Farouq Adepetu's Rendering Engine

Generated by Doxygen 1.9.4

Chapter 1

Namespace Index

1.1 Namespace List

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Namespace Documentation

4.1 RenderingEngine Namespace Reference

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

Classes

- struct 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.
  .
- class Color

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

  A wrapper for depth stencil buffer resources. Uses DirectD 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.
- class DeviceResources

  A wrapper for resources that are needed to render objects and text using the Direct3D 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.
- struct DrawArguments

  Data that are used as parameters to draw an object.
- class DynamicBuffer

  This class stores data in a Direct3D 12 upload buffer. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.
- class MultiSampling

  A wrapper for multisampling resources. Uses DirectD 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.
- struct OrthogrpahicProjection

  A struct that holds the properties for doing orthographics projection.
- struct PerspectiveProjection

  A struct that holds the properties for doing perspective projection.
- class RenderScene

  This class is used to render a scene using Direct3D 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.
- class RenderTargetBuffer

A wrapper for render target buffer resources. Uses DirectD 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

- class StaticBuffer

  This class stores data in a Direct3D 12 default buffer. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.
- class SwapChain

  A wrapper for swap chain resources. Uses DirectD 12 API and DXGI API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.
- class Text

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

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

  A struct that holds the properties for time.
- struct Window

  The window struct is used to make a Window using Win32 API.

Enumerations

- enum **BufferTypes** { **VERTEX_BUFFER** , **INDEX_BUFFER** , **CONSTANT_BUFFER** , **TEXTURE_BUFFER** }
- enum **TextureTypes** { **TEX2D** , **TEX2D_MS** }

Functions

- void SetProperties (Camera &camera, vec3 position, vec3 x, vec3 y, vec3 z, float linearSpeed, float angular↵ Speed)

  Sets the properties of the camera.
- void LookAt (Camera &camera, vec3 position, vec3 target, vec3 up)

  Defines the camera space using UVN.
- void UpdateViewMatrix (Camera &camera)

  After modifying the camera position and/or orientation, call this to rebuild the view transformation matrix.
- void Left (Camera &camera, float dt)

  Moves the camera left along the camera's x-axis.
- void Right (Camera &camera, float dt)

  Moves the camera right along the camera's x-axis.
- void Foward (Camera &camera, float dt)

  Moves the camera foward along the camera's z-axis.
- void Backward (Camera &camera, float dt)

  Moves the camera backward along the camera's z-axis.
- void Up (Camera &camera, float dt)

  Moves the camera up along the camera's y-axis.
- void Down (Camera &camera, float dt)

  Moves the camera down along the camera's y-axis.
- void RotateCameraLeftRight (Camera &camera, float xDiff)

  Rotates the camera to look left and right.
- void RotateCameraUpDown (Camera &camera, float yDiff)

  Rotates the camera to look up and down.
- Color operator+ (const Color &c1, const Color &c2)

  Returns the result of c1 + c2.

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

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

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

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

  Returns the result of c1 ∗ c2.
- void InitializeTime (Time &time)
- void Reset (Time &time)
- void Tick (Time &time)
- void Start (Time &time)
- void Stop (Time &time)
- void SetProperties (OrthogrpahicProjection &ortho, float width, float height, float znear, float zfar)

  Sets the properties of the OrthogrpahicProjection object to the specified values.
- void UpdateProjectionMatrix (OrthogrpahicProjection &ortho)

  Updates the perspective projection matrix of the PerspecitveProjection object.
- void SetProperties (PerspectiveProjection &perspective, float znear, float zfar, float vFov, float aspectRatio)

  Sets the properties of the PerspectiveProjection object to the specified values.
- void UpdateProjectionMatrix (PerspectiveProjection &perspective)

  Updates the perspective projection matrix of the PerspecitveProjection object.
- void CreateParentWindow (Window &window, const HINSTANCE &hInstance, WNDPROC window←Procedure, const RenderingEngine::Color &backgroundColor, const std::wstring &windowClassName, const std::wstring &windowName, unsigned int styles, unsigned int x, unsigned int y, unsigned int width, unsigned int height, void ∗additionalData=nullptr)

  Creates a parent window.
- void CreateChildWindow (Window &window, const HINSTANCE &hInstance, HWND parent, unsigned long long int identifier, WNDPROC windowProcedure, const RenderingEngine::Color &backgroundColor, const std::wstring &windowClassName, const std::wstring &windowName, unsigned int styles, unsigned int x, unsigned int y, unsigned int width, unsigned int height, void ∗additionalData=nullptr)

  Creates a non-control child window.
- void CreateControlWindow (Window &window, const HINSTANCE &hInstance, HWND parent, unsigned long long int identifier, const std::wstring &windowClassName, const std::wstring &windowName, unsigned int styles, unsigned int x, unsigned int y, unsigned int width, unsigned int height, void ∗additionalData=nullptr)

  Creates a control window.
- unsigned int GetWidth (const Window &window)
- unsigned int GetHeight (const Window &window)
- unsigned int GetX (const Window &window)
- unsigned int GetY (const Window &window)

4.1.1 Detailed Description

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

4.1.2 Function Documentation

4.1.2.1 Backward()

```
void RenderingEngine::Backward (
             Camera & camera,
             float dt )
```

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

Parameters

in	*camera*	The camera object.
in	*dt*	The time between frames.

4.1.2.2 CreateChildWindow()

```
void RenderingEngine::CreateChildWindow (
            Window & window,
            const HINSTANCE & hInstance,
            HWND parent,
            unsigned long long int identifier,
            WNDPROC windowProcedure,
            const RenderingEngine::Color & backgroundColor,
            const std::wstring & windowClassName,
            const std::wstring & windowName,
            unsigned int styles,
            unsigned int x,
            unsigned int y,
            unsigned int width,
            unsigned int height,
            void * additionalData = nullptr )
```

Creates a non-control child window.

Parameters

in	*window*	A Window object.
in	*hInstance*	The handle to a module used to identify the executable.
in	*parent*	A handle to a parent window.
in	*identifier*	An unsigned integer to identify the child window.
in	*windowProcedure*	The window procedure that is called when an event occurs.
in	*backgroundColor*	The background color the window.
in	*windowClassName*	The name of the window class.
in	*windowName*	The name of the window.
in	*styles*	The style of the window. OR together the styles at `https://learn.microsoft.↩ com/en-us/windows/win32/winmsg/window-styles`
in	*The*	x position of the top left corner of the window from the parent window top left corner.
in	*The*	y position of the top left corner of the window from the parent window top left corner..
in	*width*	The width of the client area of the window.
in	*height*	The height of the client area of the window.
	[in,optional]	additionalData A pointer to data to access in the window procedure.

