Farouq Adepetu's Rendering Engine

Generated by Doxygen 1.9.4

Chapter 1

Namespace Index

1.1 Namespace List

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Namespace Documentation

4.1 FACamera Namespace Reference

Has Camera class.

Classes

- class Camera

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

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

4.1.1 Detailed Description

Has Camera class.

4.2 FARender Namespace Reference

The namespace has utility functions and structs, VertexBuffer, IndexBuffer, ConstantBuffer, DeviceResources, RenderScene and Text classes.

Classes

- class ConstantBuffer

  This class stores constant data in a Direct3D 12 upload buffers.
- class DeviceResources

  A wrapper for a Direct3D 12 device, swapchain, depth buffer, MSAA buffers and command objects.
- struct DrawSettings

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

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

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

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

  This class stores vertices in a Direct3D 12 default buffer.

Functions

- void nextFrame ()

  Update our current frame value to go to the next frame.

4.2.1 Detailed Description

The namespace has utility functions and structs, VertexBuffer, IndexBuffer, ConstantBuffer, DeviceResources, RenderScene and Text classes.

4.2.2 Function Documentation

4.2.2.1 nextFrame()

```
void FARender::nextFrame ( )
```

Update our current frame value to go to the next frame.

4.3 FAWindow Namespace Reference

Has Window class.

Classes

- class Window

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

4.3.1 Detailed Description

Has Window class.

Chapter 5

Class Documentation

5.1 FACamera::Camera Class Reference

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

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

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

Public Member Functions

- Camera ()

  Default Constructor.
- Camera (vec3 cameraPosition, vec3 x, vec3 y, vec3 z, float znear, float zfar, float aspectRatio, float vFov, float cameraVelocity, float rotateVelocity)

  Overloaded Constructor.
- vec3 & cameraPosition ()

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

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

  Returns the x-axis of the camera.
- vec3 y () const

  Returns the y-axis of the camera.
- vec3 z () const

  Returns the z-axis of the camera.
- mat4 viewTransformationMatrix () const

  Returns the view transformation matrix of this camera.
- float & cameraVelocity ()

  Returns a reference to the camera's velocity.
- const float & cameraVelocity () const

  Returns a constant reference to the camera's velocity.
- float & rotateVelocity ()

  Returns a reference to the camera's rotate velocity.

- const float & rotateVelocity () const

  Returns a constant reference to the camera's rotate velocity.
- void lookAt (vec3 cameraPosition, vec3 target, vec3 up)

  Defines the camera space using UVN.
- float & znear ()

  Returns a reference to the near value of the frustrum.
- const float & znear () const

  Returns a constant reference to the near value of the frustrum.
- float & zfar ()

  Returns a reference to the far value of the frustrum.
- const float & zfar () const

  Returns a constant reference to the far value of the frustrum.
- float & vFov ()

  Returns a reference to the vertical field of view of the frustrum in degrees.
- const float & vFov () const

  Returns a constant reference to the vertical field of view of the frustrum in degrees.
- float & aspect ()

  Returns a reference to the aspect ratio of the frustrum.
- const float & aspect () const

  Returns a constant reference to the aspect ratio of the frustrum.
- mat4 perspectiveProjectionMatrix ()

  Returns the perspective projection transformation matrix of this camera.
- mat4 viewPerspectiveProjectionMatrix ()

  Returns the view perspective projection transformation matrix of this camera.
- void updateViewMatrix ()

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

  After modifying any of the frustrum properties, call this to rebuild the perspective projection transformation matrix.
- void updateViewPerspectiveProjectionMatrix ()

  After modifying view and/or perspective projection transformation matrix, call this to rebuild the view perspective projection transformation matrix.
- void left (float dt)

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

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

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

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

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

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

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

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

  Polls keyboard input and moves the camera. Moves the camera foward/backward if w/s or up/down arrow was pressed. Moves the camera left/right if a/d or left/right arrow was pressed. Moves the camera up/down if space/crtl was pressed.
- void mouseInput ()

  Rotates camera on mouse movement.

5.1.1 Detailed Description

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

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

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Camera() [1/2]

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

Default Constructor.

Creates a new camera.
The origin of the camera space is (0.0f, 0.0f, 0.0f).
The x, y and z axes of the camera space is the same as the x, y and z axes as world space.
Sets the frustrum properties for perspective projection
to the values:
znear = 1.0f
zfar = 1000.0f
aspect ratio = 1.0f
fov = 45 degrees
The constant velocity of the camera when moved is 10.0f. The rotate velocity is 0.5f.

5.1.2.2 Camera() [2/2]

```
FACamera::Camera::Camera (
            vec3 cameraPosition,
            vec3 x,
            vec3 y,
            vec3 z,
            float znear,
            float zfar,
            float aspectRatio,
            float vFov,
            float cameraVelocity,
            float rotateVelocity )
```

Overloaded Constructor.

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

5.1.3 Member Function Documentation

5.1.3.1 aspect() [1/2]

```
float & FACamera::Camera::aspect ( )
```

Returns a reference to the aspect ratio of the frustrum.

5.1.3.2 aspect() [2/2]

```
const float & FACamera::Camera::aspect ( ) const
```

Returns a constant reference to the aspect ratio of the frustrum.

5.1.3.3 backward()

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

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

5.1.3.4 cameraPosition() [1/2]

```
vec3 & FACamera::Camera::cameraPosition ( )
```

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

5.1.3.5 cameraPosition() [2/2]

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

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

5.1.3.6 cameraVelocity() [1/2]

```
float & FACamera::Camera::cameraVelocity ( )
```

Returns a reference to the camera's velocity.

5.1.3.7 cameraVelocity() [2/2]

```
const float & FACamera::Camera::cameraVelocity ( ) const
```

Returns a constant reference to the camera's velocity.

5.1.3.8 down()

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

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

5.1.3.9 foward()

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

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

5.1.3.10 keyboardInput()

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

Polls keyboard input and moves the camera. Moves the camera foward/backward if w/s or up/down arrow was pressed. Moves the camera left/right if a/d or left/right arrow was pressed. Moves the camera up/down if space/crtl was pressed.

5.1.3.11 left()

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

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

5.1.3.12 lookAt()

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

Defines the camera space using UVN.

5.1.3.13 mouseInput()

```
void FACamera::Camera::mouseInput ( )
```

Rotates camera on mouse movement.

5.1.3.14 perspectiveProjectionMatrix()

```
mat4 FACamera::Camera::perspectiveProjectionMatrix ( )
```

Returns the perspective projection transformation matrix of this camera.

5.1.3.15 right()

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

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

5.1.3.16 rotateCameraLeftRight()

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

Rotates the camera to look left and right.

5.1.3.17 rotateCameraUpDown()

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

Rotates the camera to look up and down.

5.1.3.18 rotateVelocity() [1/2]

```
float & FACamera::Camera::rotateVelocity ( )
```

Returns a reference to the camera's rotate velocity.

5.1.3.19 rotateVelocity() [2/2]

```
const float & FACamera::Camera::rotateVelocity ( ) const
```

Returns a constant reference to the camera's rotate velocity.

5.1.3.20 up()

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

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

5.1.3.21 updatePerspectiveProjectionMatrix()

```
void FACamera::Camera::updatePerspectiveProjectionMatrix ( )
```

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

5.1.3.22 updateViewMatrix()

```
void FACamera::Camera::updateViewMatrix ( )
```

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

5.1.3.23 updateViewPerspectiveProjectionMatrix()

```
void FACamera::Camera::updateViewPerspectiveProjectionMatrix ( )
```

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

5.1.3.24 vFov() [1/2]

```
float & FACamera::Camera::vFov ( )
```

Returns a reference to the vertical field of view of the frustrum in degrees.

5.1.3.25 vFov() [2/2]

```
const float & FACamera::Camera::vFov ( ) const
```

Returns a constant reference to the vertical field of view of the frustrum in degrees.

5.1.3.26 viewPerspectiveProjectionMatrix()

```
mat4 FACamera::Camera::viewPerspectiveProjectionMatrix ( )
```

Returns the view perspective projection transformation matrix of this camera.

5.1.3.27 viewTransformationMatrix()

```
mat4 FACamera::Camera::viewTransformationMatrix ( ) const
```

Returns the view transformation matrix of this camera.

5.1.3.28 x()

```
vec3 FACamera::Camera::x ( ) const
```

Returns the x-axis of the camera.

5.1.3.29 y()

```
vec3 FACamera::Camera::y ( ) const
```

Returns the y-axis of the camera.

