

Farouq Adepetu's DirectX 12 Rendering Engine

Generated by Doxygen 1.9.4

| | |
|--|----------|
| 1 Namespace Index | 1 |
| 1.1 Namespace List | 1 |
| 2 Class Index | 3 |
| 2.1 Class List | 3 |
| 3 File Index | 5 |
| 3.1 File List | 5 |
| 4 Namespace Documentation | 7 |
| 4.1 FACamera Namespace Reference | 7 |
| 4.1.1 Detailed Description | 7 |
| 4.2 FARender Namespace Reference | 7 |
| 4.2.1 Detailed Description | 8 |
| 4.2.2 Function Documentation | 8 |
| 4.2.2.1 nextFrame() | 8 |
| 4.3 FAWindow Namespace Reference | 8 |
| 4.3.1 Detailed Description | 8 |
| 5 Class Documentation | 9 |
| 5.1 FACamera::Camera Class Reference | 9 |
| 5.1.1 Detailed Description | 11 |
| 5.1.2 Constructor & Destructor Documentation | 11 |
| 5.1.2.1 Camera() [1/2] | 11 |
| 5.1.2.2 Camera() [2/2] | 11 |
| 5.1.3 Member Function Documentation | 12 |
| 5.1.3.1 aspect() [1/2] | 12 |
| 5.1.3.2 aspect() [2/2] | 12 |
| 5.1.3.3 backward() | 12 |
| 5.1.3.4 cameraPosition() [1/2] | 12 |
| 5.1.3.5 cameraPosition() [2/2] | 12 |
| 5.1.3.6 cameraVelocity() [1/2] | 13 |
| 5.1.3.7 cameraVelocity() [2/2] | 13 |
| 5.1.3.8 down() | 13 |
| 5.1.3.9 foward() | 13 |
| 5.1.3.10 keyboardInput() | 13 |
| 5.1.3.11 left() | 13 |
| 5.1.3.12 lookAt() | 14 |
| 5.1.3.13 mouseInput() | 14 |
| 5.1.3.14 perspectiveProjectionTransformationMatrix() | 14 |
| 5.1.3.15 right() | 14 |
| 5.1.3.16 rotateCameraLeftRight() | 14 |
| 5.1.3.17 rotateCameraUpDown() | 15 |
| 5.1.3.18 rotateVelocity() [1/2] | 15 |

| | |
|--|----|
| 5.1.3.19 rotateVelocity() [2/2] | 15 |
| 5.1.3.20 up() | 15 |
| 5.1.3.21 updatePerspectiveProjectionTransformationMatrix() | 15 |
| 5.1.3.22 updateViewPerspectiveProjectionTransformationMatrix() | 15 |
| 5.1.3.23 updateViewTransformationMatrix() | 16 |
| 5.1.3.24 vFov() [1/2] | 16 |
| 5.1.3.25 vFov() [2/2] | 16 |
| 5.1.3.26 viewPerspectiveProjectionTransformationMatrix() | 16 |
| 5.1.3.27 viewTransformationMatrix() | 16 |
| 5.1.3.28 x() | 16 |
| 5.1.3.29 y() | 17 |
| 5.1.3.30 z() | 17 |
| 5.1.3.31 zfar() [1/2] | 17 |
| 5.1.3.32 zfar() [2/2] | 17 |
| 5.1.3.33 znear() [1/2] | 17 |
| 5.1.3.34 znear() [2/2] | 17 |
| 5.2 FAREnder::ConstantBuffer Class Reference | 18 |
| 5.2.1 Detailed Description | 18 |
| 5.2.2 Constructor & Destructor Documentation | 18 |
| 5.2.2.1 ~ConstantBuffer() | 18 |
| 5.2.3 Member Function Documentation | 18 |
| 5.2.3.1 constantBuffer() [1/2] | 19 |
| 5.2.3.2 constantBuffer() [2/2] | 19 |
| 5.2.3.3 copyData() | 19 |
| 5.2.3.4 createConstantBuffer() | 19 |
| 5.2.3.5 createConstantBufferView() | 19 |
| 5.2.3.6 mappedData() | 20 |
| 5.3 FAREnder::DeviceResources Class Reference | 20 |
| 5.3.1 Detailed Description | 22 |
| 5.3.2 Constructor & Destructor Documentation | 22 |
| 5.3.2.1 ~DeviceResources() | 22 |
| 5.3.3 Member Function Documentation | 22 |
| 5.3.3.1 backBufferFormat() | 22 |
| 5.3.3.2 commandAllocator() | 22 |
| 5.3.3.3 commandList() | 23 |
| 5.3.3.4 commandQueue() | 23 |
| 5.3.3.5 currentBackBuffer() | 23 |
| 5.3.3.6 currentFenceValue() [1/2] | 23 |
| 5.3.3.7 currentFenceValue() [2/2] | 23 |
| 5.3.3.8 depthStencilBuffer() | 23 |
| 5.3.3.9 depthStencilFormat() | 24 |
| 5.3.3.10 device() | 24 |