4.1.2.3 CreateControlWindow()

```
void RenderingEngine::CreateControlWindow (
            Window & window,
            const HINSTANCE & hInstance,
            HWND parent,
            unsigned long long int identifier,
            const std::wstring & windowClassName,
            const std::wstring & windowName,
            unsigned int styles,
            unsigned int x,
            unsigned int y,
            unsigned int width,
            unsigned int height,
            void * additionalData = nullptr )
```

Creates a control window.

Parameters

in	*window*	A Window object.
in	*hInstance*	The handle to a module used to identify the executable.
in	*parent*	A handle to a parent window.
in	*identifier*	An unsigned integer to identify the child window.
in	*windowClass*	The name of the window class.
in	*windowName*	The name of the window.
in	*styles*	The style of the window. OR together the styles at `https://learn.microsoft.↩ com/en-us/windows/win32/winmsg/window-styles`
in	*The*	x position of the top left corner of the window from the parent window top left corner.
in	*The*	y position of the top left corner of the window from the parent window top left corner.
in	*width*	The width of the client area of the window.
in	*height*	The height of the client area of the window.
	[in,optional]	additionalData A pointer to data to access in the window procedure.

4.1.2.4 CreateParentWindow()

```
void RenderingEngine::CreateParentWindow (
            Window & window,
            const HINSTANCE & hInstance,
            WNDPROC windowProcedure,
            const RenderingEngine::Color & backgroundColor,
            const std::wstring & windowClassName,
            const std::wstring & windowName,
            unsigned int styles,
            unsigned int x,
            unsigned int y,
            unsigned int width,
            unsigned int height,
            void * additionalData = nullptr )
```

Creates a parent window.

The window gets displayed after it is created.

Parameters

in	*window*	A Window object.
in	*hInstance*	The handle to a module used to identify the executable.
in	*windowProcedure*	The window procedure that is called when an event occurs.
in	*backgroundColor*	The background color the window.
in	*windowClassName*	The name of the window class.
in	*windowName*	The name of the window.
in	*styles*	The style of the window. OR together the styles at `https://learn.microsoft.com/en-us/windows/win32/winmsg/window-styles`
in	*The*	x position of the top left corner of the window from the desktops top left corner. Use CW_USEDEFAULT to let system select a default position for you.
in	*The*	y position of the top left corner of the windo from the desktops top left corner. Use CW_USEDEFAULT to let system select a default position for you.
in	*width*	The width of the client area of the window.
in	*height*	The height of the client area of the window.
	[in,optional]	additionalData A pointer to data to access in the window procedure.

4.1.2.5 Down()

```
void RenderingEngine::Down (
            Camera & camera,
            float dt )
```

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

Parameters

in	*camera*	The camera object.
in	*dt*	The time between frames.

4.1.2.6 Foward()

```
void RenderingEngine::Foward (
            Camera & camera,
            float dt )
```

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

Parameters

in	*camera*	The camera object.
in	*dt*	The time between frames.

4.1.2.7 GetHeight()

```
unsigned int RenderingEngine::GetHeight (
             const Window & window )
```

brief Returns the height of the client area of the specified window.

4.1.2.8 GetWidth()

```
unsigned int RenderingEngine::GetWidth (
             const Window & window )
```

brief Returns the width of the client area of the specified window.

4.1.2.9 GetX()

```
unsigned int RenderingEngine::GetX (
             const Window & window )
```

brief Returns the x location of the specified window.

4.1.2.10 GetY()

```
unsigned int RenderingEngine::GetY (
             const Window & window )
```

brief Returns the y location of the specified window.

4.1.2.11 InitializeTime()

```
void RenderingEngine::InitializeTime (
             Time & time )
```

brief Initializes the specified Time object.

4.1.2.12 Left()

```
void RenderingEngine::Left (
             Camera & camera,
             float dt )
```

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

4.1.2.13 LookAt()

```
void RenderingEngine::LookAt (
             Camera & camera,
             vec3 position,
             vec3 target,
             vec3 up )
```

Defines the camera space using UVN.

Parameters

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

4.1.2.14 operator*() [1/3]

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

Returns the result of $c * k$.

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

4.1.2.15 operator*() [2/3]

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

Returns the result of $c1 * c2$.

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

4.1.2.16 operator*() [3/3]

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

Returns the result of $k * c$.

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

4.1.2.17 operator+()

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

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

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

4.1.2.18 operator-()

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

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

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

4.1.2.19 Reset()

```
void RenderingEngine::Reset (
             Time & time )
```

brief Resets the specified Time object.

4.1.2.20 Right()

```
void RenderingEngine::Right (
             Camera & camera,
             float dt )
```

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

Parameters

in	*camera*	The camera object.
in	*dt*	The time between frames.

4.1.2.21 RotateCameraLeftRight()

```
void RenderingEngine::RotateCameraLeftRight (
             Camera & camera,
             float xDiff )
```

Rotates the camera to look left and right.

Parameters

in	*camera*	The camera object.
in	*xDiff*	How many degrees to rotate.

4.1.2.22 RotateCameraUpDown()

```
void RenderingEngine::RotateCameraUpDown (
            Camera & camera,
            float yDiff )
```

Rotates the camera to look up and down.

Parameters

in	*camera*	The camera object.
in	*yDiff*	How many degrees to rotate.

4.1.2.23 SetProperties() [1/3]

```
void RenderingEngine::SetProperties (
            Camera & camera,
            vec3 position,
            vec3 x,
            vec3 y,
            vec3 z,
            float linearSpeed,
            float angularSpeed )
```

Sets the properties of the camera.

Parameters

in	*camera*	The camera object.
in	*position*	The position of the camera.
in	*x*	The x axis of the local coordinate system of the camera.
in	*y*	The y axis of the local coordinate system of the camera.
in	*z*	The z axis of the local coordinate system of the camera.
in	*linearVelocity*	The translational velocity of the camera.
in	*angularVelocity*	The angular velocity of the camera.

4.1.2.24 SetProperties() [2/3]

```
void RenderingEngine::SetProperties (
            OrthogrpahicProjection & ortho,
            float width,
            float height,
            float znear,
            float zfar )
```

Sets the properties of the OrthogrpahicProjection object to the specified values.

Parameters

in	*ortho*	The OrthogrpahicProjection object.
in	*width*	The width of the box.
in	*heigth*	The height of the box.
in	*znear*	The z value of the near plane.
in	*zfar*	The z value of the far plane.

