Farouq Adepetu's DirectX 12 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 DrawArguments

  Has all the data that are used as parameters to draw an object.
- 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 perspectiveProjectionTransformationMatrix ()

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

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

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

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

  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 (FAMath::Vector2D currentMousePosition)

  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 1000.0f.

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;

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 (
            FAMath::Vector2D currentMousePosition )
```

Rotates camera on mouse movement.

5.1.3.14 perspectiveProjectionTransformationMatrix()

```
mat4 FACamera::Camera::perspectiveProjectionTransformationMatrix ( )
```

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 updatePerspectiveProjectionTransformationMatrix()

```
void FACamera::Camera::updatePerspectiveProjectionTransformationMatrix ( )
```

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

5.1.3.22 updateViewPerspectiveProjectionTransformationMatrix()

```
void FACamera::Camera::updateViewPerspectiveProjectionTransformationMatrix ( )
```

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

5.1.3.23 updateViewTransformationMatrix()

```
void FACamera::Camera::updateViewTransformationMatrix ( )
```

After modifying the camera position and/or orientation, call this to rebuild the view 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 viewPerspectiveProjectionTransformationMatrix()

```
mat4 FACamera::Camera::viewPerspectiveProjectionTransformationMatrix ( )
```

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** (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 ()

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

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

  Returns a constant reference to the direct direct write factory.

- void updateCurrentFrameFenceValue ()

  Updates the current frames fence value.

- void initializeDirect3D (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 depth/stencil view heap.

- 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 ()

  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 execute ()

  Executes the command list.

- void present ()

  Swaps the front and back buffers.

- void **draw** ()
- void **rtBufferTransition** (Text *text)
- void resetTextBuffers ()

  Resets the text buffers. Call before calling the text resize function and the rendering resize funciton.

- void textResize (const HWND &handle)

  Resizes the necessary text buffers. Call before calling the rendering resize funciton.

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

  Renders the text.

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 ( )
```

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

5.4.3.12 directWriteFactory()

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

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 ( )
```

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 (
            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 depth/stencil view heap.

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 resetTextBuffers()

```
void FARender::DeviceResources::resetTextBuffers ( )
```

Resets the text buffers. Call before calling the text resize function and the rendering resize funciton.

5.4.3.25 resize()

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

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

5.4.3.26 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 textDraw()

```
void FARender::DeviceResources::textDraw (
            Text * textToRender = nullptr,
            UINT numText = 0 )
```

Renders the text.

5.4.3.35 textResize()

