)

  Creates a index buffer view and stores it.
- const D3D12_VERTEX_BUFFER_VIEW & GetVertexBufferView ()

  Returns a constant reference to the vertex buffer view.
- const D3D12_INDEX_BUFFER_VIEW & GetIndexBufferView ()

  Returns a constant reference to the vertex buffer view.

5.10.1 Detailed Description

This class stores data in a Direct3D 12 default buffer.

5.10.2 Member Function Documentation

5.10.2.1 CreateIndexBufferView()

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

Creates a index buffer view and stores it.

Parameters

in	*numBytes*	The number of bytes in the static buffer.
in	*format*	The number of bytes to get from one element to another in the static buffer.

5.10.2.2 CreateStaticBuffer()

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

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

Parameters

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

5.10.2.3 CreateVertexBufferView()

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

Creates a vertex buffer view and stores it.

Parameters

in	*numBytes*	The number of bytes in the static buffer.
in	*stride*	The number of bytes to get from one element to another in the static buffer.

5.10.2.4 GetIndexBufferView()

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

Returns a constant reference to the vertex buffer view.

5.10.2.5 GetVertexBufferView()

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

Returns a constant reference to the vertex buffer view.

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

- FABuffer.h

5.11 FARender::SwapChain Class Reference

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

`#include "FASwapChain.h"`

Public Member Functions

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  Swaps the front and back buffers.

5.11.1 Detailed Description

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

5.11.2 Constructor & Destructor Documentation

5.11.2.1 SwapChain()

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

Creates a swap chain.

Parameters

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

5.11.3 Member Function Documentation

5.11.3.1 ClearCurrentBackBuffer()

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

Clears the current render target buffer.

Parameters

in	*commadList*	A Direct3D 12 graphics command list.

Parameters

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

5.11.3.2 ClearDepthStencilBuffer()

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

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

Parameters

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

5.11.3.3 CreateDepthStencilBufferAndView()

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

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

Parameters

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

5.11.3.4 CreateRenderTargetBuffersAndViews()

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

Creates the render target buffers and views to them.

Parameters

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

5.11.3.5 GetBackBufferFormat()

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

Returns the format of the swap chain.

5.11.3.6 GetCurrentBackBuffer()

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

Returns a constant reference to the current render target buffer.

5.11.3.7 GetCurrentBackBufferIndex()

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

Returns the current back buffer index.

5.11.3.8 GetDepthStencilFormat()

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

Returns the format of the depth stencil buffer.

5.11.3.9 GetNumRenderTargetBuffers()

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

Returns the number of swap chain buffers.

5.11.3.10 GetRenderTargetBuffers()

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

Returns a constant pointer to the render target buffers.

5.11.3.11 Present()

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

Swaps the front and back buffers.

5.11.3.12 ResetBuffers()

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

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

5.11.3.13 ResizeSwapChain()

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

Resizes the swap chain.

Parameters

in	*width*	The width to resize the render target buffers to.
in	*height*	The height to resize the render target buffers to.

5.11.3.14 Transition()

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

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

Parameters

in	*commandList*	A Direct3D 12 graphics command list.

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

- FASwapChain.h

5.12 FARender::Text Class Reference

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

`#include "FAText.h"`

Public Member Functions

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

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

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

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

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

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

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

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

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

  Changes the text location to the specified textLocation.

5.12.1 Detailed Description

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

5.12.2 Constructor & Destructor Documentation

5.12.2.1 Text()

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

Constructs a Text object.

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

Parameters

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

5.12.3 Member Function Documentation

5.12.3.1 GetTextColor()

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

Returns a constant reference to the text color.

5.12.3.2 GetTextLocation()

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

Returns a constant reference to the text location.

5.12.3.3 GetTextSize()

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

Returns the text size.

5.12.3.4 GetTextString()

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

Returns a constant reference to the text string.

5.12.3.5 SetTextColor()

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

Changes the text color to the specified *textColor*.

5.12.3.6 SetTextLocation()

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

Changes the text location to the specified *textLocation*.

5.12.3.7 SetTextSize()

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

Changes the text size to the specified *textSize*.

5.12.3.8 SetTextString()

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

Changes the text string to the specified *textString*.

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

- FAText.h

5.13 FARender::TextResources Class Reference

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

`#include "FATextResources.h"`

Public Member Functions

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

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

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

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

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

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

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

  Executes text commands.

5.13.1 Detailed Description

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

5.13.2 Constructor & Destructor Documentation

5.13.2.1 TextResources()

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

Initializes the text resources.

Parameters

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

5.13.3 Member Function Documentation

5.13.3.1 AfterRenderText()

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

Executes text commands.

Parameters

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

5.13.3.2 BeforeRenderText()

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

Prepares to render text.

Parameters

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

5.13.3.3 GetDirect2DDeviceContext()

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

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

5.13.3.4 GetDirectWriteFactory()

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

Returns a constant reference to the direct direct write factory.