| | |
|--|----|
| 5.3.3.11 device2DContext() | 24 |
| 5.3.3.12 directWriteFactory() | 24 |
| 5.3.3.13 dsvDescriptorHeap() | 24 |
| 5.3.3.14 dsvDescriptorSize() | 24 |
| 5.3.3.15 execute() | 25 |
| 5.3.3.16 flushCommandQueue() | 25 |
| 5.3.3.17 initializeDirect3D() | 25 |
| 5.3.3.18 isMSAAEnabled() [1/2] | 25 |
| 5.3.3.19 isMSAAEnabled() [2/2] | 25 |
| 5.3.3.20 numOfSwapChainBuffers() | 26 |
| 5.3.3.21 present() | 26 |
| 5.3.3.22 resetCommandAllocator() | 26 |
| 5.3.3.23 resetCommandList() | 26 |
| 5.3.3.24 resetTextBuffers() | 26 |
| 5.3.3.25 resize() | 26 |
| 5.3.3.26 rtvDescriptorHeap() | 27 |
| 5.3.3.27 rtvDescriptorSize() | 27 |
| 5.3.3.28 sampleCount() [1/2] | 27 |
| 5.3.3.29 sampleCount() [2/2] | 27 |
| 5.3.3.30 scissor() | 27 |
| 5.3.3.31 signal() | 27 |
| 5.3.3.32 swapChain() | 28 |
| 5.3.3.33 swapChainBuffers() | 28 |
| 5.3.3.34 textDraw() | 28 |
| 5.3.3.35 textResize() | 28 |
| 5.3.3.36 updateCurrentFrameFenceValue() | 28 |
| 5.3.3.37 viewport() | 29 |
| 5.3.3.38 waitForGPU() | 29 |
| 5.4 DirectXException Class Reference | 29 |
| 5.5 FAREnder::DrawArguments Struct Reference | 29 |
| 5.5.1 Detailed Description | 30 |
| 5.6 FAREnder::IndexBuffer Class Reference | 30 |
| 5.6.1 Detailed Description | 30 |
| 5.6.2 Member Function Documentation | 30 |
| 5.6.2.1 createIndexBuffer() | 30 |
| 5.6.2.2 createIndexBufferView() | 31 |
| 5.6.2.3 indexBufferView() | 31 |
| 5.7 FAREnder::RenderScene Class Reference | 31 |
| 5.7.1 Detailed Description | 32 |
| 5.7.2 Member Function Documentation | 32 |
| 5.7.2.1 afterDraw() | 33 |
| 5.7.2.2 beforeDraw() | 33 |

| | |
|--|----|
| 5.7.2.3 cbvHeap() | 33 |
| 5.7.2.4 cbvHeapRootParameter() | 33 |
| 5.7.2.5 cbvSize() | 33 |
| 5.7.2.6 createCBVHeap() | 34 |
| 5.7.2.7 createConstantBuffer() | 34 |
| 5.7.2.8 createConstantBufferView() | 34 |
| 5.7.2.9 createIndexBuffer() | 34 |
| 5.7.2.10 drawObjects() | 35 |
| 5.8 FAREnder::Text Class Reference | 35 |
| 5.8.1 Detailed Description | 36 |
| 5.8.2 Member Function Documentation | 36 |
| 5.8.2.1 brush() | 36 |
| 5.8.2.2 changeTextColor() | 36 |
| 5.8.2.3 changeTextLocation() | 36 |
| 5.8.2.4 changeTextSize() | 37 |
| 5.8.2.5 changeTextString() | 37 |
| 5.8.2.6 format() | 37 |
| 5.8.2.7 initialize() | 37 |
| 5.8.2.8 textColor() | 37 |
| 5.8.2.9 textLocation() | 38 |
| 5.8.2.10 textSize() | 38 |
| 5.8.2.11 textString() | 38 |
| 5.9 FATime::Time Class Reference | 38 |
| 5.9.1 Constructor & Destructor Documentation | 38 |
| 5.9.1.1 Time() | 39 |
| 5.9.2 Member Function Documentation | 39 |
| 5.9.2.1 DeltaTime() | 39 |
| 5.9.2.2 Reset() | 39 |
| 5.9.2.3 Start() | 39 |
| 5.9.2.4 Stop() | 39 |
| 5.9.2.5 Tick() | 39 |
| 5.9.2.6 TotalTime() | 40 |
| 5.10 Time Class Reference | 40 |
| 5.10.1 Detailed Description | 40 |
| 5.11 FAREnder::VertexBuffer Class Reference | 40 |
| 5.11.1 Detailed Description | 40 |
| 5.11.2 Member Function Documentation | 41 |
| 5.11.2.1 createVertexBuffer() | 41 |
| 5.11.2.2 createVertexBufferView() | 41 |
| 5.11.2.3 vertexBufferView() | 41 |
| 5.12 FAWindow::Window Class Reference | 41 |
| 5.12.1 Detailed Description | 42 |

| | |
|---|-----------|
| 5.12.2 Member Function Documentation | 42 |
| 5.12.2.1 createWindow() | 42 |
| 5.12.2.2 height() | 42 |
| 5.12.2.3 maximized() | 43 |
| 5.12.2.4 minimized() | 43 |
| 5.12.2.5 resizing() | 43 |
| 5.12.2.6 showWindow() | 43 |
| 5.12.2.7 width() | 43 |
| 5.12.2.8 windowHandle() | 43 |
| 6 File Documentation | 45 |
| 6.1 Direct3DLink.h | 45 |
| 6.2 FABuffer.h File Reference | 45 |
| 6.2.1 Detailed Description | 46 |
| 6.3 FABuffer.h | 46 |
| 6.4 FACamera.h File Reference | 47 |
| 6.4.1 Detailed Description | 47 |
| 6.4.2 Typedef Documentation | 47 |
| 6.4.2.1 vec2 | 48 |
| 6.5 FACamera.h | 48 |
| 6.6 FADeviceResources.h File Reference | 49 |
| 6.6.1 Detailed Description | 50 |
| 6.7 FADeviceResources.h | 50 |
| 6.8 FADirectXException.h | 52 |
| 6.9 FARenderingUtility.h File Reference | 53 |
| 6.9.1 Detailed Description | 53 |
| 6.10 FARenderingUtility.h | 53 |
| 6.11 FARenderScene.h File Reference | 54 |
| 6.11.1 Detailed Description | 54 |
| 6.12 FARenderScene.h | 54 |
| 6.13 FAText.h File Reference | 56 |
| 6.13.1 Detailed Description | 57 |
| 6.14 FAText.h | 57 |
| 6.15 FATime.h File Reference | 58 |
| 6.15.1 Detailed Description | 58 |
| 6.16 FATime.h | 58 |
| 6.17 FAWindow.h File Reference | 58 |
| 6.17.1 Detailed Description | 59 |
| 6.18 FAWindow.h | 59 |
| Index | 61 |

Chapter 1

Namespace Index

1.1 Namespace List

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

| | | |
|---|-------|---|
| FACamera | | |
| Has Camera class | | 7 |
| FARender | | |
| The namespace has utility functions and structs, VertexBuffer , IndexBuffer , ConstantBuffer , DeviceResources , RenderScene and Text classes | | 7 |
| FAWindow | | |
| Has Window class | | 8 |

Chapter 2

Class Index

2.1 Class List

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

[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.