```
void FARender::DeviceResources::textResize (
            const HWND & handle )
```

Resizes the necessary text buffers. Call before calling the rendering resize funciton.

5.4.3.36 updateCurrentFrameFenceValue()

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

Updates the current frames fence value.

5.4.3.37 viewport()

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

Returns a constant reference to the D3D12_VIEWPORT objcet.

5.4.3.38 waitForGPU()

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

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 line↩ Number)
- std::wstring **errorMsg** () const

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

- FADirectXException.h

5.6 FARender::DrawArguments Struct Reference

Has all the data that are used as parameters to draw an object.

`#include "FARenderingUtility.h"`

Public Attributes

- UINT **indexCount**
- UINT **locationFirstIndex**
- INT **indexOfFirstVertex**
- INT **indexOfConstantData**

5.6.1 Detailed Description

Has all the data that are used as parameters to draw an object.

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

- FARenderingUtility.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** (const RenderScene &)=delete
- RenderScene & **operator=** (const RenderScene &)=delete
- 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 &name) const
- const Microsoft::WRL::ComPtr< ID3D12RootSignature > & **rootSignature** (const std::wstring &name) const
- ConstantBuffer & **cBuffer** (const std::wstring &name)
- const ConstantBuffer & **cBuffer** (const std::wstring &name) 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.
- const DrawArguments & **drawArgument** (const std::wstring &groupName, const std::wstring &objectName) const
- 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 Microsoft::WRL::ComPtr< ID3D12Device > &device, const std::wstring &psoName, const std::wstring &rsName, const std::wstring &rStateName, const std::wstring &vsName, const std::wstring &psName, const std::wstring &inputLayoutName, const D3D12_PRIMITIVE_TOPOLOGY_TYPE &primitiveType, DXGI_FORMAT rtvFormat, DXGI_FORMAT dsvFormat, UINT sampleCount)
- void **createRootSignature** (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const std::wstring &name, const D3D12_ROOT_PARAMETER *rootParameters, UINT numParameters)
- void **storeDrawArgument** (const std::wstring &groupName, const std::wstring &objectName, const DrawArguments &drawArgs)
- void **createVertexBuffer** (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const std::wstring &vbName, const void *data, UINT numBytes)
- void **createVertexBufferView** (const std::wstring &vbName, UINT numBytes, UINT stride)
- void createIndexBuffer (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const std::wstring &ibName, const void *data, UINT numBytes)

  Creates an indexbuffer with the specified name and stores all of given data in the index buffer. Execute commands and flush the command queue after calling createVertexBuffer() and createIndexBuffer().
- void **createIndexBufferView** (const std::wstring &ibName, UINT numBytes, DXGI_FORMAT format)
- void createCBVHeap (const Microsoft::WRL::ComPtr< ID3D12Device > &device, UINT numDescriptors, UINT shaderRegister)

Creates the CBV heap.

- void createConstantBuffer (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const std::wstring &name, const UINT &numOfBytes)

  Creates a constant buffer for each frame and stores it with the specified name.
- void createConstantBufferView (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const std::wstring &name, UINT index, UINT numBytes)

  Creates a constant buffer view for eacg frame and stores it in the CBV heap.
- void beforeDraw (DeviceResources &deviceResource)

  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 Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const std::wstring &drawArgsGroupName, const std::wstring &vbName, const std::wstring &ibName, const std::wstring &psoName, const std::wstring &rootSignatureName, const D3D_PRIMITIVE_TOPOLOGY &primitive)

  Draws all of the objects that are in the same vertex and index buffers and use the same PSO and primitive. Call in between a beforeDraw function and a afterDraw function.

  .
- void afterDraw (DeviceResources &deviceResource, 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.

5.8.1 Detailed Description

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

5.8.2 Member Function Documentation

5.8.2.1 afterDraw()

```
void FARender::RenderScene::afterDraw (
            DeviceResources & deviceResource,
            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.

5.8.2.2 beforeDraw()

```
void FARender::RenderScene::beforeDraw (
            DeviceResources & deviceResource )
```

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

5.8.2.3 cbvHeap()

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

Returns a constant reference to the CBV descriptor heap.

5.8.2.4 cbvHeapRootParameter()

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

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

5.8.2.5 cbvSize()

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

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

5.8.2.6 createCBVHeap()

```
void FARender::RenderScene::createCBVHeap (
            const Microsoft::WRL::ComPtr< ID3D12Device > & device,
            UINT numDescriptors,
            UINT shaderRegister )
```

Creates the CBV heap.

5.8.2.7 createConstantBuffer()

```
void FARender::RenderScene::createConstantBuffer (
            const Microsoft::WRL::ComPtr< ID3D12Device > & device,
            const std::wstring & name,
            const UINT & numOfBytes )
```

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

5.8.2.8 createConstantBufferView()

```
void FARender::RenderScene::createConstantBufferView (
            const Microsoft::WRL::ComPtr< ID3D12Device > & device,
            const std::wstring & name,
            UINT index,
            UINT numBytes )
```

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

5.8.2.9 createIndexBuffer()

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

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

5.8.2.10 drawObjects()

```
void FARender::RenderScene::drawObjects (
            const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & commandList,
            const std::wstring & drawArgsGroupName,
            const std::wstring & vbName,
            const std::wstring & ibName,
            const std::wstring & psoName,
            const std::wstring & rootSignatureName,
            const D3D_PRIMITIVE_TOPOLOGY & primitive )
```

Draws all of the objects that are in the same vertex and index buffers and use the same PSO 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 vertex buffer, index buffer, draw argument group, PSO, or root signature does not exist.