5.1.3.30 z()

```
vec3 FACamera::Camera::z ( ) const
```

Returns the z-axis of the camera.

5.1.3.31 zfar() [1/2]

```
float & FACamera::Camera::zfar ( )
```

Returns a reference to the far value of the frustrum.

5.1.3.32 zfar() [2/2]

```
const float & FACamera::Camera::zfar ( ) const
```

Returns a constant reference to the far value of the frustrum.

5.1.3.33 znear() [1/2]

```
float & FACamera::Camera::znear ( )
```

Returns a reference to the near value of the frustrum.

5.1.3.34 znear() [2/2]

```
const float & FACamera::Camera::znear ( ) const
```

Returns a constant reference to the near value of the frustrum.

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 ()

  Default Constructor. Initializes the color to black (0.0, 0.0, 0.0, 1.0).
- Color (const FAMath::Vector4D &color)

  Overloaded Constructor. Initializes the color to the specified color.
- Color (float r, float g, float b, float a)

  Overloaded Constructor. Initializes the color to the specified RGBA values.
- 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.
- 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.
- Color & operator+= (const Color &c)

  Adds this objects color to the specified color and stores the result in this object. Does component-wise addtion. If any of the resultant components are $>$ 1.0f, they are set to 1.0f.
- Color & operator-= (const Color &c)

  Subtracts the specified color from this objects color and stores the result in this object. Does component-wise subtraction. If any of the resultant components are $<$ 0.0f, they are set to 0.0f.
- Color & operator*= (float k)

  Multiplies this objects color by the specified float value k and stores the result in this object. If k $<$ 0.0f, no multiplication happens and this objects color does not get modified.
  If any of the resultant components are $>$ 1.0f, they are set to 1.0f.
  .
- Color & operator*= (const Color &c)

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

5.2.1 Detailed Description

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

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Color() [1/3]

```
FAColor::Color::Color ( )
```

Default Constructor. Initializes the color to black (0.0, 0.0, 0.0, 1.0).

5.2.2.2 Color() [2/3]

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

Overloaded Constructor. Initializes the color to the specified color.

5.2.2.3 Color() [3/3]

```
FAColor::Color::Color (
            float r,
            float g,
            float b,
            float a )
```

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

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()

```
FAMath::Vector4D FAColor::Color::getColor ( ) const
```

Returns the color.

5.2.3.4 getGreen()

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

Returns the value of the blue component.

5.2.3.5 getRed()

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

Returns the value of the red component.

5.2.3.6 operator*=() [1/2]

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

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

5.2.3.7 operator*=() [2/2]

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

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

5.2.3.8 operator+=()

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

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

5.2.3.9 operator-=()

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

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

5.2.3.10 setAlpha()

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

Sets the alpha component to the specified float value.

5.2.3.11 setBlue()

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

Sets the blue component to the specified float value.

5.2.3.12 setColor()

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

Sets the color to the specified color.

5.2.3.13 setGreen()

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

Sets the green component to the specified float value.

5.2.3.14 setRed()

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

Sets the red component to the specified float value.

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

- FAColor.h

5.3 FARender::ConstantBuffer Class Reference

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

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

Public Member Functions

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

  Unmaps the pointer to the constant buffer.
- Microsoft::WRL::ComPtr< ID3D12Resource > & constantBuffer ()

  Returns a reference to the constant buffer resource.
- const Microsoft::WRL::ComPtr< ID3D12Resource > & constantBuffer () const

  Returns a constant reference to the constant buffer resource.
- BYTE *& mappedData ()

  Returns a reference to the mapped data pointer.
- void createConstantBuffer (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const UINT &numOfBytes)

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

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

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

5.3.1 Detailed Description

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

5.3.2 Constructor & Destructor Documentation

5.3.2.1 ~ConstantBuffer()

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

Unmaps the pointer to the constant buffer.

5.3.3 Member Function Documentation

5.3.3.1 constantBuffer() [1/2]

```
Microsoft::WRL::ComPtr< ID3D12Resource > & FARender::ConstantBuffer::constantBuffer ( )
```

Returns a reference to the constant buffer resource.

5.3.3.2 constantBuffer() [2/2]

```
const Microsoft::WRL::ComPtr< ID3D12Resource > & FARender::ConstantBuffer::constantBuffer ( )
const
```

Returns a constant reference to the constant buffer resource.

5.3.3.3 copyData()

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

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

5.3.3.4 createConstantBuffer()

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

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

5.3.3.5 createConstantBufferView()

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

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

5.3.3.6 mappedData()

```
BYTE *& FARender::ConstantBuffer::mappedData ( )
```

Returns a reference to the mapped data pointer.

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

- FABuffer.h

5.4 FARender::DeviceResources Class Reference

A wrapper for a Direct3D 12 device, swapchain, depth buffer, MSAA buffers and command objects.

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

Public Member Functions

- **DeviceResources** (unsigned int width, unsigned int height, HWND windowHandle)
- **DeviceResources** (const DeviceResources &)=delete
- DeviceResources & **operator=** (const DeviceResources &)=delete
- ~DeviceResources ()

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

  Returns a constant reference to the ID3D12Device objcet.
- const Microsoft::WRL::ComPtr< ID3D12CommandQueue > & commandQueue () const

  Returns a constant reference to the ID3D12CommandQueue objcet.
- const Microsoft::WRL::ComPtr< ID3D12CommandAllocator > & commandAllocator () const

  Returns a constant reference to the current ID3D12CommandAllocator objcet.
- const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & commandList () const

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

  Returns a constant reference to the back buffer format.
- const UINT numOfSwapChainBuffers () const

  Returns a constant reference to the number of swap chains.
- const Microsoft::WRL::ComPtr< IDXGISwapChain1 > & swapChain () const

  Returns a constant reference to the IDXGISwapChain1 object.
- const UINT & rtvDescriptorSize () const

  Returns a constant reference to the render target view descriptor size.
- const UINT & dsvDescriptorSize () const

  Returns a constant reference to the depth/stencil view descriptor size.
- const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & rtvDescriptorHeap () const

  Returns a constant reference to the render target descriptor heap.
- const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & dsvDescriptorHeap () const

  Returns a constant reference to the depth/stencil descriptor heap.
- const UINT & currentBackBuffer () const

  Returns a constant reference to the current back buffer.
- const Microsoft::WRL::ComPtr< ID3D12Resource > * swapChainBuffers () const

  Returns a pointer to the swap chain buffers.
  There are two swap chain buffers.
  To access each buffer do swapChainBuffers()[i], where i is the index of the buffer you want to access.
- const Microsoft::WRL::ComPtr< ID3D12Resource > & depthStencilBuffer () const

  Returns a constant reference to the depth stencil buffer.
- const DXGI_FORMAT & depthStencilFormat () const

  Returns a constant reference to the depth stencil format.
- const D3D12_VIEWPORT & viewport () const

  Returns a constant reference to the D3D12_VIEWPORT objcet.
- const D3D12_RECT & scissor () const

  Returns a constant reference to the D3D12_RECT scissor objcet.
- bool & isMSAAEnabled ()

  Returns a reference to check if MSAA is enabled or not.
- const bool & isMSAAEnabled () const

  Returns a constant reference to check if MSAA is enabled or not.
- UINT & sampleCount ()

  Returns a reference to the sample count.
- const UINT & sampleCount () const

  Returns a constant reference to the sample count.
- UINT64 & currentFenceValue ()

Returns a reference to the current fence value.

- const UINT64 & currentFenceValue () const

  Returns a constant reference to the current fence value.

- const Microsoft::WRL::ComPtr< ID2D1DeviceContext > & device2DContext () const

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

- const Microsoft::WRL::ComPtr< IDWriteFactory > & directWriteFactory () const

  Returns a constant reference to the direct direct write factory.

- void updateCurrentFrameFenceValue ()

  Updates the current frames fence value.

- void initializeDirect3D (unsigned int width, unsigned int height, HWND handle)

  Initializes Direct3D. Enables the debug layer if in debug mode.
  Creates a Direct3D 12 device.
  Creates a DXGI factory object.
  Creates a fence.
  Queries descriptor sizes.
  Creates command objects.
  Creates a swap chain.
  Creates a render target view and a depth/stencil view heap. Creates the initial render target buffers, depth stencil buffer, MSAA buffers and text buffers.

- void flushCommandQueue ()

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

- void waitForGPU () const

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

- void signal ()

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

- void resize (int width, int height, const HWND &handle)

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

- void resetCommandList ()

  Resets the command list to open it with a current frame command allocator.

- void **resetDirectCommandList** ()
- void resetCommandAllocator ()

  Resets command allocator to allow reuse of the memory.