9

[FARender::ConstantBuffer](#)

This class stores constant data in a Direct3D 12 upload buffers 18

[FARender::DeviceResources](#)

A wrapper for a Direct3D 12 device, swapchain, depth buffer, MSAA buffers and command objects 20

[DirectXException](#) 29

[FARender::DrawArguments](#)

Has all the data that are used as parameters to draw an object 29

[FARender::IndexBuffer](#)

This class stores indices in a Direct3D 12 default buffer 30

[FARender::RenderScene](#)

This class is used to render a scene using Direct3D 12 API 31

[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 35

[FATime::Time](#) 38

[Time](#)

This class is used to get the time between each frame. You can stop start, reset and get the total time 40

[FARender::VertexBuffer](#)

This class stores vertices in a Direct3D 12 default buffer 40

[FAWindow::Window](#)

The window class is used to make a [Window](#) using Windows API 41

Chapter 3

File Index

3.1 File List

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

| | |
|--|----|
| Direct3DLink.h | ?? |
| FABuffer.h | |
| File has classes VertexBuffer, IndexBuffer and ConstantBuffer under namespace FARender | 45 |
| FACamera.h | |
| File that has namespace FACamera . Withn the namespace is the class Camera | 47 |
| FADeviceResources.h | |
| File has class DeviceResources under namespace FARender | 49 |
| FADirectXException.h | ?? |
| FARenderingUtility.h | |
| File has static variables numFrames and current frame, function nextFrame() and struct Draw↔ Arguments under the namespace FARender | 53 |
| FARenderScene.h | |
| File has class RenderScene under namespace FARender | 54 |
| FAText.h | |
| File has class Text under namespace FARender | 56 |
| FATime.h | |
| File that has namespace FATime. Withn the namespace is the class Time | 58 |
| FAWindow.h | |
| File that has namespace FAWindow . Withn the namespace is the class Window | 58 |

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 frustum 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 frustum 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 frustum 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 frustum.

5.1.3.32 zfar() [2/2]

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

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

5.1.3.33 znear() [1/2]

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

Returns a reference to the near value of the frustum.

5.1.3.34 znear() [2/2]

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

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

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

- [FACamera.h](#)

5.2 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.

5.2.1 Detailed Description

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

5.2.2 Constructor & Destructor Documentation

5.2.2.1 ~ConstantBuffer()

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

Unmaps the pointer to the constant buffer.

5.2.3 Member Function Documentation

5.2.3.1 constantBuffer() [1/2]

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

Returns a reference to the constant buffer resource.

5.2.3.2 constantBuffer() [2/2]

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

Returns a constant reference to the constant buffer resource.

5.2.3.3 copyData()

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

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

5.2.3.4 createConstantBuffer()

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

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

5.2.3.5 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.

5.2.3.6 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](#)

5.3 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 object.
- const Microsoft::WRL::ComPtr< ID3D12CommandQueue > & **commandQueue** () const
Returns a constant reference to the ID3D12CommandQueue object.
- const Microsoft::WRL::ComPtr< ID3D12CommandAllocator > & **commandAllocator** () const
Returns a constant reference to the current ID3D12CommandAllocator object.
- const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & **commandList** () const
Returns a constant reference to the ID3D12GraphicsCommandList object.
- 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 object.
- `const D3D12_RECT & scissor () const`
Returns a constant reference to the D3D12_RECT scissor object.
- `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 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 function.
- void `textResize` (const HWND &handle)
Resizes the necessary text buffers. Call before calling the rendering resize function.
- void `textDraw` (`Text` *textToRender=NULLptr, UINT numText=0)
Renders the text.

5.3.1 Detailed Description

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

5.3.2 Constructor & Destructor Documentation

5.3.2.1 ~DeviceResources()

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

Flushes the command queue.

5.3.3 Member Function Documentation

5.3.3.1 backBufferFormat()

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

Returns a constant reference to the back buffer format.

5.3.3.2 commandAllocator()

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

Returns a constant reference to the current ID3D12CommandAllocator object.

5.3.3.3 commandList()

```
const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & FARender::DeviceResources::commandList ( ) const
```

Returns a constant reference to the ID3D12GraphicsCommandList object.

5.3.3.4 commandQueue()

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

Returns a constant reference to the ID3D12CommandQueue object.

5.3.3.5 currentBackBuffer()

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

Returns a constant reference to the current back buffer.

5.3.3.6 currentFenceValue() [1/2]

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

Returns a reference to the current fence value.

5.3.3.7 currentFenceValue() [2/2]

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

Returns a constant reference to the current fence value.

5.3.3.8 depthStencilBuffer()

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

Returns a constant reference to the depth stencil buffer.

5.3.3.9 depthStencilFormat()

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

Returns a constant reference to the depth stencil format.

5.3.3.10 device()

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

Returns a constant reference to the ID3D12Device object.

5.3.3.11 device2DContext()

```
const Microsoft::WRL::ComPtr< ID2D1DeviceContext > & FARender::DeviceResources::device2D↵  
Context ( )
```

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

5.3.3.12 directWriteFactory()

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

Returns a constant reference to the direct write factory.

5.3.3.13 dsvDescriptorHeap()

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

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

5.3.3.14 dsvDescriptorSize()

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

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

5.3.3.15 execute()

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

Executes the command list.

5.3.3.16 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.

5.3.3.17 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.

5.3.3.18 isMSAAEnabled() [1/2]

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

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

5.3.3.19 isMSAAEnabled() [2/2]

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

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

5.3.3.20 numOfSwapChainBuffers()

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

Returns a constant reference to the number of swap chains.

5.3.3.21 present()

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

Swaps the front and back buffers.

5.3.3.22 resetCommandAllocator()

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

Resets command allocator to allow reuse of the memory.

5.3.3.23 resetCommandList()

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

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

5.3.3.24 resetTextBuffers()

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

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

5.3.3.25 resize()

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

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

5.3.3.26 rtvDescriptorHeap()

```
const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & FARender::DeviceResources::rtvDescriptorHeap ( ) const
```

Returns a constant reference to the render target descriptor heap.