5.13.3.5 ResetBuffers()

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

Resets the text buffers.

5.13.3.6 ResizeBuffers()

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

Resizes the buffers.

Parameters

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

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

- FATextResources.h

5.14 FATime::Time Class Reference

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

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

Public Member Functions

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

5.14.1 Detailed Description

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

5.14.2 Constructor & Destructor Documentation

5.14.2.1 Time()

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

Gets and stores the seconds per count.

5.14.3 Member Function Documentation

5.14.3.1 DeltaTime()

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

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

5.14.3.2 Reset()

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

Resets all time variables.

5.14.3.3 Start()

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

Starts the timer.

5.14.3.4 Stop()

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

Stops the timer.

5.14.3.5 Tick()

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

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

5.14.3.6 TotalTime()

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

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

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

- FATime.h

5.15 FAWindow::Window Class Reference

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

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

Public Member Functions

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

  Creates and displays a window.
- 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.
- HWND GetWindowHandle () const

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

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

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

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

  Sets the height of the window o the specified height.

5.15.1 Detailed Description

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

5.15.2 Constructor & Destructor Documentation

5.15.2.1 Window() [1/2]

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

Creates and displays a window.

Parameters

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

5.15.2.2 Window() [2/2]

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

Creates and displays a window.

Parameters

in	*hInstance*	The handle to a module used to identify the executable.
in	*windowClass*	The window class for this window.

Parameters

in	*windowName*	The name of the window.
in	*width*	The width of the window.
in	*height*	The height of the window.
	[in,optional]	additionalData A pointer to data to access in the window procedure.

5.15.3 Member Function Documentation

5.15.3.1 GetHeight()

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

Returns the height of the window.

5.15.3.2 GetWidth()

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

Returns the width of the window.

5.15.3.3 GetWindowHandle()

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

Returns the window handle.

5.15.3.4 SetHeight()

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

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

5.15.3.5 SetWidth()

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

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

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

- FAWindow.h

Chapter 6

File Documentation

6.1 Direct3DLink.h

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

6.2 FABuffer.h File Reference

File has classes RenderTargetBuffer, DepthStencilBuffer, StaticBuffer and DynamicBuffer under namespace FARender.

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

Classes

- class FARender::RenderTargetBuffer

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

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

  This class stores data in a Direct3D 12 default buffer.
- class FARender::DynamicBuffer

  This class stores data in a Direct3D 12 upload buffer.

Namespaces

- namespace FARender

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

6.2.1 Detailed Description

File has classes RenderTargetBuffer, DepthStencilBuffer, StaticBuffer and DynamicBuffer under namespace FARender.