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

- 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
- const D3D12_VERTEX_BUFFER_VIEW & vertexBufferView ()

  Returns a constant reference to the vertex buffer view.
- 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.

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

- HWND windowHandle () const

  Returns the window handle.
- unsigned int width ()

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

  Returns the height of the window.
- bool & minimized ()

  Returns a reference to the bool variable that tells you if the window is minimized or not.
- bool & maximized ()

  Returns a reference to the bool variable that tells you if the window is maximized or not.
- bool & resizing ()

  Returns a reference to the bool variable that tells you if the window is being resized or not.
- void **registerWindowClass** (const WNDCLASSEX &windowClass)
- void createWindow (const HINSTANCE &hInstance, const std::wstring &windowName, unsigned int width, unsigned int height)

  Registers the window class and creates the window.
- void showWindow ()

  Updates and shows the window.

5.13.1 Detailed Description

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

5.13.2 Member Function Documentation

5.13.2.1 createWindow()

```
void FAWindow::Window::createWindow (
            const HINSTANCE & hInstance,
            const std::wstring & windowName,
            unsigned int width,
            unsigned int height )
```

Registers the window class and creates the window.

5.13.2.2 height()

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

Returns the height of the window.

5.13.2.3 maximized()

```
bool & FAWindow::Window::maximized ( )
```

Returns a reference to the bool variable that tells you if the window is maximized or not.

5.13.2.4 minimized()

```
bool & FAWindow::Window::minimized ( )
```

Returns a reference to the bool variable that tells you if the window is minimized or not.

5.13.2.5 resizing()

```
bool & FAWindow::Window::resizing ( )
```

Returns a reference to the bool variable that tells you if the window is being resized or not.

5.13.2.6 showWindow()

```
void FAWindow::Window::showWindow ( )
```

Updates and shows the window.

5.13.2.7 width()

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

Returns the width of the window.

5.13.2.8 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         const D3D12_VERTEX_BUFFER_VIEW& vertexBufferView();
26
29         void createVertexBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
30             const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const void* data, UINT
    numBytes);
31
34         void createVertexBufferView(UINT numBytes, UINT stride);
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 "FAVector2D.h"
#include "FAVector3D.h"
#include "FAVector4D.h"
#include "FAMatrix4x4.h"
#include "FAQuaternion.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
13 #include "FAVector2D.h"
14 #include "FAVector3D.h"
15 #include "FAVector4D.h"
16 #include "FAMatrix4x4.h"
17 #include "FAQuaternion.h"
18 #include <Windows.h>
19
20 typedef FAMath::Vector2D vec2;
21 typedef FAMath::Vector3D vec3;
22 typedef FAMath::Vector4D vec4;
23 typedef FAMath::Matrix4x4 mat4;
24
28 namespace FACamera
29 {
35     class Camera
36     {
37     public:
38
52         Camera();
53
64         Camera(vec3 cameraPosition, vec3 x, vec3 y, vec3 z,
65             float znear, float zfar, float aspectRatio, float vFov, float cameraVelocity, float
      rotateVelocity);
66
69         vec3& cameraPosition();
70
73         const vec3& cameraPosition() const;
74
77         vec3 x() const;
78
81         vec3 y() const;
82
85         vec3 z() const;
86
89         mat4 viewTransformationMatrix() const;
90
93         float& cameraVelocity();
94
97         const float& cameraVelocity() const;
98
101         float& rotateVelocity();
102
105         const float& rotateVelocity() const;
106
109         void lookAt(vec3 cameraPosition, vec3 target, vec3 up);
110
113         float& znear();
114
117         const float& znear() const;
118
121         float& zfar();
122
125         const float& zfar() const;
126
129         float& vFov();
130
133         const float& vFov() const;
134
137         float& aspect();
138
141         const float& aspect() const;
142
145         mat4 perspectiveProjectionTransformationMatrix();
146
149         mat4 viewPerspectiveProjectionTransformationMatrix();
150
153         void updateViewTransformationMatrix();
```