5.3.3.27 rtvDescriptorSize()

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

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

5.3.3.28 sampleCount() [1/2]

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

Returns a reference to the sample count.

5.3.3.29 sampleCount() [2/2]

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

Returns a constant reference to the sample count.

5.3.3.30 scissor()

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

Returns a constant reference to the D3D12_RECT scissor object.

5.3.3.31 signal()

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

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

5.3.3.32 swapChain()

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

Returns a constant reference to the IDXGISwapChain1 object.

5.3.3.33 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.

5.3.3.34 textDraw()

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

Renders the text.

5.3.3.35 textResize()

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

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

5.3.3.36 updateCurrentFrameFenceValue()

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

Updates the current frames fence value.

5.3.3.37 viewport()

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

Returns a constant reference to the D3D12_VIEWPORT object.

5.3.3.38 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](#)

5.4 DirectXException Class Reference

Public Member Functions

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

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

- [FADirectXException.h](#)

5.5 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**

5.5.1 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](#)

5.6 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.

5.6.1 Detailed Description

This class stores indices in a Direct3D 12 default buffer.

5.6.2 Member Function Documentation

5.6.2.1 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.

5.6.2.2 createIndexBufferView()

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

Creates the vertex buffer view and stores it.

5.6.2.3 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](#)

5.7 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 each 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.

5.7.1 Detailed Description

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

5.7.2 Member Function Documentation

5.7.2.1 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.

5.7.2.2 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.

5.7.2.3 cbvHeap()

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

Returns a constant reference to the CBV descriptor heap.

5.7.2.4 cbvHeapRootParameter()

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

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

5.7.2.5 cbvSize()

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

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

5.7.2.6 createCBVHeap()

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

Creates the CBV heap.

5.7.2.7 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.

5.7.2.8 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 each frame and stores it in the CBV heap.

5.7.2.9 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()`.

5.7.2.10 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](#)

5.8 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.

5.8.1 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.

5.8.2 Member Function Documentation

5.8.2.1 `brush()`

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

Returns a constant reference to the color brush.

5.8.2.2 `changeTextColor()`

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

Changes the text color to the specified color.

5.8.2.3 `changeTextLocation()`

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

Changes the text location to the specified location.

5.8.2.4 changeTextSize()

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

Changes the text size to the specified size.

5.8.2.5 changeTextString()

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

Changes the text string to the specified string.

5.8.2.6 format()

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

Returns a constant reference to the format of the text.

5.8.2.7 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.

5.8.2.8 textColor()

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

Returns a constant reference to the text color.

5.8.2.9 textLocation()

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

Returns a constant reference to the text location.

5.8.2.10 textSize()

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

Returns a constant reference to the text size.

5.8.2.11 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](#)

5.9 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.

5.9.1 Constructor & Destructor Documentation

5.9.1.1 Time()

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

Default Constructor. Gets and stores the seconds per count.

5.9.2 Member Function Documentation

5.9.2.1 DeltaTime()

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

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

5.9.2.2 Reset()

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

Resets all time variables.

5.9.2.3 Start()

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

Starts the timer.

5.9.2.4 Stop()

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

Stops the timer.

5.9.2.5 Tick()

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

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

5.9.2.6 TotalTime()

```
float FTime::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:

- [FTime.h](#)

5.10 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 "FTime.h"
```

5.10.1 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:

- [FTime.h](#)

5.11 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.

5.11.1 Detailed Description

This class stores vertices in a Direct3D 12 default buffer.

5.11.2 Member Function Documentation

5.11.2.1 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.

5.11.2.2 createVertexBufferView()

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

Creates the vertex buffer view and stores it.

5.11.2.3 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](#)

5.12 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.

5.12.1 Detailed Description

The window class is used to make a [Window](#) using Windows API.

5.12.2 Member Function Documentation

5.12.2.1 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.

5.12.2.2 height()

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

Returns the height of the window.

5.12.2.3 maximized()

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

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

5.12.2.4 minimized()

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

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

5.12.2.5 resizing()

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

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

5.12.2.6 showWindow()

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

Updates and shows the window.

5.12.2.7 width()

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

Returns the width of the window.

5.12.2.8 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
3  #include <wrl.h>
4  #include <d3d12.h>
5
6  namespace FARender
7  {
8      class VertexBuffer
9      {
10     public:
11         VertexBuffer() = default;
12         VertexBuffer(const VertexBuffer&) = delete;
13         VertexBuffer& operator=(const VertexBuffer&) = delete;
14
15         const D3D12_VERTEX_BUFFER_VIEW& vertexBufferView();
16
17         void createVertexBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
18             const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const void* data, UINT
19             numBytes);
20
21         void createVertexBufferView(UINT numBytes, UINT stride);
22
23     private:
24         Microsoft::WRL::ComPtr<ID3D12Resource> mVertexDefaultBuffer;
25         Microsoft::WRL::ComPtr<ID3D12Resource> mVertexUploadBuffer;
26         D3D12_VERTEX_BUFFER_VIEW mVertexBufferView;
27     };
28
29     class IndexBuffer
30     {
31     public:
32         IndexBuffer() = default;
33         IndexBuffer(const IndexBuffer&) = delete;
34         IndexBuffer& operator=(const IndexBuffer&) = delete;
35
36         const D3D12_INDEX_BUFFER_VIEW& indexBufferView();
37
38         void createIndexBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
39             const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const void* data, UINT
40             numBytes);
41
42         void createIndexBufferView(UINT numBytes, DXGI_FORMAT format);
43
44     private:
45         Microsoft::WRL::ComPtr<ID3D12Resource> mIndexDefaultBuffer;
46         Microsoft::WRL::ComPtr<ID3D12Resource> mIndexUploadBuffer;
47         D3D12_INDEX_BUFFER_VIEW mIndexBufferView;
48     };
49
50     class ConstantBuffer
51     {
52     public:
53         ConstantBuffer() = default;
54
55         ConstantBuffer(const ConstantBuffer&) = delete;
56         ConstantBuffer& operator=(const ConstantBuffer&) = delete;
57
58         ~ConstantBuffer();
59
60         Microsoft::WRL::ComPtr<ID3D12Resource>& constantBuffer();
61
62         const Microsoft::WRL::ComPtr<ID3D12Resource>& constantBuffer() const;
63
64         BYTE*& mappedData();
65
66         void createConstantBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device, const UINT&
67             numBytes);
68
69         void createConstantBufferView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
70             const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& cbvHeap, UINT cbvSize, UINT
71             cBufferIndex,
72             UINT cbvHeapIndex, UINT numBytes);

```