- void rtBufferTransition (Text *text)
- void **textDraw** (Text *textToRender=nullptr, UINT numText=0)
- void execute () const

  Executes the command list.

- void present ()

  Swaps the front and back buffers.

- void **draw** ()

5.4.1 Detailed Description

A wrapper for a Direct3D 12 device, swapchain, depth buffer, MSAA buffers and command objects.

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 backBufferFormat()

```
const DXGI_FORMAT & FARender::DeviceResources::backBufferFormat ( ) const
```

Returns a constant reference to the back buffer format.

5.4.3.2 commandAllocator()

```
const Microsoft::WRL::ComPtr< ID3D12CommandAllocator > & FARender::DeviceResources::command↩
Allocator ( ) const
```

Returns a constant reference to the current ID3D12CommandAllocator objcet.

5.4.3.3 commandList()

```
const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & FARender::DeviceResources::command↩
List ( ) const
```

Returns a constant reference to the ID3D12GraphicsCommandList objcet.

5.4.3.4 commandQueue()

```
const Microsoft::WRL::ComPtr< ID3D12CommandQueue > & FARender::DeviceResources::commandQueue (
) const
```

Returns a constant reference to the ID3D12CommandQueue objcet.

5.4.3.5 currentBackBuffer()

```
const UINT & FARender::DeviceResources::currentBackBuffer ( ) const
```

Returns a constant reference to the current back buffer.

5.4.3.6 currentFenceValue() [1/2]

```
UINT64 & FARender::DeviceResources::currentFenceValue ( )
```

Returns a reference to the current fence value.

5.4.3.7 currentFenceValue() [2/2]

```
const UINT64 & FARender::DeviceResources::currentFenceValue ( ) const
```

Returns a constant reference to the current fence value.

5.4.3.8 depthStencilBuffer()

```
const Microsoft::WRL::ComPtr< ID3D12Resource > & FARender::DeviceResources::depthStencilBuffer
( ) const
```

Returns a constant reference to the depth stencil buffer.

5.4.3.9 depthStencilFormat()

```
const DXGI_FORMAT & FARender::DeviceResources::depthStencilFormat ( ) const
```

Returns a constant reference to the depth stencil format.

5.4.3.10 device()

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

Returns a constant reference to the ID3D12Device objcet.

5.4.3.11 device2DContext()

```
const Microsoft::WRL::ComPtr< ID2D1DeviceContext > & FARender::DeviceResources::device2↩
DContext ( ) const
```

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

5.4.3.12 directWriteFactory()

```
const Microsoft::WRL::ComPtr< IDWriteFactory > & FARender::DeviceResources::directWriteFactory
( ) const
```

Returns a constant reference to the direct direct write factory.

5.4.3.13 dsvDescriptorHeap()

```
const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & FARender::DeviceResources::dsvDescriptor↩
Heap ( ) const
```

Returns a constant reference to the depth/stencil descriptor heap.

5.4.3.14 dsvDescriptorSize()

```
const UINT & FARender::DeviceResources::dsvDescriptorSize ( ) const
```

Returns a constant reference to the depth/stencil view descriptor size.

5.4.3.15 execute()

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

Executes the command list.

5.4.3.16 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.

5.4.3.17 initializeDirect3D()

```
void FARender::DeviceResources::initializeDirect3D (
            unsigned int width,
            unsigned int height,
            HWND handle )
```

Initializes Direct3D. Enables the debug layer if in debug mode.
Creates a Direct3D 12 device.
Creates a DXGI factory object.
Creates a fence.
Queries descriptor sizes.
Creates command objects.
Creates a swap chain.
Creates a render target view and a depth/stencil view heap. Creates the initial render target buffers, depth stencil buffer, MSAA buffers and text buffers.

5.4.3.18 isMSAAEnabled() [1/2]

```
bool & FARender::DeviceResources::isMSAAEnabled ( )
```

Returns a reference to check if MSAA is enabled or not.

5.4.3.19 isMSAAEnabled() [2/2]

```
const bool & FARender::DeviceResources::isMSAAEnabled ( ) const
```

Returns a constant reference to check if MSAA is enabled or not.

5.4.3.20 numOfSwapChainBuffers()

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

Returns a constant reference to the number of swap chains.

5.4.3.21 present()

```
void FARender::DeviceResources::present ( )
```

Swaps the front and back buffers.

5.4.3.22 resetCommandAllocator()

```
void FARender::DeviceResources::resetCommandAllocator ( )
```

Resets command allocator to allow reuse of the memory.

5.4.3.23 resetCommandList()

```
void FARender::DeviceResources::resetCommandList ( )
```

Resets the command list to open it with a current frame command allocator.

5.4.3.24 resize()

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

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

5.4.3.25 rtBufferTransition()

```
void FARender::DeviceResources::rtBufferTransition (
            Text * text )
```

@briefTransistions the render target buffer.

5.4.3.26 rtvDescriptorHeap()

```
const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & FARender::DeviceResources::rtvDescriptor↩
Heap ( ) const
```

Returns a constant reference to the render target descriptor heap.

5.4.3.27 rtvDescriptorSize()

```
const UINT & FARender::DeviceResources::rtvDescriptorSize ( ) const
```

Returns a constant reference to the render target view descriptor size.

5.4.3.28 sampleCount() [1/2]

```
UINT & FARender::DeviceResources::sampleCount ( )
```

Returns a reference to the sample count.

5.4.3.29 sampleCount() [2/2]

```
const UINT & FARender::DeviceResources::sampleCount ( ) const
```

Returns a constant reference to the sample count.

5.4.3.30 scissor()

```
const D3D12_RECT & FARender::DeviceResources::scissor ( ) const
```

Returns a constant reference to the D3D12_RECT scissor objcet.

5.4.3.31 signal()

```
void FARender::DeviceResources::signal ( )
```

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

5.4.3.32 swapChain()

```
const Microsoft::WRL::ComPtr< IDXGISwapChain1 > & FARender::DeviceResources::swapChain ( )
const
```

Returns a constant reference to the IDXGISwapChain1 object.

5.4.3.33 swapChainBuffers()

```
const Microsoft::WRL::ComPtr< ID3D12Resource > * FARender::DeviceResources::swapChainBuffers (
) const
```

Returns a pointer to the swap chain buffers.
There are two swap chain buffers.
To access each buffer do swapChainBuffers()[i], where i is the index of the buffer you want to access.

5.4.3.34 updateCurrentFrameFenceValue()

```
void FARender::DeviceResources::updateCurrentFrameFenceValue ( )
```

Updates the current frames fence value.

5.4.3.35 viewport()

```
const D3D12_VIEWPORT & FARender::DeviceResources::viewport ( ) const
```

Returns a constant reference to the D3D12_VIEWPORT objcet.

5.4.3.36 waitForGPU()

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

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

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

- FADeviceResources.h

5.5 DirectXException Class Reference

Public Member Functions

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

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

- FADirectXException.h

5.6 FARender::DrawSettings Struct Reference

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

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

Public Attributes

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

5.6.1 Detailed Description

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

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

- FARenderScene.h

5.7 FARender::IndexBuffer Class Reference

This class stores indices in a Direct3D 12 default buffer.

`#include "FABuffer.h"`

Public Member Functions

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

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

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

  Creates the vertex buffer view and stores it.

5.7.1 Detailed Description

This class stores indices in a Direct3D 12 default buffer.

5.7.2 Member Function Documentation

5.7.2.1 createIndexBuffer()

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

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

5.7.2.2 createIndexBufferView()

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

Creates the vertex buffer view and stores it.

5.7.2.3 indexBufferView()

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

Returns a constant reference to the vertex buffer view.