4.1.2.25 SetProperties() [3/3]

```
void RenderingEngine::SetProperties (
            PerspectiveProjection & perspective,
            float znear,
            float zfar,
            float vFov,
            float aspectRatio )
```

Sets the properties of the PerspectiveProjection object to the specified values.

Parameters

in	*perspective*	The PerspectiveProjection object.
in	*znear*	The z value of where the near plane of the frustrum intersects the z-axis.
in	*zfar*	The z value of where the far plane of the frustrum intersects the z-axis.
in	*vFov*	The vertical field of view of the frustrum.
in	*aspectRatio*	The aspect ratio of the view plane.

4.1.2.26 Start()

```
void RenderingEngine::Start (
            Time & time )
```

brief Starts the time for the specified Time object.

4.1.2.27 Stop()

```
void RenderingEngine::Stop (
            Time & time )
```

brief Stops the time for the specified Time object.

4.1.2.28 Tick()

```
void RenderingEngine::Tick (
            Time & time )
```

brief Computes the delta time (time between each frame) for the specified Time object.

4.1.2.29 Up()

```
void RenderingEngine::Up (
            Camera & camera,
            float dt )
```

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

Parameters

in	*camera*	The camera object.
in	*dt*	The time between frames.

4.1.2.30 UpdateProjectionMatrix() [1/2]

```
void RenderingEngine::UpdateProjectionMatrix (
            OrthogrpahicProjection & ortho )
```

Updates the perspective projection matrix of the PerspecitveProjection object.

Parameters

in	*p*	The PerspectiveProjection object.

4.1.2.31 UpdateProjectionMatrix() [2/2]

```
void RenderingEngine::UpdateProjectionMatrix (
            PerspectiveProjection & perspective )
```

Updates the perspective projection matrix of the PerspecitveProjection object.

Parameters

in	*p*	The PerspectiveProjection object.

4.1.2.32 UpdateViewMatrix()

```
void RenderingEngine::UpdateViewMatrix (
            Camera & camera )
```

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

Chapter 5

Class Documentation

5.1 RenderingEngine::Camera Struct 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.
.

```
#include "Camera.h"
```

Public Attributes

- vec3 **position**
- vec3 **x**
- vec3 **y**
- vec3 **z**
- mat4 **viewMatrix**
- float **linearSpeed** = 0.0f
- float **angularSpeed** = 0.0f

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.
.

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

- Camera.h

5.2 RenderingEngine::Color Class Reference

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

```
#include "Color.h"
```

Public Member Functions

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

    Initializes the color to the specified RGBA values.
- Color (const vec4 &color)

    Initializes the color to the specified color.
- const vec4 & GetColor () const

    Returns the color.
- float GetRed () const

    Returns the value of the red component.
- float GetGreen () const

    Returns the value of the blue component.
- float GetBlue () const

    Returns the value of the green component.
- float GetAlpha () const

    Returns the value of the alpha component.
- void SetColor (const vec4 &color)

    Sets the color to the specified color.
- void SetRed (float r)

    Sets the red component to the specified float value.
- void SetGreen (float g)

    Sets the green component to the specified float value.
- void SetBlue (float b)

    Sets the blue component to the specified float value.
- void SetAlpha (float a)

    Sets the alpha component to the specified float value.
- Color & operator+= (const Color &c)

    Adds this objects color to the specified color c and stores the result in this object.
- Color & operator-= (const Color &c)

    Subtracts the specified color c from this objects color and stores the result in this object.
- Color & operator*= (float k)

    Multiplies this objects color by the specified value k and stores the result in this object.
- Color & operator*= (const Color &c)

    Multiplies this objects color by the specified color c and stores the result in this object.

5.2.1 Detailed Description

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

5.2.2 Constructor & Destructor Documentation

5.2.2.1 Color() `[1/2]`

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

Initializes the color to the specified RGBA values.

5.2.2.2 Color() `[2/2]`

```
RenderingEngine::Color::Color (
             const vec4 & color )
```

Initializes the color to the specified *color*.

5.2.3 Member Function Documentation

5.2.3.1 GetAlpha()

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

Returns the value of the alpha component.

5.2.3.2 GetBlue()

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

Returns the value of the green component.

5.2.3.3 GetColor()

```
const vec4 & RenderingEngine::Color::GetColor ( ) const
```

Returns the color.

5.2.3.4 GetGreen()

```
float RenderingEngine::Color::GetGreen ( ) const
```

Returns the value of the blue component.

5.2.3.5 GetRed()

```
float RenderingEngine::Color::GetRed ( ) const
```

Returns the value of the red component.

5.2.3.6 operator*=() [1/2]

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

Multiplies this objects color by the specified color *c* and stores the result in this object.

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

5.2.3.7 operator*=() [2/2]

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

Multiplies this objects color by the specified value *k* and stores the result in this object.

If $k <$ 0.0f, no multiplication happens and this objects color does not get modified.
If any of the resultant components are $>$ 1.0f, they are set to 1.0f.

5.2.3.8 operator+=()

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

Adds this objects color to the specified color *c* and stores the result in this object.

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

5.2.3.9 operator-=()

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

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

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

5.2.3.10 SetAlpha()

```
void RenderingEngine::Color::SetAlpha (
             float a )
```

Sets the alpha component to the specified float value.

5.2.3.11 SetBlue()

```
void RenderingEngine::Color::SetBlue (
             float b )
```

Sets the blue component to the specified float value.

5.2.3.12 SetColor()

```
void RenderingEngine::Color::SetColor (
             const vec4 & color )
```

Sets the color to the specified color.

5.2.3.13 SetGreen()

```
void RenderingEngine::Color::SetGreen (
             float g )
```

Sets the green component to the specified float value.

5.2.3.14 SetRed()

```
void RenderingEngine::Color::SetRed (
            float r )
```

Sets the red component to the specified float value.