```

111
115         void copyData(UINT index, UINT byteSize, const void* data, const UINT64& numBytes);
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 FADeviceResources.h File Reference

File has class DeviceResources under namespace [FARender](#).

```

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

6.6.1 Detailed Description

File has class [DeviceResources](#) under namespace [FARender](#).

6.7 FADeviceResources.h

[Go to the documentation of this file.](#)

```

1 #pragma once
2
3 #include <wrl.h>
4 #include <d3d12.h>
5 #include <dxgi1_4.h>
6 #include <vector>
7 #include "FARenderingUtility.h"
8 #include "FAText.h"
9
10 namespace FAScene
11 {
12     class DeviceResources
13     {
14     public:
15
16         DeviceResources() = default;
17
18         DeviceResources(const DeviceResources&) = delete;
19         DeviceResources& operator=(const DeviceResources&) = delete;
20
21         ~DeviceResources();
22
23         const Microsoft::WRL::ComPtr<ID3D12Device>& device() const;
24
25         const Microsoft::WRL::ComPtr<ID3D12CommandQueue>& commandQueue() const;
26
27         const Microsoft::WRL::ComPtr<ID3D12CommandAllocator>& commandAllocator() const;
28
29         const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList() const;
30
31         const DXGI_FORMAT& backBufferFormat() const;
32
33         const UINT numOfSwapChainBuffers() const;
34
35         const Microsoft::WRL::ComPtr<IDXGISwapChain1>& swapChain() const;
36
37         const UINT& rtvDescriptorSize() const;
38
39         const UINT& dsvDescriptorSize() const;
40
41         const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvDescriptorHeap() const;
42         const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvDescriptorHeap() const;
43
44         const UINT& currentBackBuffer() const;
45
46         const Microsoft::WRL::ComPtr<ID3D12Resource>* swapChainBuffers() const;
47
48         const Microsoft::WRL::ComPtr<ID3D12Resource>& depthStencilBuffer() const;
49
50         const DXGI_FORMAT& depthStencilFormat() const;
51
52         const D3D12_VIEWPORT& viewport() const;
53
54         const D3D12_RECT& scissor() const;
55
56         bool& isMSAAEnabled();
57
58         const bool& isMSAAEnabled() const;
59
60         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 Transitions 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 }

```

6.8 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 DirectException(hr, L#x, filename, __LINE__); }
39 }
40 #endif

```

6.9 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.

6.9.1 Detailed Description

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

6.10 FARenderingUtility.h

[Go to the documentation of this file.](#)

```

1 #pragma once
2
3 #include <d3d12.h>
4
5 namespace FAScene
6 {
7     static const UINT numFrames{ 3 };
8     static UINT currentFrame{ 0 };
9
10    void nextFrame();
11
12    struct DrawArguments
13    {
14        UINT indexCount;
15        UINT locationFirstIndex;
16        INT indexOfFirstVertex;
17        INT indexOfConstantData;
18    };
19 }

```

6.11 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.

6.11.1 Detailed Description

File has class `RenderScene` under namespace `FARender`.

6.12 FARenderScene.h

[Go to the documentation of this file.](#)

```
1 #pragma once
2
3
4
5
6
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 {
18
19     class RenderScene
20     {
21     public:
22
23         RenderScene() = default;
24
25         RenderScene(const RenderScene&) = delete;
26         RenderScene& operator=(const RenderScene&) = delete;
27
28         /*@brief Returns a constant reference to the shader with the specified name.
29          * Throws an out_of_range exception if the shader does not exist.
30          */
31         const Microsoft::WRL::ComPtr<ID3DBlob>& shader(const std::wstring& name) const;
32
33     };
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
65     const UINT& cbvSize() const;
66
67     const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& cbvHeap() const;
68
69     const D3D12_ROOT_PARAMETER& cbvHeapRootParameter() const;
70
71     /*@brief Returns a constant reference to the draw argument with the specified name in the
    specified group.
72 * Throws an out_of_range exception if the draw argument does not exist.
73 */
74     const DrawArguments& drawArgument(const std::wstring& groupName, const std::wstring& objectName)
    const;
75
76     /*@brief Loads a shader's bytecode and stores it with the specified name.
77 */
78     void loadShader(const std::wstring& filename, const std::wstring& name);
79
80     /*@brief Stores an array of input element descriptions with the specified name.
81 */
82     void storeInputElementDescriptions(const std::wstring& name, const
    std::vector<D3D12_INPUT_ELEMENT_DESC>& inputElementLayout);
83
84     /*@brief Stores an array of input element descriptions with the specified name.
85 */
86     void storeInputElementDescriptions(const std::wstring& name, const D3D12_INPUT_ELEMENT_DESC*
    inputElementLayout, UINT numElements);
87
88     /*@brief Creates a rasterization description and stores it with the specified name.
89 */
90     void createRasterizationState(D3D12_FILL_MODE fillMode, BOOL enableMultisample, const
    std::wstring& name);
91
92     /*@brief Creates a PSO and stores it with the specified name.
93 */
94     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);
95
96     /*@brief Creates a root signature and stores it with the specified name.
97 */
98     void createRootSignature(const Microsoft::WRL::ComPtr<ID3D12Device>& device, const std::wstring&
    name,
    const D3D12_ROOT_PARAMETER* rootParameters, UINT numParameters);
99
100    /*@brief Stores a DrawArgument object with the specified name in the specified group.
101 */
102    void storeDrawArgument(const std::wstring& groupName, const std::wstring& objectName, const
    DrawArguments& drawArgs);
103
104    /*@brief Creates a vertex buffer with the specified name and stores all of given data in the
    vertex buffer.
105 * Execute commands and the flush command queue after calling createVertexBuffer() and
    createIndexBuffer().
106 */