6.3 FABuffer.h

Go to the documentation of this file.

```
1 #pragma once
2
7 #include <wrl.h>
8 #include <d3d12.h>
9
13 namespace FARender
14 {
18     class RenderTargetBuffer
19     {
20     public:
25         RenderTargetBuffer(DXGI_FORMAT format = DXGI_FORMAT_R8G8B8A8_UNORM);
26
29         DXGI_FORMAT GetRenderTargetFormat() const;
30
33         Microsoft::WRL::ComPtr<ID3D12Resource>& GetRenderTargetBuffer();
34
37         const Microsoft::WRL::ComPtr<ID3D12Resource>& GetRenderTargetBuffer() const;
38
49         void CreateRenderTargetBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
50             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int
    indexOfWhereToStoreView, unsigned int rtvSize,
51             unsigned int width, unsigned int height, unsigned int sampleCount = 1);
52
55         void ResetBuffer();
56
67         void ClearRenderTargetBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
    commandList,
68             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int indexOfView,
    unsigned int rtvSize,
69             const float* clearValue);
70
71     private:
72         Microsoft::WRL::ComPtr<ID3D12Resource> mRenderTargetBuffer;
73         DXGI_FORMAT mRenderTargetFormat;
74
75     };
76
80     class DepthStencilBuffer
81     {
82     public:
83
88         DepthStencilBuffer(DXGI_FORMAT format = DXGI_FORMAT_D24_UNORM_S8_UINT);
89
92         DXGI_FORMAT GetDepthStencilFormat() const;
93
104          void CreateDepthStencilBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
105              const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int
    indexOfWhereToStoreView, unsigned int dsvSize,
106              unsigned int width, unsigned int height, unsigned int sampleCount = 1);
107
110          void ResetBuffer();
111
122          void ClearDepthStencilBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
    commandList,
123              const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int indexOfView,
    unsigned int dsvSize,
124              float clearValue);
125
126     private:
127         Microsoft::WRL::ComPtr<ID3D12Resource> mDepthStencilBuffer;
128         DXGI_FORMAT mDepthStencilFormat;
129     };
130
131
135     class StaticBuffer
136     {
137     public:
138         StaticBuffer() = default;
139         StaticBuffer(const StaticBuffer&) = delete;
140         StaticBuffer& operator=(const StaticBuffer&) = delete;
141
```

```
142         StaticBuffer(StaticBuffer&&) = default;
143 
151         void CreateStaticBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
152             const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const void* data, UINT
    numBytes);
153 
159         void CreateVertexBufferView(UINT numBytes, UINT stride);
160 
166         void CreateIndexBufferView(UINT numBytes, DXGI_FORMAT format);
167 
170         const D3D12_VERTEX_BUFFER_VIEW& GetVertexBufferView();
171 
174         const D3D12_INDEX_BUFFER_VIEW& GetIndexBufferView();
175 
176     private:
177         Microsoft::WRL::ComPtr<ID3D12Resource> mStaticDefaultBuffer;
178         Microsoft::WRL::ComPtr<ID3D12Resource> mStaticUploadBuffer;
179 
180         union
181         {
182             D3D12_VERTEX_BUFFER_VIEW mVertexBufferView{};
183             D3D12_INDEX_BUFFER_VIEW mIndexBufferView;
184         };
185     };
186 
190     class DynamicBuffer
191     {
192     public:
193         DynamicBuffer() = default;
194 
195         DynamicBuffer(const DynamicBuffer&) = delete;
196         DynamicBuffer& operator=(const DynamicBuffer&) = delete;
197 
198         DynamicBuffer(DynamicBuffer&&) = default;
199 
202         ~DynamicBuffer();
203 
206         D3D12_GPU_VIRTUAL_ADDRESS GetGPUAddress() const;
207 
210         const unsigned int& GetStride() const;
211 
214         const DXGI_FORMAT& GetFormat() const;
215 
224         void CreateDynamicBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device, UINT numOfBytes,
    UINT stride);
225 
234         void CreateDynamicBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device, UINT numOfBytes,
    DXGI_FORMAT format);
235 
244         void CreateConstantBufferView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
245             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& cbvHeap, UINT cbvSize, UINT
    cbvHeapIndex,
246             UINT cBufferIndex);
247 
252         void CreateVertexBufferView(UINT numBytes);
253 
258         void CreateIndexBufferView(UINT numBytes);
259 
262         const D3D12_VERTEX_BUFFER_VIEW& GetVertexBufferView();
263 
266         const D3D12_INDEX_BUFFER_VIEW& GetIndexBufferView();
267 
274         void CopyData(UINT index, const void* data, UINT64 numOfBytes);
275 
276     private:
277         Microsoft::WRL::ComPtr<ID3D12Resource> mDynamicBuffer;
278         BYTE* mMappedData{ nullptr };
279 
280         union
281         {
282             UINT mStride;
283             DXGI_FORMAT mFormat;
284         };
285 
286         union
287         {
288             D3D12_VERTEX_BUFFER_VIEW mVertexBufferView{};
289             D3D12_INDEX_BUFFER_VIEW mIndexBufferView;
290         };
291     };
292 }
```

6.4 FACamera.h File Reference

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

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

Classes

- class FACamera::Camera

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

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

Namespaces

- namespace FACamera

  Has Camera class.

Typedefs

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

6.4.1 Detailed Description

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

6.4.2 Typedef Documentation

6.4.2.1 vec2

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

FACAMERA_H FILE

6.5 FACamera.h

Go to the documentation of this file.

```
1 #pragma once
2
12 #include "FAMathEngine.h"
13 #include <Windows.h>
14
15 typedef FAMath::Vector2D vec2;
16 typedef FAMath::Vector3D vec3;
17 typedef FAMath::Vector4D vec4;
18 typedef FAMath::Matrix4x4 mat4;
19
23 namespace FACamera
24 {
30     class Camera
31     {
32     public:
46         Camera(vec3 cameraPosition = vec3(0.0f, 0.0f, 0.0f),
47             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),
48             float znear = 1.0f, float zfar = 100.f, float aspectRatio = 1.0f, float vFov = 45.0f,
49             float cameraVelocity = 10.0f, float angularVelocity = 0.25f);
50
53         const vec3& GetCameraPosition() const;
54
57         const vec3& GetX() const;
58
61         const vec3& GetY() const;
62
65         const vec3& GetZ() const;
66
69         const mat4& GetViewMatrix() const;
70
73         float GetCameraVelocity() const;
74
77         float GetAngularVelocity() const;
78
85         void LookAt(vec3 cameraPosition, vec3 target, vec3 up);
86
89         float GetZNear() const;
90
93         float GetZFar() const;
94
97         float GetVerticalFov() const;
98
101         float GetAspectRatio() const;
102
105         void SetCameraPosition(const vec3& position);
106
109         void SetX(const vec3& x);
110
113         void SetY(const vec3& y);
114
117         void SetZ(const vec3& z);
118
121         void SetCameraVelocity(float velocity);
122
125         void SetAngularVelocity(float velcoity);
126
129         void SetZNear(float znear);
130
133         void SetZFar(float zfar);
134
137         void SetVerticalFov(float fov);
138
141         void SetAspectRatio(float ar);
142
145         const mat4& GetPerspectiveProjectionMatrix() const;
146
149         const mat4& GetViewPerspectiveProjectionMatrix() const;
150
153         void UpdateViewMatrix();
154
157         void UpdatePerspectiveProjectionMatrix();
158
163         void UpdateViewPerspectiveProjectionMatrix();
164
167         void Left(float dt);
168
173         void Right(float dt);
174
179         void Foward(float dt);
180
185         void Backward(float dt);
```

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

6.6 FAColor.h File Reference

File has class Color under namespace FAColor.