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

- Color.h

5.3 RenderingEngine::DepthStencilBuffer Class Reference

A wrapper for depth stencil buffer resources. Uses DirectD 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

```
#include "Buffer.h"
```

Public Member Functions

- **DepthStencilBuffer** (const DepthStencilBuffer &)=delete
- DepthStencilBuffer & **operator=** (const DepthStencilBuffer &)=delete
- DepthStencilBuffer ()

  Creates a depth stencil buffer object. Call the CreateDepthStencilBufferAndView() to allocate memory for the buffer.
- DepthStencilBuffer (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::↩ ComPtr< ID3D12DescriptorHeap > &dsvHeap, unsigned int index, unsigned int dsvSize, unsigned int width, unsigned int height, DXGI_FORMAT format=DXGI_FORMAT_D24_UNORM_S8_UINT, unsigned int sampleCount=1)

  Creates the depth stencil buffer and view.
- DXGI_FORMAT GetDepthStencilFormat () const

  Returns the format of the depth stencil buffer.
- void CreateDepthStencilBufferAndView (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &dsvHeap, unsigned int index, unsigned int dsvSize, unsigned int width, unsigned int height, DXGI_FORMAT format=DXGI_FORMAT_D24_UNORM_S8_UINT, unsigned int sampleCount=1)

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

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

  Clears the depth stencil buffer with the specified clear value.

5.3.1 Detailed Description

A wrapper for depth stencil buffer resources. Uses DirectD 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 DepthStencilBuffer() [1/2]

```
RenderingEngine::DepthStencilBuffer::DepthStencilBuffer ( )
```

Creates a depth stencil buffer object. Call the CreateDepthStencilBufferAndView() to allocate memory for the buffer.

5.3.2.2 DepthStencilBuffer() [2/2]

```
RenderingEngine::DepthStencilBuffer::DepthStencilBuffer (
            const Microsoft::WRL::ComPtr< ID3D12Device > & device,
            const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & dsvHeap,
            unsigned int index,
            unsigned int dsvSize,
            unsigned int width,
            unsigned int height,
            DXGI_FORMAT format = DXGI_FORMAT_D24_UNORM_S8_UINT,
            unsigned int sampleCount = 1 )
```

Creates the depth stencil buffer and view.

Parameters

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

5.3.3 Member Function Documentation

5.3.3.1 ClearDepthStencilBuffer()

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

Clears the depth stencil buffer with the specified clear value.

Parameters

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

5.3.3.2 CreateDepthStencilBufferAndView()

```
void RenderingEngine::DepthStencilBuffer::CreateDepthStencilBufferAndView (
            const Microsoft::WRL::ComPtr< ID3D12Device > & device,
            const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & dsvHeap,
            unsigned int index,
            unsigned int dsvSize,
            unsigned int width,
            unsigned int height,
            DXGI_FORMAT format = DXGI_FORMAT_D24_UNORM_S8_UINT,
            unsigned int sampleCount = 1 )
```

Creates the depth stencil buffer and view.

Parameters

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

5.3.3.3 GetDepthStencilFormat()

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

Returns the format of the depth stencil buffer.

5.3.3.4 ReleaseBuffer()

```
void RenderingEngine::DepthStencilBuffer::ReleaseBuffer ( )
```

Frees the memory of the buffer.

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

- Buffer.h

5.4 RenderingEngine::DeviceResources Class Reference

A wrapper for resources that are needed to render objects and text using the Direct3D 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

```
#include "DeviceResources.h"
```

Public Member Functions

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

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

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

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

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

  Returns a constant reference to the depth stencil format.
- unsigned int GetCBVSRVUAVSize () const

  The size of a constant buffer/shader resource/unordered access view.
- unsigned int GetCurrentFrame () const

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

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

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

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

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

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

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

  Transitions the render target buffer.
- void BeforeTextDraw ()

  Prepares to render text.
- void AfterTextDraw ()

  Executes the text commands.
- void Execute () const

  Executes the command list.
- void Present ()

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

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

Static Public Member Functions

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

  Call to make an object of DeviceResources.

Static Public Attributes

- static const unsigned int NUM_OF_FRAMES { 3 }

  The number of frames in the ciruclar array.

5.4.1 Detailed Description

A wrapper for resources that are needed to render objects and text using the Direct3D 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 ∼DeviceResources()

```
RenderingEngine::DeviceResources::∼DeviceResources ( )
```

Flushes the command queue.

5.4.3 Member Function Documentation

5.4.3.1 AfterTextDraw()

```
void RenderingEngine::DeviceResources::AfterTextDraw ( )
```

Executes the text commands.

5.4.3.2 BeforeTextDraw()

```
void RenderingEngine::DeviceResources::BeforeTextDraw ( )
```

Prepares to render text.

5.4.3.3 Execute()

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

Executes the command list.

5.4.3.4 FlushCommandQueue()

```
void RenderingEngine::DeviceResources::FlushCommandQueue ( )
```

Synchronizes the CPU and GPU.

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

5.4.3.5 GetBackBufferFormat()

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

Returns a constant reference to the back buffer format.

5.4.3.6 GetCBVSRVUAVSize()

```
unsigned int RenderingEngine::DeviceResources::GetCBVSRVUAVSize ( ) const
```

The size of a constant buffer/shader resource/unordered access view.

5.4.3.7 GetCommandList()

```
const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & RenderingEngine::DeviceResources↩
::GetCommandList ( ) const
```

Returns a constant reference to the ID3D12GraphicsCommandList objcet.

5.4.3.8 GetCurrentFrame()

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

Returns the current frame.

5.4.3.9 GetDepthStencilFormat()

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

Returns a constant reference to the depth stencil format.

5.4.3.10 GetDevice()

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

Returns a constant reference to the ID3D12Device objcet.

5.4.3.11 GetInstance()

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

Call to make an object of DeviceResources.

Only one instance of DeviceResources can exist in a program.

Parameters

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

5.4.3.12 GetTextResources()

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

Returns a constant reference to the TextResources object.

5.4.3.13 NextFrame()

```
void RenderingEngine::DeviceResources::NextFrame ( )
```

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

5.4.3.14 Present()

```
void RenderingEngine::DeviceResources::Present ( )
```

Swaps the front and back buffers.

5.4.3.15 Resize()

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

Call when the window gets resized.

Call when you initialize your program.

Parameters

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

5.4.3.16 RTBufferTransition()

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

Transistions the render target buffer.

Parameters

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

5.4.3.17 Signal()

```
void RenderingEngine::DeviceResources::Signal ( )
```

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

5.4.3.18 UpdateCurrentFrameFenceValue()

```
void RenderingEngine::DeviceResources::UpdateCurrentFrameFenceValue ( )
```

Updates the current frames fence value.

5.4.3.19 WaitForGPU()

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

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

Signal should have been called before this function is called.

5.4.4 Member Data Documentation

5.4.4.1 NUM_OF_FRAMES

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

The number of frames in the ciruclar array.

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

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

- DeviceResources.h

5.5 DirectXException Class Reference

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

```
#include "DirectXException.h"
```

Public Member Functions

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

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

  Returns a message describing the error.