```

```

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
125     void createIndexBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
126         const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList,
127         const std::wstring& ibName, const void* data, UINT numBytes);
128
129     /*@brief Creates an index buffer view for the index buffer with the specified name.
130     */
131     void createIndexBufferView(const std::wstring& ibName, UINT numBytes, DXGI_FORMAT format);
132
133     void createCBVHeap(const Microsoft::WRL::ComPtr<ID3D12Device>& device, UINT numDescriptors, UINT
134         shaderRegister);
135
136     void createConstantBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device, const
137         std::wstring& name,
138         const UINT& numBytes);
139
140     void createConstantBufferView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
141         const std::wstring& name, UINT index, UINT numBytes);
142
143     void beforeDraw(DeviceResources& deviceResource);
144
145     void drawObjects(const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList,
146         const std::wstring& drawArgsGroupName, const std::wstring& vbName, const std::wstring&
147         ibName,
148         const std::wstring& psoName, const std::wstring& rootSignatureName,
149         const D3D_PRIMITIVE_TOPOLOGY& primitive);
150
151     void afterDraw(DeviceResources& deviceResource, Text* textToRender = nullptr, UINT numText = 0);
152
153     private:
154         //Stores all of the shaders and input element descriptions for this scene.
155         std::unordered_map<std::wstring, Microsoft::WRL::ComPtr<ID3DBlob>> mShaders;
156         std::unordered_map<std::wstring, std::vector<D3D12_INPUT_ELEMENT_DESC>>
157         mInputElementDescriptions;
158
159         //Stores all of the rasterization states, PSOs, and root signatures for this scene.
160         std::unordered_map<std::wstring, D3D12_RASTERIZER_DESC> mRasterizationStates;
161         std::unordered_map<std::wstring, Microsoft::WRL::ComPtr<ID3D12PipelineState>> mPSOs;
162         std::unordered_map<std::wstring, Microsoft::WRL::ComPtr<ID3D12RootSignature>> mRootSignatures;
163
164         //Each scene gets one CBV heap.
165         Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mCBVHeap;
166         UINT mCBVSize;
167         D3D12_DESCRIPTOR_RANGE mCBVHeapDescription{};
168         D3D12_ROOT_PARAMETER mCBVHeapRootParameter;
169
170         //Stores all of the constant buffers this scene uses. We can't update a constant buffer until
171         the GPU
172         //is done executing all the commands that reference it, so each frame needs its own constant
173         buffers.
174         std::unordered_map<std::wstring, ConstantBuffer> mConstantBuffers[numFrames];
175
176         //Groups all of the objects draw arguments that are in the same vertex buffer and index buffer,
177         //and uses the same shaders, rasterization states, PSO, and root signatures.
178         std::unordered_map<std::wstring, std::unordered_map<std::wstring, DrawArguments>> mDrawArgs;
179
180         //Stores all of the vertex buffers and index buffers for this scene.
181         std::unordered_map<std::wstring, VertexBuffer> mVertexBuffers;
182         std::unordered_map<std::wstring, IndexBuffer> mIndexBuffers;
183     };
184 }

```

6.13 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.

6.13.1 Detailed Description

File has class Text under namespace [FARender](#).

6.14 FAText.h

[Go to the documentation of this file.](#)

```

1  #pragma once
2
3  #include <wrl.h>
4  #include <d3d11.h>
5  #include <d3d11on12.h>
6  #include <d2d1_3.h>
7  #include <dwrite.h>
8  #include <string>
9
10 namespace FArender
11 {
12     class Text
13     {
14     public:
15
16         Text() = default;
17
18         void initialize(const Microsoft::WRL::ComPtr<ID2D1DeviceContext>& deviceContext,
19             const Microsoft::WRL::ComPtr<IDWriteFactory>& writeFactory,
20             const D2D1_RECT_F& textLocation, const std::wstring& textString, float textSize, const
21             D2D1_COLOR_F& textColor);
22
23         const D2D1_RECT_F& textLocation();
24
25         const std::wstring& textString();
26
27         const float& textSize();
28
29         const Microsoft::WRL::ComPtr<ID2D1SolidColorBrush>& brush();
30
31         const Microsoft::WRL::ComPtr<IDWriteTextFormat>& format();
32
33         const D2D1_COLOR_F& textColor();
34
35         void changeTextSize(const Microsoft::WRL::ComPtr<IDWriteFactory>& mDirectWriteFactory, float
36             textSize);
37
38         void changeTextColor(const D2D1_COLOR_F& textColor);
39
40         void changeTextString(const std::wstring& textString);
41
42         void changeTextLocation(const D2D1_RECT_F& textLocation);
43
44     private:
45
46         D2D1_RECT_F mTextLocation;
47         std::wstring mText;
48         float mTextSize;
49         D2D1_COLOR_F mTextColor;
50
51         Microsoft::WRL::ComPtr<ID2D1SolidColorBrush> mDirect2DBrush;
52         Microsoft::WRL::ComPtr<IDWriteTextFormat> mDirectWriteFormat;
53     };
54 }
```

6.15 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](#)

6.15.1 Detailed Description

File that has namespace FATime. Withn the namespace is the class [Time](#).

6.16 FATime.h

[Go to the documentation of this file.](#)

```
1 #pragma once
2
3 #include <Windows.h>
4 #include <iostream>
5
6 namespace FATime
7 {
8     class Time
9     {
10     public:
11         Time();
12
13         void Tick();
14
15         float DeltaTime() const;
16
17         void Reset();
18
19         void Stop();
20
21         void Start();
22
23         float TotalTime() const;
24
25     private:
26         __int64 mCurrTime; //holds current time stamp ti
27         __int64 mPrevTime; //holds previous time stamp ti-1
28         __int64 mStopTime; //holds the time we stopped the game/animation
29         __int64 mPausedTime; //holds how long the game/animation was paused for
30         __int64 mBaseTime; //holds the time we started / resetted
31
32         double mSecondsPerCount;
33         double mDeltaTime; //time elapsed btw frames change in t = ti - ti-1
34
35         bool mStopped; //flag to indicate if the game/animation is paused or not
36     };
37 }
```

6.17 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.