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

- FABuffer.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 handle)
- **RenderScene** (const RenderScene &)=delete
- RenderScene & **operator=** (const RenderScene &)=delete
- DeviceResources & **deviceResources** ()
- const DeviceResources & **deviceResources** () const
- const Microsoft::WRL::ComPtr< ID3DBlob > & **shader** (const std::wstring &name) const
- const std::vector< D3D12_INPUT_ELEMENT_DESC > & **inputElementLayout** (const std::wstring &name) const
- const D3D12_RASTERIZER_DESC & **rasterizationState** (const std::wstring &name) const
- const Microsoft::WRL::ComPtr< ID3D12PipelineState > & **pso** (const std::wstring &drawSettingsName) const
- const Microsoft::WRL::ComPtr< ID3D12RootSignature > & **rootSignature** (const std::wstring &drawSettingsName) const
- const D3D_PRIMITIVE_TOPOLOGY & **primitive** (const std::wstring &drawSettingsName) const
- FAShapes::DrawArguments & **drawArguments** (const std::wstring &drawSettingsName, unsigned int index)
- const FAShapes::DrawArguments & **drawArguments** (const std::wstring &drawSettingsName, unsigned int index) const
- ConstantBuffer & **cBuffer** ()
- const ConstantBuffer & **cBuffer** () const
- const UINT & cbvSize () const

  Returns a constant reference to the CBV/SRV/UAV descriptor size.
- const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & cbvHeap () const

  Returns a constant reference to the CBV descriptor heap.
- const D3D12_ROOT_PARAMETER & cbvHeapRootParameter () const

  Returns a constant reference to the CBV's heap root parameter.
- void **loadShader** (const std::wstring &filename, const std::wstring &name)
- void **storeInputElementDescriptions** (const std::wstring &name, const std::vector< D3D12_INPUT_ELEMENT_DESC > &inputElementLayout)
- void **storeInputElementDescriptions** (const std::wstring &name, const D3D12_INPUT_ELEMENT_DESC *inputElementLayout, UINT numElements)
- void **createRasterizationState** (D3D12_FILL_MODE fillMode, BOOL enableMultisample, const std::wstring &name)
- void **createPSO** (const std::wstring &drawSettingsName, const std::wstring &rStateName, const std::wstring &vsName, const std::wstring &psName, const std::wstring &inputLayoutName, const D3D12_PRIMITIVE_TOPOLOGY_TYPE &primitiveType, UINT sampleCount)
- void **createRootSignature** (const std::wstring &drawSettingsName, const D3D12_ROOT_PARAMETER *rootParameters, UINT numParameters)
- void **createVertexBuffer** (const void *data, UINT numBytes, UINT stride)
- void createIndexBuffer (const void *data, UINT numBytes, DXGI_FORMAT format)

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

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

  Creates a constant buffer for each frame and stores it with the specified name.
- void createConstantBufferView (UINT index, UINT numBytes)

  Creates a constant buffer view for eacg frame and stores it in the CBV heap.
- void setPSO (const std::wstring &drawSettingsName, const Microsoft::WRL::ComPtr< ID3D12PipelineState > &pso)

  Stores the specified pipeline state object with the specifed DrawSettings name.
- void setRootSignature (const std::wstring &drawSettingsName, const Microsoft::WRL::ComPtr< ID3D12RootSignature > &rootSignature)

Stores the specified root signature with the specifed DrawSettings name.

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

  Stores the specified primitive with the specifed DrawSettings name.
- void addDrawArgument (const std::wstring &drawSettingsName, const FAShapes::DrawArguments &drawArg)

  Adds the specified draw argument to the DrawArguments vector of the specified DrawSettings.
- void beforeDraw ()

  Puts all of the commands needed in the command list before drawing the objects of the scene. Call before calling the first drawObjects function.
- void drawObjects (const std::wstring &drawSettingsName)

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

  .
- void afterDraw (Text *textToRender=nullptr, UINT numText=0)

  Puts all of the commands needed in the command list after drawing the objects of the scene. Call after calling all the drawObjects functions.
- void executeAndFlush ()

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

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 (
            const std::wstring & drawSettingsName,
            const FAShapes::DrawArguments & drawArg )
```

Adds the specified draw argument to the DrawArguments vector of the specified DrawSettings.

5.8.2.2 afterDraw()

```
void FARender::RenderScene::afterDraw (
            Text * textToRender = nullptr,
            UINT numText = 0 )
```

Puts all of the commands needed in the command list after drawing the objects of the scene. Call after calling all the drawObjects functions.

5.8.2.3 beforeDraw()

```
void FARender::RenderScene::beforeDraw ( )
```

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

5.8.2.4 cbvHeap()

```
const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & FARender::RenderScene::cbvHeap ( )
const
```

Returns a constant reference to the CBV descriptor heap.

5.8.2.5 cbvHeapRootParameter()

```
const D3D12_ROOT_PARAMETER & FARender::RenderScene::cbvHeapRootParameter ( ) const
```

Returns a constant reference to the CBV's heap root parameter.

5.8.2.6 cbvSize()

```
const UINT & FARender::RenderScene::cbvSize ( ) const
```

Returns a constant reference to the CBV/SRV/UAV descriptor size.

5.8.2.7 createCBVHeap()

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

Creates the CBV heap.

5.8.2.8 createConstantBuffer()

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

Creates a constant buffer for each frame and stores it with the specified name.

5.8.2.9 createConstantBufferView()

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

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

5.8.2.10 createIndexBuffer()

```
void FARender::RenderScene::createIndexBuffer (
            const void * data,
            UINT numBytes,
            DXGI_FORMAT format )
```

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

5.8.2.11 drawObjects()

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

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

Ex.
beforeDraw()
drawObjects()
drawObjects()
afterDraw()
Throws an out_of_range exception if the specified draw settings does not exist.

5.8.2.12 executeAndFlush()

```
void FARender::RenderScene::executeAndFlush ( )
```

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

5.8.2.13 setPrimitive()

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

Stores the specified primitive with the specifed DrawSettings name.

5.8.2.14 setPSO()

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

Stores the specified pipeline state object with the specifed DrawSettings name.

5.8.2.15 setRootSignature()

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

Stores the specified root signature with the specifed DrawSettings name.

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

- FARenderScene.h

5.9 FARender::Text Class Reference

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

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

Public Member Functions

- **Text** (const Microsoft::WRL::ComPtr< ID2D1DeviceContext > &deviceContext, const Microsoft::WRL::ComPtr< IDWriteFactory > &writeFactory, const D2D1_RECT_F &textLocation, const std::wstring &textString, float textSize, const D2D1_COLOR_F &textColor)
- void initialize (const Microsoft::WRL::ComPtr< ID2D1DeviceContext > &deviceContext, const Microsoft::WRL::ComPtr< IDWriteFactory > &writeFactory, const D2D1_RECT_F &textLocation, const std::wstring &textString, float textSize, const D2D1_COLOR_F &textColor)

  Initializes the format of the text.
- const D2D1_RECT_F & textLocation ()

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

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

  Returns a constant reference to the text size.
- const Microsoft::WRL::ComPtr< ID2D1SolidColorBrush > & brush ()

  Returns a constant reference to the color brush.
- const Microsoft::WRL::ComPtr< IDWriteTextFormat > & format ()

  Returns a constant reference to the format of the text.
- const D2D1_COLOR_F textColor ()

  Returns a constant reference to the text color.
- void changeTextSize (const Microsoft::WRL::ComPtr< IDWriteFactory > &mDirectWriteFactory, float textSize)

  Changes the text size to the specified size.
- void changeTextColor (const D2D1_COLOR_F &textColor)

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

  Changes the text string to the specified string.
- void changeTextLocation (const D2D1_RECT_F &textLocation)

  Changes the text location to the specified location.

5.9.1 Detailed Description

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

5.9.2 Member Function Documentation

5.9.2.1 brush()

```
const Microsoft::WRL::ComPtr< ID2D1SolidColorBrush > & FARender::Text::brush ( )
```

Returns a constant reference to the color brush.

5.9.2.2 changeTextColor()

```
void FARender::Text::changeTextColor (
            const D2D1_COLOR_F & textColor )
```

Changes the text color to the specified color.

5.9.2.3 changeTextLocation()

```
void FARender::Text::changeTextLocation (
            const D2D1_RECT_F & textLocation )
```

Changes the text location to the specified location.

5.9.2.4 changeTextSize()

```
void FARender::Text::changeTextSize (
            const Microsoft::WRL::ComPtr< IDWriteFactory > & mDirectWriteFactory,
            float textSize )
```

Changes the text size to the specified size.

5.9.2.5 changeTextString()

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

Changes the text string to the specified string.

5.9.2.6 format()

```
const Microsoft::WRL::ComPtr< IDWriteTextFormat > & FARender::Text::format ( )
```

Returns a constant reference to the format of the text.

5.9.2.7 initialize()

```
void FARender::Text::initialize (
            const Microsoft::WRL::ComPtr< ID2D1DeviceContext > & deviceContext,
            const Microsoft::WRL::ComPtr< IDWriteFactory > & writeFactory,
            const D2D1_RECT_F & textLocation,
            const std::wstring & textString,
            float textSize,
            const D2D1_COLOR_F & textColor )
```

Initializes the format of the text.

5.9.2.8 textColor()

```
const D2D1_COLOR_F FARender::Text::textColor ( )
```

Returns a constant reference to the text color.

5.9.2.9 textLocation()