5.5.1 Detailed Description

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

5.5.2 Constructor & Destructor Documentation

5.5.2.1 DirectXException()

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

Constructs a DirectXException object.

Parameters

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

5.5.3 Member Function Documentation

5.5.3.1 ErrorMsg()

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

Returns a message describing the error.

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

- DirectXException.h

5.6 RenderingEngine::DrawArguments Struct Reference

Data that are used as parameters to draw an object.

```
#include "DrawArguments.h"
```

Public Attributes

- unsigned int **indexCount** = 0
- unsigned int **locationOfFirstIndex** = 0
- int **indexOfFirstVertex** = 0
- unsigned int **indexOfConstantData** = 0
- unsigned int **rootParameterIndex** = 0
- std::wstring **constantBufferKey** = L""
- D3D_PRIMITIVE_TOPOLOGY **primtive** = D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST

5.6.1 Detailed Description

Data that are used as parameters to draw an object.

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

- DrawArguments.h

5.7 RenderingEngine::DynamicBuffer Class Reference

This class stores data in a Direct3D 12 upload buffer. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

```
#include "Buffer.h"
```

Public Member Functions

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

  Creates a dynamic buffer object. No memory is allocated for the buffer. Call one of the CreateDynamicBuffer() functions to allocate memory for the buffer.
- DynamicBuffer (const Microsoft::WRL::ComPtr< ID3D12Device > &device, unsigned int numOfBytes, unsigned int stride)

  Creates and maps a dynamic vertex buffer or a dynamic constant buffer.
- DynamicBuffer (const Microsoft::WRL::ComPtr< ID3D12Device > &device, unsigned int numOfBytes, DXGI_FORMAT format)

  Creates and maps a dynamic index buffer.
- ∼DynamicBuffer ()

  Unmaps the pointer to the dynamic buffer.
- void CreateDynamicBuffer (const Microsoft::WRL::ComPtr< ID3D12Device > &device, unsigned int num←OfBytes, unsigned int stride)

Creates and maps a dynamic vertex buffer or a dynamic constant buffer.

- void CreateDynamicBuffer (const Microsoft::WRL::ComPtr< ID3D12Device > &device, unsigned int num↩ OfBytes, DXGI_FORMAT format)

  Creates and maps a dynamic index buffer.
- const D3D12_GPU_VIRTUAL_ADDRESS GetGPUAddress (unsigned int index) const

  Returns the GPU address of the data at the specified index.
- void CreateConstantBufferView (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &cbvHeap, unsigned int cbvSize, unsigned int cbv↩ HeapIndex, unsigned int cBufferIndex)

  Creates the constant buffer view and stores it in the specified descriptor heap.
- const D3D12_VERTEX_BUFFER_VIEW GetVertexBufferView ()

  Returns a the vertex buffer view of the dynamic buffer.
- const D3D12_INDEX_BUFFER_VIEW GetIndexBufferView ()

  Returns the index buffer view of the dynamic buffer.
- void CopyData (unsigned int index, const void ∗data, unsigned long long numOfBytes)

  Copies data from the given data into the dynamic buffer. Uses 0-indexing.
- void ReleaseBuffer ()

  Frees the dynamic buffer memory.

5.7.1 Detailed Description

This class stores data in a Direct3D 12 upload buffer. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 DynamicBuffer() [1/3]

```
RenderingEngine::DynamicBuffer::DynamicBuffer ( )
```

Creates a dynamic buffer object. No memory is allocated for the buffer. Call one of the CreateDynamicBuffer() functions to allocate memory for the buffer.

5.7.2.2 DynamicBuffer() [2/3]

```
RenderingEngine::DynamicBuffer::DynamicBuffer (
            const Microsoft::WRL::ComPtr< ID3D12Device > & device,
            unsigned int numOfBytes,
            unsigned int stride )
```

Creates and maps a dynamic vertex buffer or a dynamic constant buffer.

Parameters

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

5.7.2.3 DynamicBuffer() [3/3]

```
RenderingEngine::DynamicBuffer::DynamicBuffer (
            const Microsoft::WRL::ComPtr< ID3D12Device > & device,
            unsigned int numOfBytes,
            DXGI_FORMAT format )
```

Creates and maps a dynamic index buffer.

Parameters

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

5.7.2.4 ∼DynamicBuffer()

```
RenderingEngine::DynamicBuffer::∼DynamicBuffer ( )
```

Unmaps the pointer to the dynamic buffer.

5.7.3 Member Function Documentation

5.7.3.1 CopyData()

```
void RenderingEngine::DynamicBuffer::CopyData (
            unsigned int index,
            const void * data,
            unsigned long long numOfBytes )
```

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

Parameters

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

5.7.3.2 CreateConstantBufferView()

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

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

Parameters

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

5.7.3.3 CreateDynamicBuffer() [1/2]

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

Creates and maps a dynamic index buffer.

Parameters

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

5.7.3.4 CreateDynamicBuffer() [2/2]

```
void RenderingEngine::DynamicBuffer::CreateDynamicBuffer (
            const Microsoft::WRL::ComPtr< ID3D12Device > & device,
            unsigned int numOfBytes,
            unsigned int stride )
```

Creates and maps a dynamic vertex buffer or a dynamic constant buffer.

Parameters

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

5.7.3.5 GetGPUAddress()

```
const D3D12_GPU_VIRTUAL_ADDRESS RenderingEngine::DynamicBuffer::GetGPUAddress (
             unsigned int index ) const
```

Returns the GPU address of the data at the specified index.

5.7.3.6 GetIndexBufferView()

```
const D3D12_INDEX_BUFFER_VIEW RenderingEngine::DynamicBuffer::GetIndexBufferView ( )
```

Returns the index buffer view of the dynamic buffer.

5.7.3.7 GetVertexBufferView()

```
const D3D12_VERTEX_BUFFER_VIEW RenderingEngine::DynamicBuffer::GetVertexBufferView ( )
```

Returns a the vertex buffer view of the dynamic buffer.

5.7.3.8 ReleaseBuffer()

```
void RenderingEngine::DynamicBuffer::ReleaseBuffer ( )
```

Frees the dynamic buffer memory.