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

Classes

- class FAColor::Color

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

Namespaces

- namespace FAColor

  Has the Color class.

Functions

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

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

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

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

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

  Returns the result of c1 ∗ c2.

6.6.1 Detailed Description

File has class Color under namespace FAColor.

6.7 FAColor.h

Go to the documentation of this file.

```
1 #pragma once
2
3 #include "FAMathEngine.h"
4
12 namespace FAColor
13 {
19     class Color
20     {
21     public:
22
25         Color(float r = 0.0f, float g = 0.0f, float b = 0.0f, float a = 1.0f);
26
29         Color(const FAMath::Vector4D& color);
30
33         const FAMath::Vector4D& GetColor() const;
34
37         float GetRed() const;
38
41         float GetGreen() const;
42
45         float GetBlue() const;
46
49         float GetAlpha() const;
50
53         void SetColor(const FAMath::Vector4D& color);
54
57         void SetRed(float r);
58
61         void SetGreen(float g);
62
65         void SetBlue(float b);
66
69         void SetAlpha(float a);
70
75         Color& operator+=(const Color& c);
76
81         Color& operator-=(const Color& c);
82
88         Color& operator*=(float k);
89
95         Color& operator*=(const Color& c);
96
97     private:
98         FAMath::Vector4D mColor;
99     };
100
105     Color operator+(const Color& c1, const Color& c2);
106
111     Color operator-(const Color& c1, const Color& c2);
```

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

6.8 FADeviceResources.h File Reference

File has class DeviceResources under namespace FARender.

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

Classes

- class FARender::DeviceResources

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

Namespaces

- namespace FARender

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

6.8.1 Detailed Description

File has class DeviceResources under namespace FARender.

6.9 FADeviceResources.h

Go to the documentation of this file.

```
1 #pragma once
2
7 #include <wrl.h>
8 #include <d3d12.h>
9 #include <dxgi1_4.h>
10 #include "FASwapChain.h"
11 #include "FAMultiSampling.h"
12 #include "FATextResources.h"
13
14 namespace FARender
15 {
19     class DeviceResources
20     {
21     public:
22
27         static const unsigned int NUM_OF_FRAMES{ 3 };
28
40         static DeviceResources& GetInstance(unsigned int width, unsigned int height, HWND windowHandle,
      bool isMSAAEnabled, bool isTextEnabled);
41
```

```
42          DeviceResources(const DeviceResources&) = delete;
43          DeviceResources& operator=(const DeviceResources&) = delete;
44
47          ~DeviceResources();
48
51          const Microsoft::WRL::ComPtr<ID3D12Device>& GetDevice() const;
52
55          const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& GetCommandList() const;
56
59          DXGI_FORMAT GetBackBufferFormat() const;
60
63          DXGI_FORMAT GetDepthStencilFormat() const;
64
67          unsigned int GetCBVSize() const;
68
71          unsigned int GetCurrentFrame() const;
72
75          const TextResources& GetTextResources() const;
76
79          void UpdateCurrentFrameFenceValue();
80
86          void FlushCommandQueue();
87
92          void WaitForGPU() const;
93
96          void Signal();
97
108          void Resize(int width, int height, const HWND& handle, bool isMSAAEnabled, bool isTextEnabled);
109
115          void RTBufferTransition(bool isMSAAEnabled, bool isTextEnabled);
116
119          void BeforeTextDraw();
120
123          void AfterTextDraw();
124
127          void Execute() const;
128
131          void Present();
132
133          /*@brief Calls the necessary functions to let the user draw their objects.
134 *
135 *  @param[in] isMSAAEnabled Pass in true if MSAA enabled, false otherwise.
136 */
137          void Draw(bool isMSAAEnabled);
138
141          void NextFrame();
142
143      private:
144
157          DeviceResources(unsigned int width, unsigned int height, HWND windowHandle,
158              bool isMSAAEnabled, bool isTextEnabled);
159
160          unsigned int mCurrentFrameIndex;
161
162          Microsoft::WRL::ComPtr<ID3D12Device> mDirect3DDevice;
163
164          Microsoft::WRL::ComPtr<IDXGIFactory4> mDXGIFactory;
165
166          Microsoft::WRL::ComPtr<ID3D12Fence> mFence;
167          UINT64 mFenceValue;
168          UINT64 mCurrentFrameFenceValue[NUM_OF_FRAMES];
169
170          Microsoft::WRL::ComPtr<ID3D12CommandQueue> mCommandQueue;
171          Microsoft::WRL::ComPtr<ID3D12CommandAllocator> mCommandAllocators[NUM_OF_FRAMES];
172          Microsoft::WRL::ComPtr<ID3D12CommandAllocator> mDirectCommandAllocator;
173          Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList> mCommandList;
174
175          UINT mRTVSize;
176          UINT mDSVSize;
177          UINT mCBVSize;
178
179          Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mRTVHeap;
180          Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mDSVHeap;
181
182          SwapChain mSwapChain;
183
184          MultiSampling mMultiSampling;
185
186          D3D12_VIEWPORT mViewport{};
187          D3D12_RECT mScissor{};
188
189          TextResources mTextResources;
190
191          //Call all of these functions to initialize Direct3D
192          void mEnableDebugLayer();
193          void mCreateDirect3DDevice();
194          void mCreateDXGIFactory();
```