```
const D2D1_RECT_F & FARender::Text::textLocation ( )
```

Returns a constant reference to the text location.

5.9.2.10 textSize()

```
const float & FARender::Text::textSize ( )
```

Returns a constant reference to the text size.

5.9.2.11 textString()

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

Returns a constant reference to the text string.

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

- FAText.h

5.10 FATime::Time Class Reference

Public Member Functions

- Time ()

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

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

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

  Resets all time variables.
- void Stop ()

  Stops the timer.
- void Start ()

  Starts the timer.
- float TotalTime () const

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

5.10.1 Constructor & Destructor Documentation

5.10.1.1 Time()

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

Default Constructor. Gets and stores the seconds per count.

5.10.2 Member Function Documentation

5.10.2.1 DeltaTime()

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

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

5.10.2.2 Reset()

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

Resets all time variables.

5.10.2.3 Start()

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

Starts the timer.

5.10.2.4 Stop()

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

Stops the timer.

5.10.2.5 Tick()

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

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

5.10.2.6 TotalTime()

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

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

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

- FATime.h

5.11 Time Class Reference

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

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

5.11.1 Detailed Description

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

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

- FATime.h

5.12 FARender::VertexBuffer Class Reference

This class stores vertices in a Direct3D 12 default buffer.

`#include "FABuffer.h"`

Public Member Functions

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

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

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

  Returns a constant reference to the vertex buffer view.

5.12.1 Detailed Description

This class stores vertices in a Direct3D 12 default buffer.

5.12.2 Member Function Documentation

5.12.2.1 createVertexBuffer()

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

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

5.12.2.2 createVertexBufferView()

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

Creates the vertex buffer view and stores it.

5.12.2.3 vertexBufferView()

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

Returns a constant reference to the vertex buffer view.

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

- FABuffer.h

5.13 FAWindow::Window Class Reference

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

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

Public Member Functions

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

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

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

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

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

  Returns the height of the window.

5.13.1 Detailed Description

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

5.13.2 Constructor & Destructor Documentation

5.13.2.1 Window() [1/2]

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

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

5.13.2.2 Window() [2/2]

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

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

5.13.3 Member Function Documentation

5.13.3.1 height()

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

Returns the height of the window.

5.13.3.2 width()

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

Returns the width of the window.

5.13.3.3 windowHandle()

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

Returns the window handle.

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

- FAWindow.h

Chapter 6

File Documentation

6.1 Direct3DLink.h

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

6.2 FABuffer.h File Reference

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

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

Classes

- class FARender::VertexBuffer

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

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

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

Namespaces

- namespace FARender

  The namespace has utility functions and structs, VertexBuffer, IndexBuffer, ConstantBuffer, DeviceResources, RenderScene and Text classes.

6.2.1 Detailed Description

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

6.3 FABuffer.h

Go to the documentation of this file.

```
1 #pragma once
2
7 #include <wrl.h>
8 #include <d3d12.h>
9
10 namespace FARender
11 {
16     class VertexBuffer
17     {
18     public:
19         VertexBuffer() = default;
20         VertexBuffer(const VertexBuffer&) = delete;
21         VertexBuffer& operator=(const VertexBuffer&) = delete;
22
25         void createVertexBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
26             const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const void* data, UINT
    numBytes);
27
30         void createVertexBufferView(UINT numBytes, UINT stride);
31
34         const D3D12_VERTEX_BUFFER_VIEW& vertexBufferView();
35
36     private:
37         Microsoft::WRL::ComPtr<ID3D12Resource> mVertexDefaultBuffer;
38         Microsoft::WRL::ComPtr<ID3D12Resource> mVertexUploadBuffer;
39         D3D12_VERTEX_BUFFER_VIEW mVertexBufferView{};
40     };
41
46     class IndexBuffer
47     {
48     public:
49         IndexBuffer() = default;
50         IndexBuffer(const IndexBuffer&) = delete;
51         IndexBuffer& operator=(const IndexBuffer&) = delete;
52
55         const D3D12_INDEX_BUFFER_VIEW& indexBufferView();
56
59         void createIndexBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
60             const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const void* data, UINT
    numBytes);
61
64         void createIndexBufferView(UINT numBytes, DXGI_FORMAT format);
65
66     private:
67         Microsoft::WRL::ComPtr<ID3D12Resource> mIndexDefaultBuffer;
68         Microsoft::WRL::ComPtr<ID3D12Resource> mIndexUploadBuffer;
69         D3D12_INDEX_BUFFER_VIEW mIndexBufferView;
70     };
71
76     class ConstantBuffer
77     {
78     public:
79         ConstantBuffer() = default;
80
81         ConstantBuffer(const ConstantBuffer&) = delete;
82         ConstantBuffer& operator=(const ConstantBuffer&) = delete;
83
86         ~ConstantBuffer();
87
90         Microsoft::WRL::ComPtr<ID3D12Resource>& constantBuffer();
91
94         const Microsoft::WRL::ComPtr<ID3D12Resource>& constantBuffer() const;
95
98         BYTE*& mappedData();
99
103         void createConstantBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device, const UINT&
    numOfBytes);
104
108         void createConstantBufferView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
109             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& cbvHeap, UINT cbvSize, UINT
    cBufferIndex,
110             UINT cbvHeapIndex, UINT numBytes);
```

```
111
115         void copyData(UINT index, UINT byteSize, const void* data, const UINT64& numOfBytes);
116
117     private:
118         Microsoft::WRL::ComPtr<ID3D12Resource> mConstantBuffer;
119         BYTE* mMappedData{ nullptr };
120     };
121 }
```

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:
33
48         Camera();
49
61         Camera(vec3 cameraPosition, vec3 x, vec3 y, vec3 z,
62             float znear, float zfar, float aspectRatio, float vFov, float cameraVelocity, float
      rotateVelocity);
63
66         vec3& cameraPosition();
67
70         const vec3& cameraPosition() const;
71
74         vec3 x() const;
75
78         vec3 y() const;
79
82         vec3 z() const;
83
86         mat4 viewTransformationMatrix() const;
87
90         float& cameraVelocity();
91
94         const float& cameraVelocity() const;
95
98         float& rotateVelocity();
99
102         const float& rotateVelocity() const;
103
106         void lookAt(vec3 cameraPosition, vec3 target, vec3 up);
107
110         float& znear();
111
114         const float& znear() const;
115
118         float& zfar();
119
122         const float& zfar() const;
123
126         float& vFov();
127
130         const float& vFov() const;
131
134         float& aspect();
135
138         const float& aspect() const;
139
142         mat4 perspectiveProjectionMatrix();
143
146         mat4 viewPerspectiveProjectionMatrix();
147
150         void updateViewMatrix();
151
154         void updatePerspectiveProjectionMatrix();
155
159         void updateViewPerspectiveProjectionMatrix();
```

```
160
163         void left(float dt);
164
167         void right(float dt);
168
171         void foward(float dt);
172
175         void backward(float dt);
176
179         void up(float dt);
180
183         void down(float dt);
184
187         void rotateCameraLeftRight(float xDiff);
188
191         void rotateCameraUpDown(float yDiff);
192
198         void keyboardInput(float dt);
199
202         void mouseInput();
203
204     private:
205         //camera position in world coordinates
206         vec3 m_cameraPosition;
207
208         //z-axis of the camera coordinate system
209         vec3 m_n;
210
211         //y-axis of the camera coordinate system
212         vec3 m_v;
213
214         //x-axis of the camera coordinate system
215         vec3 m_u;
216
217         //stores the world to camera transform
218         mat4 m_viewMatrix;
219
220         //frustrum properties
221         float m_near;
222         float m_far;
223         float m_verticalFov;
224         float m_aspectRatio;
225         mat4 m_perspectiveProjectionMatrix;
226
227         mat4 m_viewPerspectiveProjectionMatrix;
228
229         float m_cameraVelocity;
230         float m_rotateVelocity;
231
232         vec2 lastMousePosition;
233     };
234 }
```

6.6 FAColor.h File Reference

File has class Color under namespace FAColor.

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

Classes

- class FAColor::Color

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

Functions

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

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

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

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

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

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

6.6.1 Detailed Description

File has class Color under namespace FAColor.

6.6.2 Function Documentation

6.6.2.1 operator∗() [1/3]

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

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

6.6.2.2 operator∗() [2/3]

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

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

6.6.2.3 operator∗() [3/3]

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

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

6.6.2.4 operator+()

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

Returns the result of c1 + c2. Does component-wise addtion. If any of the resultant components are > 1.0f, they are set to 1.0f.