6.17.1 Detailed Description

File that has namespace [FAWindow](#). Withn the namespace is the class Window.

6.18 FAWindow.h

[Go to the documentation of this file.](#)

```
1 #pragma once
2
3 #include <Windows.h>
4 #include <string>
5 #include <stdexcept>
6
7 namespace FAWindow
8 {
9     class Window
10     {
11     public:
12         Window();
13
14         HWND windowHandle() const;
15
16         unsigned int width();
17
18         unsigned int height();
19
20         bool& minimized();
21
22         bool& maximized();
23
24         bool& resizing();
25
26         void registerWindowClass(const WNDCLASSEX& windowClass);
27
28         void createWindow(const HINSTANCE& hInstance, const std::wstring& windowName, unsigned int width,
29             unsigned int height);
30
31         void showWindow();
32
33     private:
34         HWND mWindowHandle;
35         WNDCLASSEX mWindowClass;
36         unsigned int mWidth;
37         unsigned int mHeight;
38         bool mMinimized;
39         bool mMaximized;
40         bool mResizing;
41     };
42 }
```


Index

- ~ConstantBuffer
 - FARender::ConstantBuffer, [18](#)
- ~DeviceResources
 - FARender::DeviceResources, [22](#)
- afterDraw
 - FARender::RenderScene, [32](#)
- aspect
 - FACamera::Camera, [12](#)
- backBufferFormat
 - FARender::DeviceResources, [22](#)
- backward
 - FACamera::Camera, [12](#)
- beforeDraw
 - FARender::RenderScene, [33](#)
- brush
 - FARender::Text, [36](#)
- Camera
 - FACamera::Camera, [11](#)
- cameraPosition
 - FACamera::Camera, [12](#)
- cameraVelocity
 - FACamera::Camera, [12](#), [13](#)
- cbvHeap
 - FARender::RenderScene, [33](#)
- cbvHeapRootParameter
 - FARender::RenderScene, [33](#)
- cbvSize
 - FARender::RenderScene, [33](#)
- changeTextColor
 - FARender::Text, [36](#)
- changeTextLocation
 - FARender::Text, [36](#)
- changeTextSize
 - FARender::Text, [36](#)
- changeTextString
 - FARender::Text, [37](#)
- commandAllocator
 - FARender::DeviceResources, [22](#)
- commandList
 - FARender::DeviceResources, [22](#)
- commandQueue
 - FARender::DeviceResources, [23](#)
- constantBuffer
 - FARender::ConstantBuffer, [18](#), [19](#)
- copyData
 - FARender::ConstantBuffer, [19](#)
- createCBVHeap
 - FARender::RenderScene, [33](#)
- createConstantBuffer
 - FARender::ConstantBuffer, [19](#)
 - FARender::RenderScene, [34](#)
- createConstantBufferView
 - FARender::ConstantBuffer, [19](#)
 - FARender::RenderScene, [34](#)
- createIndexBuffer
 - FARender::IndexBuffer, [30](#)
 - FARender::RenderScene, [34](#)
- createIndexBufferView
 - FARender::IndexBuffer, [30](#)
- createVertexBuffer
 - FARender::VertexBuffer, [41](#)
- createVertexBufferView
 - FARender::VertexBuffer, [41](#)
- createWindow
 - FAWindow::Window, [42](#)
- currentBackBuffer
 - FARender::DeviceResources, [23](#)
- currentFenceValue
 - FARender::DeviceResources, [23](#)
- DeltaTime
 - FATime::Time, [39](#)
- depthStencilBuffer
 - FARender::DeviceResources, [23](#)
- depthStencilFormat
 - FARender::DeviceResources, [23](#)
- device
 - FARender::DeviceResources, [24](#)
- device2DContext
 - FARender::DeviceResources, [24](#)
- directWriteFactory
 - FARender::DeviceResources, [24](#)
- DirectXException, [29](#)
- down
 - FACamera::Camera, [13](#)
- drawObjects
 - FARender::RenderScene, [34](#)
- dsvDescriptorHeap
 - FARender::DeviceResources, [24](#)
- dsvDescriptorSize
 - FARender::DeviceResources, [24](#)
- execute
 - FARender::DeviceResources, [24](#)
- FABuffer.h, [45](#)
- FACamera, [7](#)