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

6.10 FADirectXException.h

```
1 #pragma once
2
7 #include <wrl.h>
8 #include <dxgidebug.h>
9 #include <comdef.h>
10 #include <string>
11 #include <sstream>
12 #include <vector>
13
18 inline std::wstring AnsiToWString(const std::string& str)
19 {
20     WCHAR buffer[1024];
21     MultiByteToWideChar(CP_ACP, 0, str.c_str(), -1, buffer, 1024);
22     return std::wstring(buffer);
23 }
24
28 class DirectXException
29 {
30 public:
31
39     DirectXException(HRESULT hr, const std::wstring& functionName, const std::wstring& fileName, int
      lineNumber);
40
43     std::wstring ErrorMsg() const;
44
45 private:
46     HRESULT errorCode;
47     std::wstring functionName;
48     std::wstring fileName;
49     int lineNumber;
50     Microsoft::WRL::ComPtr<IDXGIInfoQueue> mInfoQueue;
51 };
52
55 #ifndef ThrowIfFailed
56 #define ThrowIfFailed(x)                                                    \
57 {                                                                           \
58 HRESULT hr = (x);                                                   \
59 std::wstring filename(AnsiToWString(__FILE__));                     \
60 if (FAILED(hr)) { throw DirectXException(hr, L#x, filename, __LINE__); }    \
61 }
62 #endif
```

6.11 FAMultiSampling.h

```
1 #pragma once
2
3 #include <wrl.h>
4 #include "d3dx12.h"
5 #include "FABuffer.h"
6
7 namespace FARender
8 {
12     class MultiSampling
13     {
14     public:
15
16         MultiSampling() = default;
17
27         MultiSampling(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
28             DXGI_FORMAT rtFormat, DXGI_FORMAT dsFormat, unsigned int sampleCount);
29
32         const Microsoft::WRL::ComPtr<ID3D12Resource>& GetRenderTargetBuffer();
33
34         DXGI_FORMAT GetRenderTargetFormat();
35
36         DXGI_FORMAT GetDepthStencilFormat();
37
40         void ResetBuffers();
41
```

```
51          void CreateRenderTargetBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
52              const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int
    indexOfWhereToStoreView, unsigned int rtvSize,
53              unsigned int width, unsigned int height);
54
64          void CreateDepthStencilBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
65              const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int
    indexOfWhereToStoreView, unsigned int dsvSize,
66              unsigned int width, unsigned int height);
67
72          void Transition(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList,
73              D3D12_RESOURCE_STATES before, D3D12_RESOURCE_STATES after);
74
83          void ClearRenderTargetBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
    commandList,
84              const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int indexOfView,
    unsigned int rtvSize,
85              const float* clearValue);
86
95          void ClearDepthStencilBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
    commandList,
96              const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int indexOfView,
    unsigned int dsvSize,
97              float clearValue);
98
99      private:
100         RenderTargetBuffer mMSAARenderTargetBuffer;
101         DepthStencilBuffer mMSAADepthStencilBuffer;
102         unsigned int mSampleCount;
103     };
104 }
```

6.12 FARenderScene.h File Reference

File has class RenderScene under namespace FARender.

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

Classes

- class FARender::RenderScene

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

Namespaces

- namespace FARender

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

6.12.1 Detailed Description

File has class RenderScene under namespace FARender.