6.6.2.5 operator-()

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

Returns the result of c1 - c2. Does component-wise subtraction. If any of the resultant components are < 0.0f, they are set to 0.0f.

6.7 FAColor.h

Go to the documentation of this file.

```
1 #pragma once
2
3 #include "FAMathEngine.h"
4
9 namespace FAColor
10 {
16     class Color
17     {
18     public:
22         Color();
23
27         Color(const FAMath::Vector4D& color);
28
32         Color(float r, float g, float b, float a);
33
36         void setColor(const FAMath::Vector4D& color);
37
40         void setRed(float r);
41
44         void setGreen(float g);
45
48         void setBlue(float b);
49
52         void setAlpha(float a);
53
```

```
56          FAMath::Vector4D getColor() const;
57
60          float getRed() const;
61
64          float getGreen() const;
65
68          float getBlue() const;
69
72          float getAlpha() const;
73
77          Color& operator+=(const Color& c);
78
82          Color& operator-=(const Color& c);
83
88          Color& operator*=(float k);
89
94          Color& operator*=(const Color& c);
95
96      private:
97          FAMath::Vector4D mColor;
98      };
99
103     Color operator+(const Color& c1, const Color& c2);
104
108     Color operator-(const Color& c1, const Color& c2);
109
114     Color operator*(const Color& c, float k);
115
120     Color operator*(float k, const Color& c);
121
125     Color operator*(const Color& c1, const Color& c2);
126 }
```

6.8 FADeviceResources.h File Reference

File has class DeviceResources under namespace FARender.

```
#include <wrl.h>
#include <d3d12.h>
#include <dxgi1_4.h>
#include <vector>
#include "FARenderingUtility.h"
#include "FAText.h"
```

Classes

- class FARender::DeviceResources

  A wrapper for a Direct3D 12 device, swapchain, depth buffer, MSAA buffers and command objects.

Namespaces

- namespace FARender

  The namespace has utility functions and structs, VertexBuffer, IndexBuffer, ConstantBuffer, DeviceResources, RenderScene and Text classes.

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 <vector>
11 #include "FARenderingUtility.h"
12 #include "FAText.h"
13
14 namespace FARender
15 {
19     class DeviceResources
20     {
21     public:
22         DeviceResources();
23
24         DeviceResources(unsigned int width, unsigned int height, HWND windowHandle);
25
26         DeviceResources(const DeviceResources&) = delete;
27         DeviceResources& operator=(const DeviceResources&) = delete;
28
31         ~DeviceResources();
32
35         const Microsoft::WRL::ComPtr<ID3D12Device>& device() const;
36
39         const Microsoft::WRL::ComPtr<ID3D12CommandQueue>& commandQueue() const;
40
43         const Microsoft::WRL::ComPtr<ID3D12CommandAllocator>& commandAllocator() const;
44
47         const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList() const;
48
51         const DXGI_FORMAT& backBufferFormat() const;
52
55         const UINT numOfSwapChainBuffers() const;
56
59         const Microsoft::WRL::ComPtr<IDXGISwapChain1>& swapChain() const;
60
63         const UINT& rtvDescriptorSize() const;
64
67         const UINT& dsvDescriptorSize() const;
68
71         const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvDescriptorHeap() const;
72
75         const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvDescriptorHeap() const;
76
79         const UINT& currentBackBuffer() const;
80
85         const Microsoft::WRL::ComPtr<ID3D12Resource>* swapChainBuffers() const;
86
89         const Microsoft::WRL::ComPtr<ID3D12Resource>& depthStencilBuffer() const;
90
93         const DXGI_FORMAT& depthStencilFormat() const;
94
97         const D3D12_VIEWPORT& viewport() const;
98
101         const D3D12_RECT& scissor() const;
102
105         bool& isMSAAEnabled();
106
109         const bool& isMSAAEnabled() const;
110
113         UINT& sampleCount();
114
117         const UINT& sampleCount() const;
118
121         UINT64& currentFenceValue();
122
125         const UINT64& currentFenceValue() const;
126
129         const Microsoft::WRL::ComPtr<ID2D1DeviceContext>& device2DContext() const;
130
133         const Microsoft::WRL::ComPtr<IDWriteFactory>& directWriteFactory() const;
134
137         void updateCurrentFrameFenceValue();
138
150         void initializeDirect3D(unsigned int width, unsigned int height, HWND handle);
151
156         void flushCommandQueue();
157
161         void waitForGPU() const;
162
165         void signal();
```

```
166
170         void resize(int width, int height, const HWND& handle);
171
174         void resetCommandList();
175
176         /*@brief Resets the command list to open it with the direct command allocator.
177 */
178         void resetDirectCommandList();
179
182         void resetCommandAllocator();
183
186         void rtBufferTransition(Text* text);
187
188         /*@brief Renders the text.
189 */
190         void textDraw(Text* textToRender = nullptr, UINT numText = 0);
191
194         void execute() const;
195
198         void present();
199
200         /*@brief Calls the necessary functions to let the user draw their objects.
201 */
202         void draw();
203
204     private:
205         Microsoft::WRL::ComPtr<ID3D12Device> mDirect3DDevice;
206
207         Microsoft::WRL::ComPtr<IDXGIFactory4> mDXGIFactory;
208
209         Microsoft::WRL::ComPtr<ID3D12Fence> mFence;
210         UINT64 mFenceValue{ 0 };
211         UINT64 mCurrentFrameFenceValue[numFrames];
212
213         Microsoft::WRL::ComPtr<ID3D12CommandQueue> mCommandQueue;
214         Microsoft::WRL::ComPtr<ID3D12CommandAllocator> mCommandAllocator[numFrames];
215         Microsoft::WRL::ComPtr<ID3D12CommandAllocator> mDirectCommandAllocator;
216         Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList> mCommandList;
217
218         DXGI_FORMAT mBackBufferFormat{ DXGI_FORMAT_R8G8B8A8_UNORM };
219         static const UINT mNumOfSwapChainBuffers{ 2 };
220         UINT mCurrentBackBuffer{ 0 };
221         Microsoft::WRL::ComPtr<IDXGISwapChain1> mSwapChain;
222         Microsoft::WRL::ComPtr<ID3D12Resource> mSwapChainBuffers[mNumOfSwapChainBuffers];
223
224         Microsoft::WRL::ComPtr<ID3D12Resource> mDepthStencilBuffer;
225         DXGI_FORMAT mDepthStencilFormat = DXGI_FORMAT_D24_UNORM_S8_UINT;
226
227         UINT mRTVSize;
228         UINT mDSVSize;
229         Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mRTVHeap;
230         Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mDSVHeap;
231
232         D3D12_VIEWPORT mViewport;
233         D3D12_RECT mScissor;
234
235         bool mMSAA4xSupported = false;
236         bool mIsMSAAEnabled = false;
237         UINT mSampleCount{ 4 };
238         Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mMSAARTVDescriptorHeap;
239         Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mMSAADSVDescriptorHeap;
240         Microsoft::WRL::ComPtr<ID3D12Resource> mMSAARenderTargetBuffer;
241         Microsoft::WRL::ComPtr<ID3D12Resource> mMSAADepthStencilBuffer;
242
243         Microsoft::WRL::ComPtr<ID3D11Device> mDevice11;
244         Microsoft::WRL::ComPtr<ID3D11DeviceContext> mDevice11Context;
245         Microsoft::WRL::ComPtr<ID3D11On12Device> mDevice11on12;
246
247         Microsoft::WRL::ComPtr<ID2D1Device2> mDirect2DDevice;
248         Microsoft::WRL::ComPtr<ID2D1Factory3> mDirect2DFactory;
249         Microsoft::WRL::ComPtr<ID2D1DeviceContext> mDirect2DDeviceContext;
250
251         Microsoft::WRL::ComPtr<IDWriteFactory> mDirectWriteFactory;
252
253         std::vector<Microsoft::WRL::ComPtr<ID3D11Resource>> mWrappedBuffers;
254         std::vector<Microsoft::WRL::ComPtr<ID2D1Bitmap1>> mDirect2DBuffers;
255         std::vector<Microsoft::WRL::ComPtr<IDXGISurface>> mSurfaces;
256
257         //Call all of these functions to initialize Direct3D
258         void enableDebugLayer();
259         void createDirect3DDevice();
260         void createDXGIFactory();
261         void createFence();
262         void queryDescriptorSizes();
263         void createCommandObjects();
264         void createSwapChain(HWND handle);
265         void createRTVHeap();
```