```
154
157         void updatePerspectiveProjectionTransformationMatrix();
158
162         void updateViewPerspectiveProjectionTransformationMatrix();
163
166         void left(float dt);
167
170         void right(float dt);
171
174         void foward(float dt);
175
178         void backward(float dt);
179
182         void up(float dt);
183
186         void down(float dt);
187
190         void rotateCameraLeftRight(float xDiff);
191
194         void rotateCameraUpDown(float yDiff);
195
201         void keyboardInput(float dt);
202
205         void mouseInput(FAMath::Vector2D currentMousePosition);
206
207     private:
208         //camera position in world coordinates
209         vec3 m_cameraPosition;
210
211         //z-axis of the camera coordinate system
212         vec3 m_n;
213
214         //y-axis of the camera coordinate system
215         vec3 m_v;
216
217         //x-axis of the camera coordinate system
218         vec3 m_u;
219
220         //stores the world to camera transform
221         mat4 m_viewMatrix;
222
223         //frustrum properties
224         float m_near;
225         float m_far;
226         float m_verticalFov;
227         float m_aspectRatio;
228         mat4 m_perspectiveProjectionMatrix;
229
230         mat4 m_viewPerspectiveProjectionTransformationMatrix;
231
232         float m_cameraVelocity;
233         float m_rotateVelocity;
234
235         vec2 lastMousePosition;
236     };
237 }
```

6.6 FAColor.h File Reference

File has class Color under namespace FAColor.

```
#include "FAVector4D.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 "FAVector4D.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
23         DeviceResources() = default;
24
25         DeviceResources(const DeviceResources&) = delete;
26         DeviceResources& operator=(const DeviceResources&) = delete;
27
30         ~DeviceResources();
31
34         const Microsoft::WRL::ComPtr<ID3D12Device>& device() const;
35
38         const Microsoft::WRL::ComPtr<ID3D12CommandQueue>& commandQueue() const;
39
42         const Microsoft::WRL::ComPtr<ID3D12CommandAllocator>& commandAllocator() const;
43
46         const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList() const;
47
50         const DXGI_FORMAT& backBufferFormat() const;
51
54         const UINT numOfSwapChainBuffers() const;
55
58         const Microsoft::WRL::ComPtr<IDXGISwapChain1>& swapChain() const;
59
62         const UINT& rtvDescriptorSize() const;
63
66         const UINT& dsvDescriptorSize() const;
67
70         const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvDescriptorHeap() const;
71
74         const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvDescriptorHeap() const;
75
78         const UINT& currentBackBuffer() const;
79
84         const Microsoft::WRL::ComPtr<ID3D12Resource>* swapChainBuffers() const;
85
88         const Microsoft::WRL::ComPtr<ID3D12Resource>& depthStencilBuffer() const;
89
92         const DXGI_FORMAT& depthStencilFormat() const;
93
96         const D3D12_VIEWPORT& viewport() const;
97
100         const D3D12_RECT& scissor() const;
101
104         bool& isMSAAEnabled();
105
108         const bool& isMSAAEnabled() const;
109
112         UINT& sampleCount();
113
116         const UINT& sampleCount() const;
117
120         UINT64& currentFenceValue();
121
124         const UINT64& currentFenceValue() const;
125
128         const Microsoft::WRL::ComPtr<ID2D1DeviceContext>& device2DContext();
129
132         const Microsoft::WRL::ComPtr<IDWriteFactory>& directWriteFactory();
133
136         void updateCurrentFrameFenceValue();
137
148         void initializeDirect3D(HWND handle);
149
154         void flushCommandQueue();
155
159         void waitForGPU();
160
163         void signal();
164
```