6.13 FARenderScene.h

Go to the documentation of this file.

```
1 #pragma once
2
7 #include <d3dcompiler.h>
8 #include <unordered_map>
9 #include "FADeviceResources.h"
10 #include "FABuffer.h"
11 #include "FACamera.h"
12 #include "FAText.h"
13 #include "FAShapesUtility.h"
14
15 namespace FARender
16 {
20     class RenderScene
21     {
22     public:
23
24         /*@brief Initializes all necessary resources.
25 *
26 * Sets the cameras aspect ratio the specified \a width / \a height.
27 *
28 * @param[in] width The width of a window.
29 * @param[in] height The height of a window.
30 * @param[in] windowHandle A handle to a window.
31 * @param[in, optional] isMSAAEnabled Pass in true if you want to have MSAA enabled for the initial frame,
    false otherwise.
32 * @param[in, optional] isTextEnabled Pass in true if you want to have text enabled for the initial frame,
    false otherwise.
33 */
34         RenderScene(unsigned int width, unsigned int height, HWND windowHandle, bool isMSAAEnabled =
    false, bool isTextEnabled = false);
35
36         RenderScene(const RenderScene&) = delete;
37         RenderScene& operator=(const RenderScene&) = delete;
38
39         RenderScene(RenderScene&&) = default;
40
43         FACamera::Camera& GetCamera();
44
47         const FACamera::Camera& GetCamera() const;
48
53         FARender::Text& GetText(unsigned int textKey);
54
59         const FARender::Text& GetText(unsigned int textKey) const;
60
66         void LoadShader(unsigned int shaderKey, const std::wstring& filename);
67
75         void CompileShader(unsigned int shaderKey, const std::wstring& filename,
76             const std::string& entryPointName, const std::string& target);
77
82         void RemoveShader(unsigned int shaderKey);
83
98         void CreateStaticBuffer(unsigned int bufferType, unsigned int staticBufferKey,
99             const void* data, unsigned numBytes, unsigned int stride = 0, DXGI_FORMAT format =
    DXGI_FORMAT_R32_UINT);
100
116        void CreateDynamicBuffer(unsigned int bufferType, unsigned int dynamicBufferKey,
117            unsigned numBytes, unsigned int stride = 0, DXGI_FORMAT format = DXGI_FORMAT_R32_UINT);
118
130        void CreateInputElementDescription(unsigned int key, const char* semanticName, unsigned int
    semanticIndex,
131            DXGI_FORMAT format, unsigned int inputSlot, unsigned int byteOffset,
132            D3D12_INPUT_CLASSIFICATION inputSlotClass = D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA,
133            unsigned int instanceStepRate = 0);
134
140        void CreateRootParameter(unsigned int rootParameterKey, unsigned int shaderRegister);
141
149        void CreateRootSignature(unsigned int rootSigKey, unsigned int rootParametersKey);
150
172        void CreatePSO(unsigned int psoKey, D3D12_FILL_MODE fillMode, BOOL enableMultisample,
173            unsigned int vsKey, unsigned int psKey, unsigned int inputElementDescriptionsKey,
174            unsigned int rootSigKey,
175            const D3D12_PRIMITIVE_TOPOLOGY_TYPE& primitiveType, UINT sampleCount = 1);
176
179        void AddDrawArgument(unsigned int key, const FAShapes::DrawArguments& drawArg);
180
191        void CreateDrawArgument(unsigned int key,
192            unsigned int indexCount, unsigned int locationOfFirstIndex, int indexOfFirstVertex, int
    indexOfConstantData);
193
202        void RemoveDrawArgument(unsigned int key, unsigned int drawArgIndex);
203
215        void CreateText(unsigned int textKey, FAMath::Vector4D textLocation, const std::wstring&
    textString,
```

```
216             float textSize, const FAColor::Color textColor);
217
224         void RemoveText(unsigned int textKey);
225
232         void SetPSOAndRootSignature(unsigned int psoKey, unsigned int rootSigKey);
233
244         void SetStaticBuffer(unsigned int bufferType, unsigned int staticBufferKey);
245
261         void SetDynamicBuffer(unsigned int bufferType, unsigned int dynamicBufferKey, unsigned int
    rootParameterIndex = 0);
262
269         void BeforeRenderObjects(bool isMSAAEnabled = false);
270
290         void RenderObjects(unsigned int drawArgsKey, unsigned int objectConstantBufferKey, unsigned int
    rootParamterIndex,
291             D3D_PRIMITIVE_TOPOLOGY primitive);
292
298         void AfterRenderObjects(bool isMSAAEnabled = false, bool isTextEnabled = false);
299
303         void BeforeRenderText();
304
318         void RenderText(unsigned int textKey);
319
324         void AfterRenderText();
325
330         void AfterRender();
331
334         void ExecuteAndFlush();
335
344         void Resize(unsigned int width, unsigned int height, HWND windowHandle, bool isMSAAEnabled =
    false, bool isTextEnabled = false);
345
353         void CopyDataIntoDynamicBuffer(unsigned int dynamicBufferKey, unsigned int index, const void*
    data, UINT64 numOfBytes);
354
355     private:
356         //The device resources object that all RenderScene objects share.
357         DeviceResources& mDeviceResources;
358
359         //Stores all of the shaders for this scene.
360         std::unordered_map<unsigned int, Microsoft::WRL::ComPtr<ID3DBlob» mShaders;
361
362         //Stores input element descriptions for a set of shaders.
363         std::unordered_map<unsigned int, std::vector<D3D12_INPUT_ELEMENT_DESC»
    mInputElementDescriptions;
364
365         //Stores root parameters for root signatures.
366         std::unordered_map<unsigned int, std::vector<D3D12_ROOT_PARAMETER» mRootParameters;
367
368         //The root signatures for the scene.
369         //Describes all of the constant data that is expected in a set of shaders.
370         //Microsoft::WRL::ComPtr<ID3D12RootSignature> mRootSignature;
371         std::unordered_map<unsigned int, Microsoft::WRL::ComPtr<ID3D12RootSignature» mRootSignatures;
372
373         //Stores pipeline state objects.
374         std::unordered_map<unsigned int, Microsoft::WRL::ComPtr<ID3D12PipelineState» mPSOs;
375
376         //Stores the objects the scene is going to render.
377         //Group objects together that have their vertices in the same vertex buffer.
378         std::unordered_map<unsigned int, std::vector<FAShapes::DrawArguments» mObjects;
379
380         //Stores data that will not be updated on a per-frame basis.
381         std::unordered_map<unsigned int, StaticBuffer> mStaticBuffers;
382
383         //Stores data that will be updated on a per-frame basis.
384         //We can't update a dynamic buffer until the GPU
385         //is done executing all the commands that reference it, so each frame needs its own dynamic
    buffer.
386         std::unordered_map<unsigned int, std::vector<DynamicBuffer» mDynamicBuffers;
387
388         //All of the text that is rendered with the scene.
389         std::unordered_map<unsigned int, Text> mTexts;
390
391         //The camera for the scene.
392         FACamera::Camera mCamera;
393     };
394 }
```