```
266         void createDSVHeap();
267
268         //if MSAA is supported, creates a MSAA RTV and DSV heap.
269         void checkMSAASupport();
270         void createMSAARTVHeap();
271         void createMSAADSVHeap();
272
273         //Creates and initializes everything needed to render text.
274         void initializeText();
275
276         //These functions are for creating swap chain buffers, depth/stencil buffer, render target views
    and depth/stencil view.
277         //They are called in the resize function.
278         void createRenderTargetBufferAndView();
279         void createDepthStencilBufferAndView(int width, int height);
280
281         //These functions are for creating a MSAA render target buffer, MSAA depth/stencil buffer,
282         //MSAA render target view, and a MSAA depth/stencil view.
283         //They are called in the resize function.
284         void createMSAARenderTargetBufferAndView(int width, int height);
285         void createMSAADepthStencilBufferAndView(int width, int height);
286
287         /* Resets the text buffers.
288 * Gets called in the resize function.
289 */
290         void resetTextBuffers();
291
292         /*Resizes the necessary text buffers.
293 * Gets called in the resize function.
294 */
295         void textResize(const HWND& handle);
296     };
297 }
```

6.10 FADirectXException.h

```
1 #pragma once
2
3 #include <wrl.h>
4 #include <dxgidebug.h>
5 #include <comdef.h>
6 #include <string>
7 #include <sstream>
8 #include <vector>
9
10 inline std::wstring AnsiToWString(const std::string& str)
11 {
12     WCHAR buffer[1024];
13     MultiByteToWideChar(CP_ACP, 0, str.c_str(), -1, buffer, 1024);
14     return std::wstring(buffer);
15 }
16
17 class DirectXException
18 {
19 public:
20     DirectXException(HRESULT hr, const std::wstring& functionName, const std::wstring& fileName, int
    lineNumber);
21
22     std::wstring errorMsg() const;
23
24 private:
25     HRESULT errorCode;
26     std::wstring functionName;
27     std::wstring fileName;
28     int lineNumber;
29     Microsoft::WRL::ComPtr<IDXGIInfoQueue> mInfoQueue;
30 };
31
32 //use when calling Direct3D or DXGI function to check if the function failed or not.
33 #ifndef ThrowIfFailed
34 #define ThrowIfFailed(x)                                                    \
35 {                                                                           \
36 HRESULT hr = (x);                                                   \
37 std::wstring filename(AnsiToWString(__FILE__));                     \
38 if (FAILED(hr)) { throw DirectXException(hr, L#x, filename, __LINE__); }    \
39 }
40 #endif
```

6.11 FARenderingUtility.h File Reference

File has static variables numFrames and current frame, function nextFrame() and struct DrawArguments under the namespace FARender.

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

Namespaces

- namespace FARender

  The namespace has utility functions and structs, VertexBuffer, IndexBuffer, ConstantBuffer, DeviceResources, RenderScene and Text classes.

Functions

- void FARender::nextFrame ()

  Update our current frame value to go to the next frame.

6.11.1 Detailed Description

File has static variables numFrames and current frame, function nextFrame() and struct DrawArguments under the namespace FARender.

6.12 FARenderingUtility.h

Go to the documentation of this file.

```
1 #pragma once
2
8 #include <d3d12.h>
9
14 namespace FARender
15 {
16     static const UINT numFrames{ 3 };
17     static UINT currentFrame{ 0 };
18
21     void nextFrame();
22 }
```

6.13 FARenderScene.h File Reference

File has class RenderScene under namespace FARender.

```
#include <wrl.h>
#include <d3d12.h>
#include <d3dcompiler.h>
#include <unordered_map>
#include <string>
#include "FAShapesUtility.h"
#include "FADeviceResources.h"
#include "FABuffer.h"
```

Classes

- struct FARender::DrawSettings

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

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

Namespaces

- namespace FARender

  The namespace has utility functions and structs, VertexBuffer, IndexBuffer, ConstantBuffer, DeviceResources, RenderScene and Text classes.

6.13.1 Detailed Description

File has class RenderScene under namespace FARender.

6.14 FARenderScene.h

Go to the documentation of this file.

```
1 #pragma once
2
7 #include <wrl.h>
8 #include <d3d12.h>
9 #include <d3dcompiler.h>
10 #include <unordered_map>
11 #include <string>
12 #include "FAShapesUtility.h"
13 #include "FADeviceResources.h"
14 #include "FABuffer.h"
15
16 namespace FARender
17 {
21     struct DrawSettings
22     {
23         Microsoft::WRL::ComPtr<ID3D12PipelineState> pipelineState;
24         Microsoft::WRL::ComPtr<ID3D12RootSignature> rootSig;
25         D3D_PRIMITIVE_TOPOLOGY prim = D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST;
26         std::vector<FAShapes::DrawArguments> drawArgs;
27     };
28
29
33     class RenderScene
34     {
35     public:
36
37         RenderScene(unsigned int width, unsigned int height, HWND handle);
38
39         RenderScene(const RenderScene&) = delete;
40         RenderScene& operator=(const RenderScene&) = delete;
41
42         /*@brief Returns a reference to the device resources object.
43 */
44         DeviceResources& deviceResources();
45
46         /*@brief Returns a constant reference to the device resources object.
47 */
48         const DeviceResources& deviceResources() const;
49
50         /*@brief Returns a constant reference to the shader with the specified name.
51 * Throws an out_of_range exception if the shader does not exist.
52 */
53         const Microsoft::WRL::ComPtr<ID3DBlob>& shader(const std::wstring& name) const;
54
55         /*@brief Returns a constant reference to an array of input element layout descriptions.
56 * Throws an out_of_range exception if the array of input element layout descriptions does not exist.
57 */
58         const std::vector<D3D12_INPUT_ELEMENT_DESC>& inputElementLayout(const std::wstring& name) const;
```

```
59
60          /*@brief Returns a constant reference to the rasterization description with the specified name.
61 * Throws an out_of_range exception if the rasterization description does not exist.
62 */
63          const D3D12_RASTERIZER_DESC& rasterizationState(const std::wstring& name) const;
64
65          /*@brief Returns a constant reference to the PSO in the specified DrawSettings.
66 * Throws an out_of_range exception if the DrawSettings does not exist.
67 */
68          const Microsoft::WRL::ComPtr<ID3D12PipelineState>& pso(const std::wstring& drawSettingsName)
    const;
69
70          /*@brief Returns a constant reference to the root signature in the specified DrawSettings.
71 * Throws an out_of_range exception if the DrawSettings does not exist.
72 */
73          const Microsoft::WRL::ComPtr<ID3D12RootSignature>& rootSignature(const std::wstring&
    drawSettingsName) const;
74
75          /*@brief Returns a constant reference to the primitive in the specified DrawSettings.
76 * Throws an out_of_range exception if the DrawSettings does not exist.
77 */
78          const D3D_PRIMITIVE_TOPOLOGY& primitive(const std::wstring& drawSettingsName) const;
79
80          /*@brief Returns a reference to the specified draw argument in the specified DrawSettings.
81 * Throws an out_of_range exception if the DrawSettings does not exist or if the index is out of range.
82 */
83          FAShapes::DrawArguments& drawArguments(const std::wstring& drawSettingsName, unsigned int index);
84
85          /*@brief Returns a constant reference to the specified draw argument in the specified
    DrawSettings.
86 * Throws an out_of_range exception if the DrawSettings does not exist or if the index is out of range.
87 */
88          const FAShapes::DrawArguments& drawArguments(const std::wstring& drawSettingsName, unsigned int
    index) const;
89
90          /*@brief Returns a reference to the constant buffer with the specified name.
91 * Throws an out_of_range exception if the root signature does not exist.
92 */
93          ConstantBuffer& cBuffer();
94
95          /*@brief Returns a constant reference to the constant buffer with the specified name.
96 * Throws an out_of_range exception if the root signature does not exist.
97 */
98          const ConstantBuffer& cBuffer() const;
99
102          const UINT& cbvSize() const;
103
106          const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& cbvHeap() const;
107
110          const D3D12_ROOT_PARAMETER& cbvHeapRootParameter() const;
111
112          /*@brief Loads a shader's bytecode and stores it with the specified name.
113 */
114          void loadShader(const std::wstring& filename, const std::wstring& name);
115
116          /*@brief Stores an array of input element descriptions with the specified name.
117 */
118          void storeInputElementDescriptions(const std::wstring& name, const
    std::vector<D3D12_INPUT_ELEMENT_DESC>& inputElementLayout);
119
120          /*@brief Stores an array of input element descriptions with the specified name.
121 */
122          void storeInputElementDescriptions(const std::wstring& name, const D3D12_INPUT_ELEMENT_DESC*
    inputElementLayout,
123              UINT numElements);
124
125          /*@brief Creates a rasterization description and stores it with the specified name.
126 */
127          void createRasterizationState(D3D12_FILL_MODE fillMode, BOOL enableMultisample, const
    std::wstring& name);
128
129          /*@brief Creates a PSO and stores it with the specified name.
130 * If the specifed DrawSettings, Rasterization State, Vertex Shader, Pixel Shader or Input Layout
131 * does not exist an out_of_range exception is thrown.
132 */
133          void createPSO(const std::wstring& drawSettingsName, const std::wstring& rStateName,
134              const std::wstring& vsName, const std::wstring& psName, const std::wstring& inputLayoutName,
135              const D3D12_PRIMITIVE_TOPOLOGY_TYPE& primitiveType, UINT sampleCount);
136
137          /*@brief Creates a root signature and stores it with the specified name.
138 */
139          void createRootSignature(const std::wstring& drawSettingsName,
140              const D3D12_ROOT_PARAMETER* rootParameters, UINT numParameters);
141
142          /*@brief Creates a vertex buffer with the specified name and stores all of given data in the
    vertex buffer.
143 * Also creates a view to the vertex buffer.\n
```