```
168         void resize(int width, int height, const HWND& handle);
169
172         void resetCommandList();
173
174         /*@brief Resets the command list to open it with the direct command allocator.
175 */
176         void resetDirectCommandList();
177
180         void resetCommandAllocator();
181
184         void execute();
185
188         void present();
189
190         /*@brief Calls the necessary functions to let the user draw their objects.
191 */
192         void draw();
193
194         /*@brief Transistions the render target buffer.
195 */
196         void rtBufferTransition(Text* text);
197
201         void resetTextBuffers();
202
206         void textResize(const HWND& handle);
207
210         void textDraw(Text* textToRender = nullptr, UINT numText = 0);
211
212     private:
213         Microsoft::WRL::ComPtr<ID3D12Device> mDirect3DDevice;
214
215         Microsoft::WRL::ComPtr<IDXGIFactory4> mDXGIFactory;
216
217         Microsoft::WRL::ComPtr<ID3D12Fence> mFence;
218         UINT64 mFenceValue{ 0 };
219         UINT64 mCurrentFrameFenceValue[numFrames];
220
221         Microsoft::WRL::ComPtr<ID3D12CommandQueue> mCommandQueue;
222         Microsoft::WRL::ComPtr<ID3D12CommandAllocator> mCommandAllocator[numFrames];
223         Microsoft::WRL::ComPtr<ID3D12CommandAllocator> mDirectCommandAllocator;
224         Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList> mCommandList;
225
226         DXGI_FORMAT mBackBufferFormat{ DXGI_FORMAT_R8G8B8A8_UNORM };
227         static const UINT mNumOfSwapChainBuffers{ 2 };
228         UINT mCurrentBackBuffer{ 0 };
229         Microsoft::WRL::ComPtr<IDXGISwapChain1> mSwapChain;
230         Microsoft::WRL::ComPtr<ID3D12Resource> mSwapChainBuffers[mNumOfSwapChainBuffers];
231
232         Microsoft::WRL::ComPtr<ID3D12Resource> mDepthStencilBuffer;
233         DXGI_FORMAT mDepthStencilFormat = DXGI_FORMAT_D24_UNORM_S8_UINT;
234
235         UINT mRTVSize;
236         UINT mDSVSize;
237         Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mRTVHeap;
238         Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mDSVHeap;
239
240         D3D12_VIEWPORT mViewport;
241         D3D12_RECT mScissor;
242
243         bool mMSAA4xSupported = false;
244         bool mIsMSAAEnabled = false;
245         UINT mSampleCount{ 4 };
246         Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mMSAARTVDescriptorHeap;
247         Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mMSAADSVDescriptorHeap;
248         Microsoft::WRL::ComPtr<ID3D12Resource> mMSAARenderTargetBuffer;
249         Microsoft::WRL::ComPtr<ID3D12Resource> mMSAADepthStencilBuffer;
250
251
252
253         Microsoft::WRL::ComPtr<ID3D11Device> mDevice11;
254         Microsoft::WRL::ComPtr<ID3D11DeviceContext> mDevice11Context;
255         Microsoft::WRL::ComPtr<ID3D11On12Device> mDevice11on12;
256
257         Microsoft::WRL::ComPtr<ID2D1Device2> mDirect2DDevice;
258         Microsoft::WRL::ComPtr<ID2D1Factory3> mDirect2DFactory;
259         Microsoft::WRL::ComPtr<ID2D1DeviceContext> mDirect2DDeviceContext;
260
261         Microsoft::WRL::ComPtr<IDWriteFactory> mDirectWriteFactory;
262
263         std::vector<Microsoft::WRL::ComPtr<ID3D11Resource» mWrappedBuffers;
264         std::vector<Microsoft::WRL::ComPtr<ID2D1Bitmap1» mDirect2DBuffers;
265         std::vector<Microsoft::WRL::ComPtr<IDXGISurface» mSurfaces;
266
267         //Call all of these functions to initialize Direct3D
268         void enableDebugLayer();
269         void createDirect3DDevice();
270         void createDXGIFactory();
```

```
271         void createFence();
272         void queryDescriptorSizes();
273         void createCommandObjects();
274         void createSwapChain(HWND handle);
275         void createRTVHeap();
276         void createDSVHeap();
277 
278         //if MSAA is supported, creates a MSAA RTV and DSV heap.
279         void checkMSAASupport();
280         void createMSAARTVHeap();
281         void createMSAADSVHeap();
282 
283         //Creates and initializes everything needed to render text.
284         void initializeText();
285 
286         //These functions are for creating swap chain buffers, depth/stencil buffer, render target views
      and depth/stencil view.
287         //They are called in the resize function.
288         void createRenderTargetBufferAndView();
289         void createDepthStencilBufferAndView(int width, int height);
290 
291         //These functions are for creating a MSAA render target buffer, MSAA depth/stencil buffer,
292         //MSAA render target view, and a MSAA depth/stencil view.
293         //They are called in the resize function.
294         void createMSAARenderTargetBufferAndView(int width, int height);
295         void createMSAADepthStencilBufferAndView(int width, int height);
296 
297     };
298 }
```

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>
```