6.14 FASwapChain.h

```
1 #pragma once
2
3 #include <wrl.h>
```

```
4 #include "d3dx12.h"
5 #include <dxgi1_4.h>
6 #include <vector>
7 #include "FABuffer.h"
8
9 namespace FARender
10 {
14     class SwapChain
15     {
16     public:
17
18         SwapChain() = default;
19
29         SwapChain(const Microsoft::WRL::ComPtr<IDXGIFactory4>& dxgiFactory,
30             const Microsoft::WRL::ComPtr<ID3D12CommandQueue>& commandQueue, HWND windowHandle,
31             DXGI_FORMAT rtFormat = DXGI_FORMAT_R8G8B8A8_UNORM, DXGI_FORMAT dsFormat =
    DXGI_FORMAT_D24_UNORM_S8_UINT,
32             unsigned int numRenderTargetBuffers = 2);
33
36         const RenderTargetBuffer* GetRenderTargetBuffers() const;
37
40         const Microsoft::WRL::ComPtr<ID3D12Resource>& GetCurrentBackBuffer() const;
41
44         unsigned int GetNumRenderTargetBuffers() const;
45
48         unsigned int GetCurrentBackBufferIndex() const;
49
52         DXGI_FORMAT GetBackBufferFormat() const;
53
56         DXGI_FORMAT GetDepthStencilFormat() const;
57
60         void ResetBuffers();
61
67         void ResizeSwapChain(unsigned width, unsigned height);
68
76         void CreateRenderTargetBuffersAndViews(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
77             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int
    indexOfWhereToStoreFirstView,
78             unsigned int rtvSize);
79
89         void CreateDepthStencilBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
90             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int index, unsigned int
    dsvSize,
91             unsigned int width, unsigned int height);
92
103        void ClearCurrentBackBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
    commandList,
104            const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int indexOfFirstView,
    unsigned int rtvSize,
105            const float* backBufferClearValue);
106
117        void ClearDepthStencilBuffer(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>&
    commandList,
118            const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int indexOfView,
    unsigned int dsvSize,
119            float clearValue);
120
125        void Transition(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList,
126            D3D12_RESOURCE_STATES before, D3D12_RESOURCE_STATES after);
127
130        void Present();
131
132    private:
133        unsigned int mNumRenderTargetBuffers = 0;
134        unsigned int mCurrentBackBufferIndex = 0;
135
136        Microsoft::WRL::ComPtr<IDXGISwapChain1> mSwapChain;
137        std::vector<RenderTargetBuffer> mRenderTargetBuffers;
138
139        DepthStencilBuffer mDepthStencilBuffer;
140    };
141 }
```

6.15 FAText.h File Reference

File has class Text under namespace FARender.

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

Classes

- class FARender::Text

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

Namespaces

- namespace FARender

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

6.15.1 Detailed Description

File has class Text under namespace FARender.

6.16 FAText.h

Go to the documentation of this file.

```
1 #pragma once
2
7 #include <string>
8 #include "FAColor.h"
9
10 namespace FARender
11 {
16     class Text
17     {
18     public:
19
20         Text() = default;
21
32         Text(const FAMath::Vector4D& textLocation, const std::wstring& textString, float textSize, const
      FAColor::Color& textColor);
33
36         const FAMath::Vector4D& GetTextLocation() const;
37
40         const std::wstring& GetTextString() const;
41
44         float GetTextSize() const;
45
48         const FAColor::Color& GetTextColor() const;
49
52         void SetTextSize(float textSize);
53
56         void SetTextColor(const FAColor::Color& textColor);
57
60         void SetTextString(const std::wstring& textString);
61
64         void SetTextLocation(const FAMath::Vector4D& textLocation);
65
66     private:
67
68         FAMath::Vector4D mTextLocation;
69         std::wstring mText;
70         float mTextSize{ 0.0f };
71         FAColor::Color mTextColor;
72     };
73 }
```