- FACamera.h, 47
 - vec2, 47
- FACamera::Camera, 9
 - aspect, 12
 - backward, 12
 - Camera, 11
 - cameraPosition, 12
 - cameraVelocity, 12, 13
 - down, 13
 - foward, 13
 - keyboardInput, 13
 - left, 13
 - lookAt, 13
 - mouseInput, 14
 - perspectiveProjectionTransformationMatrix, 14
 - right, 14
 - rotateCameraLeftRight, 14
 - rotateCameraUpDown, 14
 - rotateVelocity, 15
 - up, 15
 - updatePerspectiveProjectionTransformationMatrix, 15
 - updateViewPerspectiveProjectionTransformationMatrix, 15
 - updateViewTransformationMatrix, 15
 - vFov, 16
 - viewPerspectiveProjectionTransformationMatrix, 16
 - viewTransformationMatrix, 16
 - x, 16
 - y, 16
 - z, 17
 - zfar, 17
 - znear, 17
- FADeviceResources.h, 49
- FARender, 7
 - nextFrame, 8
- FARender::ConstantBuffer, 18
 - ~ConstantBuffer, 18
 - constantBuffer, 18, 19
 - copyData, 19
 - createConstantBuffer, 19
 - createConstantBufferView, 19
 - mappedData, 19
- FARender::DeviceResources, 20
 - ~DeviceResources, 22
 - backBufferFormat, 22
 - commandAllocator, 22
 - commandList, 22
 - commandQueue, 23
 - currentBackBuffer, 23
 - currentFenceValue, 23
 - depthStencilBuffer, 23
 - depthStencilFormat, 23
 - device, 24
 - device2DContext, 24
 - directWriteFactory, 24
 - dsvDescriptorHeap, 24
 - dsvDescriptorSize, 24
 - execute, 24
 - flushCommandQueue, 25
 - initializeDirect3D, 25
 - isMSAAEnabled, 25
 - numOfSwapChainBuffers, 25
 - present, 26
 - resetCommandAllocator, 26
 - resetCommandList, 26
 - resetTextBuffers, 26
 - resize, 26
 - rtvDescriptorHeap, 26
 - rtvDescriptorSize, 27
 - sampleCount, 27
 - scissor, 27
 - signal, 27
 - swapChain, 27
 - swapChainBuffers, 28
 - textDraw, 28
 - textResize, 28
 - updateCurrentFrameFenceValue, 28
 - viewport, 28
 - waitForGPU, 29
- FARender::DrawArguments, 29
- FARender::IndexBuffer, 30
 - createIndexBuffer, 30
 - createIndexBufferView, 30
 - indexBufferView, 31
- FARender::RenderScene, 31
 - afterDraw, 32
 - beforeDraw, 33
 - cbvHeap, 33
 - cbvHeapRootParameter, 33
 - cbvSize, 33
 - createCBVHeap, 33
 - createConstantBuffer, 34
 - createConstantBufferView, 34
 - createIndexBuffer, 34
 - drawObjects, 34
- FARender::Text, 35
 - brush, 36
 - changeTextColor, 36
 - changeTextLocation, 36
 - changeTextSize, 36
 - changeTextString, 37
 - format, 37
 - initialize, 37
 - textColor, 37
 - textLocation, 37
 - textSize, 38
 - textString, 38
- FARender::VertexBuffer, 40
 - createVertexBuffer, 41
 - createVertexBufferView, 41
 - vertexBufferView, 41
- FARenderingUtility.h, 53
- FARenderScene.h, 54
- FAText.h, 56

- FATime.h, 58
- FATime::Time, 38
 - DeltaTime, 39
 - Reset, 39
 - Start, 39
 - Stop, 39
 - Tick, 39
 - Time, 38
 - TotalTime, 39
- FAWindow, 8
- FAWindow.h, 58
- FAWindow::Window, 41
 - createWindow, 42
 - height, 42
 - maximized, 42
 - minimized, 43
 - resizing, 43
 - showWindow, 43
 - width, 43
 - windowHandle, 43
- flushCommandQueue
 - FARender::DeviceResources, 25
- format
 - FARender::Text, 37
- foward
 - FACamera::Camera, 13
- height
 - FAWindow::Window, 42
- indexBufferView
 - FARender::IndexBuffer, 31
- initialize
 - FARender::Text, 37
- initializeDirect3D
 - FARender::DeviceResources, 25
- isMSAAEnabled
 - FARender::DeviceResources, 25
- keyboardInput
 - FACamera::Camera, 13
- left
 - FACamera::Camera, 13
- lookAt
 - FACamera::Camera, 13
- mappedData
 - FARender::ConstantBuffer, 19
- maximized
 - FAWindow::Window, 42
- minimized
 - FAWindow::Window, 43
- mouseInput
 - FACamera::Camera, 14
- nextFrame
 - FARender, 8
- numOfSwapChainBuffers
 - FARender::DeviceResources, 25
- perspectiveProjectionTransformationMatrix
 - FACamera::Camera, 14
- present
 - FARender::DeviceResources, 26
- Reset
 - FATime::Time, 39
- resetCommandAllocator
 - FARender::DeviceResources, 26
- resetCommandList
 - FARender::DeviceResources, 26
- resetTextBuffers
 - FARender::DeviceResources, 26
- resize
 - FARender::DeviceResources, 26
- resizing
 - FAWindow::Window, 43
- right
 - FACamera::Camera, 14
- rotateCameraLeftRight
 - FACamera::Camera, 14
- rotateCameraUpDown
 - FACamera::Camera, 14
- rotateVelocity
 - FACamera::Camera, 15
- rtvDescriptorHeap
 - FARender::DeviceResources, 26
- rtvDescriptorSize
 - FARender::DeviceResources, 27
- sampleCount
 - FARender::DeviceResources, 27
- scissor
 - FARender::DeviceResources, 27
- showWindow
 - FAWindow::Window, 43
- signal
 - FARender::DeviceResources, 27
- Start
 - FATime::Time, 39
- Stop
 - FATime::Time, 39
- swapChain
 - FARender::DeviceResources, 27
- swapChainBuffers
 - FARender::DeviceResources, 28
- textColor
 - FARender::Text, 37
- textDraw
 - FARender::DeviceResources, 28
- textLocation
 - FARender::Text, 37
- textResize
 - FARender::DeviceResources, 28
- textSize
 - FARender::Text, 38
- textString
 - FARender::Text, 38

Tick
 FATime::Time, [39](#)

Time, [40](#)
 FATime::Time, [38](#)

TotalTime
 FATime::Time, [39](#)

up
 FACamera::Camera, [15](#)

updateCurrentFrameFenceValue
 FARender::DeviceResources, [28](#)

updatePerspectiveProjectionTransformationMatrix
 FACamera::Camera, [15](#)

updateViewPerspectiveProjectionTransformationMatrix
 FACamera::Camera, [15](#)

updateViewTransformationMatrix
 FACamera::Camera, [15](#)

vec2
 FACamera.h, [47](#)

vertexBufferView
 FARender::VertexBuffer, [41](#)

vFov
 FACamera::Camera, [16](#)

viewPerspectiveProjectionTransformationMatrix
 FACamera::Camera, [16](#)

viewport
 FARender::DeviceResources, [28](#)

viewTransformationMatrix
 FACamera::Camera, [16](#)

waitForGPU
 FARender::DeviceResources, [29](#)

width
 FAWindow::Window, [43](#)

windowHandle
 FAWindow::Window, [43](#)

x
 FACamera::Camera, [16](#)

y
 FACamera::Camera, [16](#)

z
 FACamera::Camera, [17](#)

zfar
 FACamera::Camera, [17](#)

znear
 FACamera::Camera, [17](#)