Classes

- struct FARender::DrawArguments

  Has all the data that are used as parameters to draw an object.

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
26     struct DrawArguments
27     {
28         UINT indexCount;
29         UINT locationFirstIndex;
30         INT indexOfFirstVertex;
31         INT indexOfConstantData;
32     };
33 }
```

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 "FARenderingUtility.h"
#include "FADeviceResources.h"
#include "FABuffer.h"
```

Classes

- 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 "FARenderingUtility.h"
13 #include "FADeviceResources.h"
14 #include "FABuffer.h"
15
16 namespace FARender
17 {
21     class RenderScene
22     {
23     public:
24
25         RenderScene() = default;
26
27         RenderScene(const RenderScene&) = delete;
28         RenderScene& operator=(const RenderScene&) = delete;
29
30         /*@brief Returns a constant reference to the shader with the specified name.
31 * Throws an out_of_range exception if the shader does not exist.
32 */
33         const Microsoft::WRL::ComPtr<ID3DBlob>& shader(const std::wstring& name) const;
34
35         /*@brief Returns a constant reference to an array of input element layout descriptions.
36 * Throws an out_of_range exception if the array of input element layout descriptions does not exist.
37 */
38         const std::vector<D3D12_INPUT_ELEMENT_DESC>& inputElementLayout(const std::wstring& name) const;
39
40         /*@brief Returns a constant reference to the rasterization description with the specified name.
41 * Throws an out_of_range exception if the rasterization description does not exist.
42 */
43         const D3D12_RASTERIZER_DESC& rasterizationState(const std::wstring& name) const;
44
45         /*@brief Returns a constant reference to the PSO with the specified name.
46 * Throws an out_of_range exception if the PSO does not exist.
47 */
48         const Microsoft::WRL::ComPtr<ID3D12PipelineState>& pso(const std::wstring& name) const;
49
50         /*@brief Returns a constant reference to the root signature with the specified name.
51 * Throws an out_of_range exception if the root signature does not exist.
52 */
53         const Microsoft::WRL::ComPtr<ID3D12RootSignature>& rootSignature(const std::wstring& name) const;
54
55         /*@brief Returns a reference to the constant buffer with the specified name.
56 * Throws an out_of_range exception if the root signature does not exist.
57 */
58         ConstantBuffer& cBuffer(const std::wstring& name);
```