6.17 FATextResources.h

```
1 #pragma once
2
3 #include <wrl.h>
4 #include <d3d11.h>
5 #include <d3d11on12.h>
6 #include <d2d1_3.h>
7 #include <dwrite.h>
8 #include <vector>
9 #include "FABuffer.h"
10
11 namespace FARender
12 {
16     class TextResources
17     {
18     public:
19         TextResources() = default;
20
27         TextResources(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
28             const Microsoft::WRL::ComPtr<ID3D12CommandQueue>& commandQueue, unsigned int
      numSwapChainBuffers);
29
32         const Microsoft::WRL::ComPtr<ID2D1DeviceContext>& GetDirect2DDeviceContext() const;
33
36         const Microsoft::WRL::ComPtr<IDWriteFactory>& GetDirectWriteFactory() const;
37
40         void ResetBuffers();
41
47         void ResizeBuffers(const RenderTargetBuffer* renderTargetBuffers, HWND windowHandle);
48
53         void BeforeRenderText(unsigned int currentBackBuffer);
54
59         void AfterRenderText(unsigned int currentBackBuffer);
60
61     private:
62         Microsoft::WRL::ComPtr<ID3D11Device> mDevice11;
63         Microsoft::WRL::ComPtr<ID3D11DeviceContext> mDevice11Context;
64         Microsoft::WRL::ComPtr<ID3D11On12Device> mDevice11on12;
65
66         Microsoft::WRL::ComPtr<ID2D1Device2> mDirect2DDevice;
67         Microsoft::WRL::ComPtr<ID2D1Factory3> mDirect2DFactory;
68         Microsoft::WRL::ComPtr<ID2D1DeviceContext> mDirect2DDeviceContext;
69
70         Microsoft::WRL::ComPtr<IDWriteFactory> mDirectWriteFactory;
71
72         std::vector<Microsoft::WRL::ComPtr<ID3D11Resource» mWrappedBuffers;
73         std::vector<Microsoft::WRL::ComPtr<ID2D1Bitmap1» mDirect2DBuffers;
74         std::vector<Microsoft::WRL::ComPtr<IDXGISurface» mSurfaces;
75     };
76 }
```

6.18 FATime.h File Reference

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

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

Classes

- class FATime::Time

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

Namespaces

- namespace FATime

  Has Time class.

6.18.1 Detailed Description

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

6.19 FATime.h

Go to the documentation of this file.

```
1 #pragma once
2
7 #include <Windows.h>
8
12 namespace FATime
13 {
17     class Time
18     {
19     public:
22         Time();
23
26         void Tick();
27
30         float DeltaTime() const;
31
34         void Reset();
35
38         void Stop();
39
42         void Start();
43
46         float TotalTime() const;
47
48     private:
49         __int64 mCurrTime; //holds current time stamp ti
50         __int64 mPrevTime; //holds previous time stamp ti-1
51         __int64 mStopTime; //holds the time we stopped the game/animation
52         __int64 mPausedTime; //holds how long the game/animation was paused for
53         __int64 mBaseTime; //holds the time we started / resetted
54
55         double mSecondsPerCount;
56         double mDeltaTime; //time elapsed btw frames change in t = ti - ti-1
57
58         bool mStopped; //flag to indicate if the game/animation is paused or not
59
60     };
61 }
```

6.20 FAWindow.h File Reference

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

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

Classes

- class FAWindow::Window

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

Namespaces

- namespace FAWindow

  Has Window class.

6.20.1 Detailed Description

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

6.21 FAWindow.h

Go to the documentation of this file.

```
1 #pragma once
2
7 #include <Windows.h>
8 #include <string>
9 #include <stdexcept>
10
14 namespace FAWindow
15 {
19     class Window
20     {
21     public:
22
23         Window() = default;
24
35         Window(const HINSTANCE& hInstance, const std::wstring& windowClassName, const std::wstring&
    windowName,
36             WNDPROC winProcFunction, unsigned int width, unsigned int height, void* additionalData =
    nullptr);
37
47         Window(const HINSTANCE& hInstance, const WNDCLASSEX& windowClass, const std::wstring& windowName,
48             unsigned int width, unsigned int height, void* additionalData = nullptr);
49
52         HWND GetWindowHandle() const;
53
56         unsigned int GetWidth() const ;
57
60         unsigned int GetHeight() const;
61
64         void SetWidth(unsigned int width);
65
68         void SetHeight(unsigned int height);
69
70     private:
71         HWND mWindowHandle;
72
73         WNDCLASSEX mWindowClass;
74         std::wstring mWindowClassName;
75
76         unsigned int mWidth;
77         unsigned int mHeight;
78     };
79 }
```

Index