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

- Buffer.h

5.8 RenderingEngine::MultiSampling Class Reference

A wrapper for multisampling resources. Uses DirectD 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

```
#include "Multisampling.h"
```

Public Member Functions

- **MultiSampling** (const MultiSampling &)=delete
- MultiSampling & **operator=** (const MultiSampling &)=delete
- MultiSampling ()

  Creates a multisampling object. Call the function CheckMultiSamplingSupport() to check if the desired formats and sample count is supported by a GPU. If they are supported, a mulitsampling render target buffer and a mulitsampling depth stencil buffer can be created.
- MultiSampling (const Microsoft::WRL::ComPtr< ID3D12Device > &device, DXGI_FORMAT rtFormat, unsigned int sampleCount)

  Checks if the specifed format and sample count are supported by the specified device for multi-sampling.
- void CheckMultiSamplingSupport (const Microsoft::WRL::ComPtr< ID3D12Device > &device, DXGI_↩ FORMAT rtFormat, unsigned int sampleCount)

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

  Returns a constant refeerence to the MSAA render target buffer.
- DXGI_FORMAT GetRenderTargetFormat ()

  Returns the format of the MSAA render target buffer.
- DXGI_FORMAT GetDepthStencilFormat ()

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

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

  Creates the MSAA render target buffer and a view to it.
- void CreateDepthStencilBufferAndView (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &dsvHeap, unsigned int indexOfWhereToStoreView, unsigned int dsvSize, unsigned int width, unsigned int height)

  Creates the MSAA depth stencil buffer and a view to it.
- void Transition (const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, D3D12↩ _RESOURCE_STATES before, D3D12_RESOURCE_STATES after)

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

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

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

5.8.1 Detailed Description

A wrapper for multisampling resources. Uses DirectD 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 MultiSampling() [1/2]

```
RenderingEngine::MultiSampling::MultiSampling ( )
```

Creates a multisampling object. Call the function CheckMultiSamplingSupport() to check if the desired formats and sample count is supported by a GPU. If they are supported, a mulitsampling render target buffer and a mulitsampling depth stencil buffer can be created.

5.8.2.2 MultiSampling() [2/2]

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

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

Throws a runtime_error if they are not supproted.

Parameters

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

5.8.3 Member Function Documentation

5.8.3.1 CheckMultiSamplingSupport()

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

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

Throws a runtime_error if they are not supproted.

Parameters

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

5.8.3.2 ClearDepthStencilBuffer()

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

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

Parameters

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

5.8.3.3 ClearRenderTargetBuffer()

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

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

Parameters

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

5.8.3.4 CreateDepthStencilBufferAndView()

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

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

Parameters

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

5.8.3.5 CreateRenderTargetBufferAndView()

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

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

Parameters

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

5.8.3.6 GetDepthStencilFormat()

```
DXGI_FORMAT RenderingEngine::MultiSampling::GetDepthStencilFormat ( )
```

Returns the format of the MSAA depth stencil buffer.

5.8.3.7 GetRenderTargetBuffer()

```
const Microsoft::WRL::ComPtr< ID3D12Resource > & RenderingEngine::MultiSampling::GetRender↩
TargetBuffer ( )
```

Returns a constant refererence to the MSAA render target buffer.

5.8.3.8 GetRenderTargetFormat()

```
DXGI_FORMAT RenderingEngine::MultiSampling::GetRenderTargetFormat ( )
```

Returns the format of the MSAA render target buffer.

5.8.3.9 ReleaseBuffers()

```
void RenderingEngine::MultiSampling::ReleaseBuffers ( )
```

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

5.8.3.10 Transition()

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

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

Parameters

in	*commandList*	A Direct3D 12 graphics command list.

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

- Multisampling.h

5.9 RenderingEngine::OrthogrpahicProjection Struct Reference

A struct that holds the properties for doing orthographics projection.

```
#include "OrthographicProjection.h"
```

Public Attributes

- float **width** = 0.0f
- float **height** = 0.0f
- float **znear** = 0.0f
- float **zfar** = 0.0f
- mat4 **projectionMatrix**

5.9.1 Detailed Description

A struct that holds the properties for doing orthographics projection.

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

- OrthographicProjection.h

5.10 RenderingEngine::PerspectiveProjection Struct Reference

A struct that holds the properties for doing perspective projection.

```
#include "PerspectiveProjection.h"
```

Public Attributes

- float **znear** = 0.0f
- float **zfar** = 0.0f
- float **verticalFov** = 0.0f
- float **aspectRatio** = 0.0f
- mat4 **projectionMatrix**

5.10.1 Detailed Description

A struct that holds the properties for doing perspective projection.

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

- PerspectiveProjection.h

5.11 RenderingEngine::RenderScene Class Reference

This class is used to render a scene using Direct3D 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

```
#include "RenderScene.h"
```

Public Member Functions

- **RenderScene** (const RenderScene &)=delete
- RenderScene **operator=** (const RenderScene &)=delete
- **RenderScene** (unsigned int width, unsigned int height, HWND windowHandle, bool isMSAAEnabled=false, bool isTextEnabled=false)
- void **CreateDeviceResources** (unsigned int width, unsigned int height, HWND windowHandle, bool is↩ MSAAEnabled=false, bool isTextEnabled=false)
- void LoadShader (unsigned int shaderKey, std::wstring_view filename)

  Loads a shaders bytecode and maps it to the specified shaderKey.
- void CompileShader (unsigned int shaderKey, std::wstring_view filename, std::string_view entryPointName, std::string_view target)

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

  Removes the shader bytecode mapped to the specified shaderKey.
- void LoadShader (std::wstring_view shaderKey, std::wstring_view filename)

  Loads a shaders bytecode and maps it to the specified shaderKey.
- void CompileShader (std::wstring_view shaderKey, std::wstring_view filename, std::string_view entryPoint↩ Name, std::string_view target)

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

  Removes the shader bytecode mapped to the specified shaderKey.
- void CreateInputElementDescription (unsigned int key, const char ∗semanticName, unsigned int semantic↩ Index, DXGI_FORMAT format, unsigned int inputSlot, unsigned int byteOffset, D3D12_INPUT_↩ CLASSIFICATION inputSlotClass=D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA, unsigned int instanceStepRate=0)

  Creates an input element description and stores in an array mapped to the specified key.
- void CreateInputElementDescription (std::wstring_view key, const char ∗semanticName, unsigned int semanticIndex, DXGI_FORMAT format, unsigned int inputSlot, unsigned int byteOffset, D3D12_INPUT↩ _CLASSIFICATION inputSlotClass=D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA, unsigned int instanceStepRate=0)

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

  Creates a root descriptor and stores it in the array mapped to the specified rootParameterKey.
- void CreateDescriptorRange (unsigned int descriptorRangeKey, D3D12_DESCRIPTOR_RANGE_TYPE type, unsigned int numDescriptors, unsigned int shaderRegister, unsigned int registerSpace, unsigned int offset)