```
59
60          /*@brief Returns a constant reference to the constant buffer with the specified name.
61  * Throws an out_of_range exception if the root signature does not exist.
62  */
63          const ConstantBuffer& cBuffer(const std::wstring& name) const;
64
67          const UINT& cbvSize() const;
68
71          const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& cbvHeap() const;
72
75          const D3D12_ROOT_PARAMETER& cbvHeapRootParameter() const;
76
77          /*@brief Returns a constant reference to the draw argument with the specifed name in the
      specified group.
78  * Throws an out_of_range exception if the draw argument does not exist.
79  */
80          const DrawArguments& drawArgument(const std::wstring& groupName, const std::wstring& objectName)
      const;
81
82          /*@brief Loads a shader's bytecode and stores it with the specified name.
83  */
84          void loadShader(const std::wstring& filename, const std::wstring& name);
85
86          /*@brief Stores an array of input element descriptions with the specified name.
87  */
88          void storeInputElementDescriptions(const std::wstring& name, const
      std::vector<D3D12_INPUT_ELEMENT_DESC>& inputElementLayout);
89
90          /*@brief Stores an array of input element descriptions with the specified name.
91  */
92          void storeInputElementDescriptions(const std::wstring& name, const D3D12_INPUT_ELEMENT_DESC*
      inputElementLayout, UINT numElements);
93
94          /*@brief Creates a rasterization description and stores it with the specified name.
95  */
96          void createRasterizationState(D3D12_FILL_MODE fillMode, BOOL enableMultisample, const
      std::wstring& name);
97
98          /*@brief Creates a PSO and stores it with the specified name.
99  */
100         void createPSO(const Microsoft::WRL::ComPtr<ID3D12Device>& device, const std::wstring& psoName,
101             const std::wstring& rsName, const std::wstring& rStateName, const std::wstring& vsName,
      const std::wstring& psName,
102             const std::wstring& inputLayoutName,
103             const D3D12_PRIMITIVE_TOPOLOGY_TYPE& primitiveType, DXGI_FORMAT rtvFormat, DXGI_FORMAT
      dsvFormat, UINT sampleCount);
104
105         /*@brief Creates a root signature and stores it with the specified name.
106 */
107         void createRootSignature(const Microsoft::WRL::ComPtr<ID3D12Device>& device, const std::wstring&
      name,
108             const D3D12_ROOT_PARAMETER* rootParameters, UINT numParameters);
109
110         /*@brief Stores a DrawArgument object with the specified name in the specified group.
111 */
112         void storeDrawArgument(const std::wstring& groupName, const std::wstring& objectName, const
      DrawArguments& drawArgs);
113
114         /*@brief Creates a vertex buffer with the specified name and stores all of given data in the
      vertex buffer.
115 * Execute commands and the flush command queue after calling createVertexBuffer() and
      createIndexBuffer().
116 */
117         void createVertexBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
118             const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList,
119             const std::wstring& vbName, const void* data, UINT numBytes);
120
121         /*@brief Creates a vertex buffer view for the vertex buffer with the specified name.
122 */
123         void createVertexBufferView(const std::wstring& vbName, UINT numBytes, UINT stride);
124
128         void createIndexBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
129             const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList,
130             const std::wstring& ibName, const void* data, UINT numBytes);
131
132         /*@brief Creates an index buffer view for the index buffer with the specified name.
133 */
134         void createIndexBufferView(const std::wstring& ibName, UINT numBytes, DXGI_FORMAT format);
135
138         void createCBVHeap(const Microsoft::WRL::ComPtr<ID3D12Device>& device, UINT numDescriptors, UINT
      shaderRegister);
139
142         void createConstantBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device, const
      std::wstring& name,
143             const UINT& numOfBytes);
144
147         void createConstantBufferView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
```