```
144 * Execute commands and the flush command queue after calling createVertexBuffer() and
      createIndexBuffer().
145 */
146         void createVertexBuffer(const void* data, UINT numBytes, UINT stride);
147
152         void createIndexBuffer(const void* data, UINT numBytes, DXGI_FORMAT format);
153
156         void createCBVHeap(UINT numDescriptors, UINT shaderRegister);
157
160         void createConstantBuffer(UINT numOfBytes);
161
164         void createConstantBufferView(UINT index, UINT numBytes);
165
168         void setPSO(const std::wstring& drawSettingsName, const
    Microsoft::WRL::ComPtr<ID3D12PipelineState>& pso);
169
172         void setRootSignature(const std::wstring& drawSettingsName, const
    Microsoft::WRL::ComPtr<ID3D12RootSignature>& rootSignature);
173
176         void setPrimitive(const std::wstring& drawSettingsName, const D3D_PRIMITIVE_TOPOLOGY&
    primitive);
177
180         void addDrawArgument(const std::wstring& drawSettingsName, const FAShapes::DrawArguments&
    drawArg);
181
185         void beforeDraw();
186
198         void drawObjects(const std::wstring& drawSettingsName);
199
203         void afterDraw(Text* textToRender = nullptr, UINT numText = 0);
204
207         void executeAndFlush();
208
209
210     private:
211         static DeviceResources dResources;
212
213         //Stores all of the shaders and input element descriptions for this scene.
214         std::unordered_map<std::wstring, Microsoft::WRL::ComPtr<ID3DBlob» mShaders;
215         std::unordered_map < std::wstring, std::vector<D3D12_INPUT_ELEMENT_DESC»
    mInputElementDescriptions;
216
217         //Stores all of the rasterization states.
218         std::unordered_map <std::wstring, D3D12_RASTERIZER_DESC> mRasterizationStates;
219
220         //Stores all of the possible draw settings that the scene uses.
221         std::unordered_map < std::wstring, DrawSettings> mSceneDrawSettings;
222
223         //Each scene gets one CBV heap.
224         Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mCBVHeap;
225         UINT mCBVSize;
226         D3D12_DESCRIPTOR_RANGE mCBVHeapDescription{};
227         D3D12_ROOT_PARAMETER mCBVHeapRootParameter;
228
229         //Stores all of the constant buffers this scene uses.  We can't update a constant buffer until
    the GPU
230         //is done executing all the commands that reference it, so each frame needs its own constant
    buffer.
231         ConstantBuffer mConstantBuffer[numFrames];
232
233         //The vertex and index buffer for this scene
234         VertexBuffer mVertexBuffer;
235         IndexBuffer mIndexBuffer;
236
237     };
238 }
```

6.15 FAText.h File Reference

File has class Text under namespace FARender.

```
#include <wrl.h>
#include <d3d11.h>
#include <d3d11on12.h>
#include <d2d1_3.h>
#include <dwrite.h>
#include <string>
```

Classes

- class FARender::Text

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

Namespaces

- namespace FARender

  The namespace has utility functions and structs, VertexBuffer, IndexBuffer, ConstantBuffer, DeviceResources, RenderScene and Text classes.

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 <wrl.h>
8 #include <d3d11.h>
9 #include <d3d11on12.h>
10 #include <d2d1_3.h>
11 #include <dwrite.h>
12 #include <string>
13
14 namespace FARender
15 {
20     class Text
21     {
22     public:
23
24         Text(const Microsoft::WRL::ComPtr<ID2D1DeviceContext>& deviceContext,
25             const Microsoft::WRL::ComPtr<IDWriteFactory>& writeFactory,
26             const D2D1_RECT_F& textLocation, const std::wstring& textString, float textSize, const
    D2D1_COLOR_F& textColor);
27
30         void initialize(const Microsoft::WRL::ComPtr<ID2D1DeviceContext>& deviceContext,
31             const Microsoft::WRL::ComPtr<IDWriteFactory>& writeFactory,
32             const D2D1_RECT_F& textLocation, const std::wstring& textString, float textSize, const
    D2D1_COLOR_F& textColor);
33
36         const D2D1_RECT_F& textLocation();
37
40         const std::wstring& textString();
41
44         const float& textSize();
45
48         const Microsoft::WRL::ComPtr<ID2D1SolidColorBrush>& brush();
49
52         const Microsoft::WRL::ComPtr<IDWriteTextFormat>& format();
53
56         const D2D1_COLOR_F textColor();
57
60         void changeTextSize(const Microsoft::WRL::ComPtr<IDWriteFactory>& mDirectWriteFactory, float
    textSize);
61
64         void changeTextColor(const D2D1_COLOR_F& textColor);
65
68         void changeTextString(const std::wstring& textString);
69
72         void changeTextLocation(const D2D1_RECT_F& textLocation);
73
74     private:
75
76         D2D1_RECT_F mTextLocation;
77         std::wstring mText;
78         float mTextSize;
79         D2D1_COLOR_F mTextColor;
80
81         Microsoft::WRL::ComPtr<ID2D1SolidColorBrush> mDirect2DBrush;
82         Microsoft::WRL::ComPtr<IDWriteTextFormat> mDirectWriteFormat;
83     };
84 }
```

6.17 FATime.h File Reference

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

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

Classes

- class FATime::Time

6.17.1 Detailed Description

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

6.18 FATime.h

Go to the documentation of this file.

```
1 #pragma once
2
7 #include <Windows.h>
8
12 namespace FATime
13 {
14     class Time
15     {
16     public:
20         Time();
21
24         void Tick();
25
28         float DeltaTime() const;
29
32         void Reset();
33
36         void Stop();
37
40         void Start();
41
44         float TotalTime() const;
45
46     private:
47         __int64 mCurrTime; //holds current time stamp ti
48         __int64 mPrevTime; //holds previous time stamp ti-1
49         __int64 mStopTime; //holds the time we stopped the game/animation
50         __int64 mPausedTime; //holds how long the game/animation was paused for
51         __int64 mBaseTime; //holds the time we started / resetted
52
53         double mSecondsPerCount;
54         double mDeltaTime; //time elapsed btw frames change in t = ti - ti-1
55
56         bool mStopped; //flag to indicate if the game/animation is paused or not
57
58     };
59 }
```

6.19 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.

6.19.1 Detailed Description

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

6.20 FAWindow.h

Go to the documentation of this file.

```
1 #pragma once
2
7 #include <Windows.h>
8 #include <string>
9 #include <stdexcept>
10
14 namespace FAWindow
15 {
19     class Window
20     {
21     public:
22         //Window();
23
27         Window(const HINSTANCE& hInstance, const std::wstring& windowClassName, const std::wstring&
     windowName,
28             WNDPROC winProcFunction, unsigned int width, unsigned int height, void* additionalData =
     nullptr);
29
33         Window(const HINSTANCE& hInstance, const WNDCLASSEX& windowClass, const std::wstring& windowName,
34             unsigned int width, unsigned int height, void* additionalData = nullptr);
35
38         HWND windowHandle() const;
39
42         unsigned int width() const ;
43
46         unsigned int height() const;
47
48     private:
49         HWND mWindowHandle;
50
51         WNDCLASSEX mWindowClass;
52         std::wstring mWindowClassName;
53
54         unsigned int mWidth;
55         unsigned int mHeight;
56     };
57 }
```

Index