  Creates a descriptor range and stores it in the array mapped to the specified descriptorRangeKey.
- void CreateDescriptorTable (unsigned int rootParameterKey, unsigned int descriptorRangeKey)

  Creates a root descriptor table and stores it in the array mapped to the specified rootParameterKey.
- void CreateRootConstants (unsigned int rootParameterKey, unsigned int shaderRegister, unsigned int num↩ Values)

  Creates a root constant and stores it in the array mapped to the specified rootParameterKey.
- void CreateRootDescriptor (std::wstring_view rootParameterKey, unsigned int shaderRegister)

  Creates a root descriptor and stores it in the array mapped to the specified rootParameterKey.
- void CreateDescriptorRange (std::wstring_view descriptorRangeKey, D3D12_DESCRIPTOR_RANGE_TYPE type, unsigned int numDescriptors, unsigned int shaderRegister, unsigned int registerSpace, unsigned int offset)

  Creates a descriptor range and stores it in the array mapped to the specified descriptorRangeKey.
- void CreateDescriptorTable (std::wstring_view rootParameterKey, unsigned int descriptorRangeKey)

  Creates a root descriptor table and stores it in the array mapped to the specified rootParameterKey.
- void CreateRootConstants (std::wstring_view rootParameterKey, unsigned int shaderRegister, unsigned int numValues)

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

  Creates a root signature and maps it to the specified rootSigKey.
- void CreateRootSignature (unsigned int rootSigKey, unsigned int rootParametersKey, unsigned int statics↩ SamplerKey)

  Creates a root signature and maps it to the specified rootSigKey.
- void CreateStaticSampler (unsigned int staticSamplerKey, D3D12_FILTER filter, D3D12_TEXTURE_↩ ADDRESS_MODE u, D3D12_TEXTURE_ADDRESS_MODE v, D3D12_TEXTURE_ADDRESS_MODE w, unsigned int shaderRegister)

  Creates a static sampler and stores in an an array mapped to the specified key.
- void CreateRootSignature (std::wstring_view rootSigKey, std::wstring_view rootParametersKey)

  Creates a root signature and maps it to the specified rootSigKey.
- void CreateRootSignature (std::wstring_view rootSigKey, std::wstring_view rootParametersKey, std↩ ::wstring_view staticsSamplerKey)

  Creates a root signature and maps it to the specified rootSigKey.
- void CreateStaticSampler (std::wstring_view staticSamplerKey, D3D12_FILTER filter, D3D12_TEXTURE↩ _ADDRESS_MODE u, D3D12_TEXTURE_ADDRESS_MODE v, D3D12_TEXTURE_ADDRESS_MODE w, unsigned int shaderRegister)

  Creates a static sampler and stores in an an array mapped to the specified key.
- void CreatePSO (unsigned int psoKey, D3D12_FILL_MODE fillMode, BOOL enableMultisample, unsigned int vsKey, unsigned int psKey, unsigned int inputElementDescriptionsKey, unsigned int rootSigKey, const D3↩ D12_PRIMITIVE_TOPOLOGY_TYPE &primitiveType, UINT sampleCount=1)

  Creates a PSO and maps it to the specified psoKey.
- void LinkPSOAndRootSignature (unsigned int psoKey, unsigned int rootSigKey)

  Links the PSO and its associated root signature to the pipeline to indicate what settings you want to use to render objects. An out_of_range exception is thrown if any of the keys don't have a mapped values.
- void CreatePSO (std::wstring_view psoKey, D3D12_FILL_MODE fillMode, BOOL enableMultisample, std↩ ::wstring_view vsKey, std::wstring_view psKey, std::wstring_view inputElementDescriptionsKey, std::wstring↩ _view rootSigKey, const D3D12_PRIMITIVE_TOPOLOGY_TYPE &primitiveType, UINT sampleCount=1)

  Creates a PSO and maps it to the specified psoKey.
- void LinkPSOAndRootSignature (std::wstring_view psoKey, std::wstring_view rootSigKey)

  Links the PSO and its associated root signature to the pipeline to indicate what settings you want to use to render objects. An out_of_range exception is thrown if any of the keys don't have a mapped values.
- void CreateStaticBuffer (unsigned int staticBufferKey, const void ∗data, unsigned numBytes, unsigned int stride)

  Creates a static vertex buffer and stores the specified data in the buffer.
- void CreateStaticBuffer (unsigned int staticBufferKey, const void ∗data, unsigned numBytes, DXGI_FORMAT format)

  Creates a static index buffer and stores the specified data in the buffer.
- void CreateStaticBuffer (unsigned int staticBufferKey, const wchar_t ∗filename, unsigned int texType, unsigned int index)

  Creates a static texture buffer, stores the data from the file into the buffer and creates a view of the texture.
- void LinkStaticBuffer (unsigned int bufferType, unsigned int staticBufferKey)

  Links the static buffer mapped to the static buffer key to the pipeline.
- void CreateStaticBuffer (std::wstring_view staticBufferKey, const void ∗data, unsigned numBytes, unsigned int stride)

  Creates a static vertex buffer and stores the specified data in the buffer.
- void CreateStaticBuffer (std::wstring_view staticBufferKey, const void ∗data, unsigned numBytes, DXGI_↩ FORMAT format)

  Creates a static index buffer and stores the specified data in the buffer.
- void CreateStaticBuffer (std::wstring_view staticBufferKey, const wchar_t ∗filename, unsigned int texType, unsigned int index)

  Creates a static texture buffer, stores the data from the file into the buffer and creates a view of the texture.

- void LinkStaticBuffer (unsigned int bufferType, std::wstring_view staticBufferKey)

    Links the static buffer mapped to the static buffer key to the pipeline.
- void CreateDynamicBuffer (unsigned int dynamicBufferKey, unsigned numBytes, const void ∗data, unsigned int stride)

    Creates a dynamic vertex buffer or a dynamic constant buffer.
- void CreateDynamicBuffer (unsigned int dynamicBufferKey, unsigned numBytes, const void ∗data, DXGI_←FORMAT format)

    Creates a dynamic index buffer.
- void LinkDynamicBuffer (unsigned int bufferType, unsigned int dynamicBufferKey, unsigned int index←ConstantData=0, unsigned int rootParameterIndex=0)

    Links the dynamic buffer mapped to the dynamic buffer key to the pipeline.
- void CopyDataIntoDynamicBuffer (unsigned int dynamicBufferKey, unsigned int index, const void ∗data, UINT64 numOfBytes)

    Copies the specified data into the dyanmic buffer mapped to the dynamic buffer key.
- void CreateDynamicBuffer (std::wstring_view dynamicBufferKey, unsigned numBytes, const void ∗data, unsigned int stride)