```
148             const std::wstring& name, UINT index, UINT numBytes);
149
153         void beforeDraw(DeviceResources& deviceResource);
154
168         void drawObjects(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList,
169             const std::wstring& drawArgsGroupName, const std::wstring& vbName, const std::wstring&
    ibName,
170             const std::wstring& psoName, const std::wstring& rootSignatureName,
171             const D3D_PRIMITIVE_TOPOLOGY& primitive);
172
176         void afterDraw(DeviceResources& deviceResource, Text* textToRender = nullptr, UINT numText = 0);
177
178     private:
179         //Stores all of the shaders and input element descriptions for this scene.
180         std::unordered_map<std::wstring, Microsoft::WRL::ComPtr<ID3DBlob>> mShaders;
181         std::unordered_map < std::wstring, std::vector<D3D12_INPUT_ELEMENT_DESC>>
    mInputElementDescriptions;
182
183         //Stores all of the rasterization states, PSOs, and root signatures for this scene.
184         std::unordered_map <std::wstring, D3D12_RASTERIZER_DESC> mRasterizationStates;
185         std::unordered_map<std::wstring, Microsoft::WRL::ComPtr<ID3D12PipelineState>> mPSOs;
186         std::unordered_map<std::wstring, Microsoft::WRL::ComPtr<ID3D12RootSignature>> mRootSignatures;
187
188         //Each scene gets one CBV heap.
189         Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mCBVHeap;
190         UINT mCBVSize;
191         D3D12_DESCRIPTOR_RANGE mCBVHeapDescription{};
192         D3D12_ROOT_PARAMETER mCBVHeapRootParameter;
193
194         //Stores all of the constant buffers this scene uses.  We can't update a constant buffer until
    the GPU
195         //is done executing all the commands that reference it, so each frame needs its own constant
    buffers.
196         std::unordered_map<std::wstring, ConstantBuffer> mConstantBuffers[numFrames];
197
198         //Groups all of the objects draw arguments that are in the same vertex buffer and index buffer,
199         //and uses the same shaders, rasterization states, PSO, and root signatures.
200         std::unordered_map<std::wstring, std::unordered_map<std::wstring, DrawArguments> > mDrawArgs;
201
202         //Stores all of the vertex buffers and index buffers for this scene.
203         std::unordered_map<std::wstring, VertexBuffer> mVertexBuffers;
204         std::unordered_map<std::wstring, IndexBuffer> mIndexBuffers;
205     };
206 }
```

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() = default;
25
28         void initialize(const Microsoft::WRL::ComPtr<ID2D1DeviceContext>& deviceContext,
29             const Microsoft::WRL::ComPtr<IDWriteFactory>& writeFactory,
30             const D2D1_RECT_F& textLocation, const std::wstring& textString, float textSize, const
      D2D1_COLOR_F& textColor);
31
34         const D2D1_RECT_F& textLocation();
35
38         const std::wstring& textString();
39
42         const float& textSize();
43
46         const Microsoft::WRL::ComPtr<ID2D1SolidColorBrush>& brush();
47
50         const Microsoft::WRL::ComPtr<IDWriteTextFormat>& format();
51
54         const D2D1_COLOR_F textColor();
55
58         void changeTextSize(const Microsoft::WRL::ComPtr<IDWriteFactory>& mDirectWriteFactory, float
      textSize);
59
62         void changeTextColor(const D2D1_COLOR_F& textColor);
63
66         void changeTextString(const std::wstring& textString);
67
70         void changeTextLocation(const D2D1_RECT_F& textLocation);
71
72     private:
73
74         D2D1_RECT_F mTextLocation;
75         std::wstring mText;
76         float mTextSize;
77         D2D1_COLOR_F mTextColor;
78
79         Microsoft::WRL::ComPtr<ID2D1SolidColorBrush> mDirect2DBrush;
80         Microsoft::WRL::ComPtr<IDWriteTextFormat> mDirectWriteFormat;
81     };
82 }
```

6.17 FATime.h File Reference

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

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

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 #include <iostream>
9
13 namespace FATime
14 {
15     class Time
16     {
17     public:
21         Time();
22
25         void Tick();
26
29         float DeltaTime() const;
30
33         void Reset();
34
37         void Stop();
38
41         void Start();
42
45         float TotalTime() const;
46
47     private:
48         __int64 mCurrTime; //holds current time stamp ti
49         __int64 mPrevTime; //holds previous time stamp ti-1
50         __int64 mStopTime; //holds the time we stopped the game/animation
51         __int64 mPausedTime; //holds how long the game/animation was paused for
52         __int64 mBaseTime; //holds the time we started / resetted
53
54         double mSecondsPerCount;
55         double mDeltaTime; //time elapsed btw frames change in t = ti - ti-1
56
57         bool mStopped; //flag to indicate if the game/animation is paused or not
58
59     };
60 }
```

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
26         HWND windowHandle() const;
27
30         unsigned int width();
31
34         unsigned int height();
35
38         bool& minimized();
39
42         bool& maximized();
43
46         bool& resizing();
47
48         void registerWindowClass(const WNDCLASSEX& windowClass);
49
52         void createWindow(const HINSTANCE& hInstance, const std::wstring& windowName, unsigned int width,
      unsigned int height);
53
56         void showWindow();
57
58     private:
59         HWND mWindowHandle;
60         WNDCLASSEX mWindowClass;
61         unsigned int mWidth;
62         unsigned int mHeight;
63         bool mMinimized;
64         bool mMaximized;
65         bool mResizing;
66     };
67 }
```

Index