    Creates a dynamic vertex buffer or a dynamic constant buffer.
- void CreateDynamicBuffer (std::wstring_view dynamicBufferKey, unsigned numBytes, const void ∗data, DXGI_FORMAT format)

    Creates a dynamic index buffer.
- void LinkDynamicBuffer (unsigned int bufferType, std::wstring_view dynamicBufferKey, unsigned int index←ConstantData=0, unsigned int rootParameterIndex=0)

    Links the dynamic buffer mapped to the dynamic buffer key to the pipeline.
- void CopyDataIntoDynamicBuffer (std::wstring_view dynamicBufferKey, unsigned int index, const void ∗data, UINT64 numOfBytes)

    Copies the specified data into the dyanmic buffer mapped to the dynamic buffer key.
- void CreateTextureViewHeap (unsigned int numDescriptors)

    Creates a descriptor heap to store views of textures.
- void LinkTextureViewHeap ()

    Links the texture view heap to the pipeline.
- void LinkTexture (unsigned int rootParameterIndex)

    Links the set of textures in the descriptor table to the pipeline.
- void LinkTexture (unsigned int rootParameterIndex, unsigned int textureViewIndex)

    Links a texture to the pipeline.
- void BeforeRenderObjects (bool isMSAAEnabled=false)

    Puts all of the commands needed in the command list before drawing the objects of the scene.
- void RenderObject (unsigned int indexCount, unsigned int locationFirstIndex, int indexOfFirstVertex, D3D_←PRIMITIVE_TOPOLOGY primitive)

    Renders an object with the specified draw arguments.
- void AfterRenderObjects (bool isMSAAEnabled=false, bool isTextEnabled=false)

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

    Puts all of the commands needed in the command list before drawing the text of the scene. Call before calling the first RenderText function.
- void RenderText (const vec4 &textLocation, const Color &textColor, float textSize, const std::wstring &text←String, DWRITE_PARAGRAPH_ALIGNMENT alignment=DWRITE_PARAGRAPH_ALIGNMENT_CENTER)

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

    .
- void AfterRenderText ()

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

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

- void ExecuteAndFlush ()

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

    Resizes the window-dependent resources when the window gets resized.
- void SetConstants (unsigned int rootParameterIndex, unsigned int numValues, void ∗data, unsigned int index)

    Links an array of 32-bit values to the pipeline.

5.11.1 Detailed Description

This class is used to render a scene using Direct3D 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

5.11.2 Member Function Documentation

5.11.2.1 AfterRender()

```
void RenderingEngine::RenderScene::AfterRender ( )
```

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

Call after rendering all your objects and text.

5.11.2.2 AfterRenderObjects()

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

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

Parameters

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

5.11.2.3 AfterRenderText()

```
void RenderingEngine::RenderScene::AfterRenderText ( )
```

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

Call after calling all the RenderText functions.

5.11.2.4 BeforeRenderObjects()

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

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

Call before calling the first RenderObjects function.

Parameters

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

5.11.2.5 BeforeRenderText()

```
void RenderingEngine::RenderScene::BeforeRenderText ( )
```

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

5.11.2.6 CompileShader() [1/2]

```
void RenderingEngine::RenderScene::CompileShader (
             std::wstring_view shaderKey,
             std::wstring_view filename,
             std::string_view entryPointName,
             std::string_view target )
```

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

Parameters

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

5.11.2.7 CompileShader() [2/2]

```
void RenderingEngine::RenderScene::CompileShader (
             unsigned int shaderKey,
             std::wstring_view filename,
             std::string_view entryPointName,
             std::string_view target )
```

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

Parameters

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

5.11.2.8 CopyDataIntoDynamicBuffer() [1/2]

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

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

Parameters

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

5.11.2.9 CopyDataIntoDynamicBuffer() [2/2]

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

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

Parameters

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

5.11.2.10 CreateDescriptorRange() [1/2]

```
void RenderingEngine::RenderScene::CreateDescriptorRange (
            std::wstring_view descriptorRangeKey,
            D3D12_DESCRIPTOR_RANGE_TYPE type,
            unsigned int numDescriptors,
            unsigned int shaderRegister,
            unsigned int registerSpace,
            unsigned int offset )
```

Creates a descriptor range and stores it in the array mapped to the specified *descriptorRangeKey*.

Parameters

in	*descriptorRangeKey*	The key to an array of descriptor ranges to store the descriptor range in.
in	*type*	The type of descriptor range.
in	*numDescriptors*	The number of descriptors in the range.
in	*shaderRegister*	The shader register the views are mapped to.
in	*registerSpace*	The space of the shader register.
in	*offset*	The offset in descriptors, from the start of the descriptor table.

5.11.2.11 CreateDescriptorRange() [2/2]

```
void RenderingEngine::RenderScene::CreateDescriptorRange (
            unsigned int descriptorRangeKey,
            D3D12_DESCRIPTOR_RANGE_TYPE type,
            unsigned int numDescriptors,
            unsigned int shaderRegister,
            unsigned int registerSpace,
            unsigned int offset )
```

Creates a descriptor range and stores it in the array mapped to the specified *descriptorRangeKey*.

Parameters

in	*descriptorRangeKey*	The key to an array of descriptor ranges to store the descriptor range in.
in	*type*	The type of descriptor range.
in	*numDescriptors*	The number of descriptors in the range.
in	*shaderRegister*	The shader register the views are mapped to.
in	*registerSpace*	The space of the shader register.
in	*offset*	The offset in descriptors, from the start of the descriptor table.

5.11.2.12 CreateDescriptorTable() [1/2]

```
void RenderingEngine::RenderScene::CreateDescriptorTable (
            std::wstring_view rootParameterKey,
            unsigned int descriptorRangeKey )
```

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

Parameters

in	*rootParameterKey*	The key to a mappped array to store the created root parameter in.
in	*descriptorRangeKey*	The key to an array of descriptor ranges.

5.11.2.13 CreateDescriptorTable() [2/2]

```
void RenderingEngine::RenderScene::CreateDescriptorTable (
            unsigned int rootParameterKey,
            unsigned int descriptorRangeKey )
```

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

Parameters

in	*rootParameterKey*	The key to a mappped array to store the created root parameter in.
in	*descriptorRangeKey*	The key to an array of descriptor ranges.

5.11.2.14 CreateDynamicBuffer() [1/4]

```
void RenderingEngine::RenderScene::CreateDynamicBuffer (
            std::wstring_view dynamicBufferKey,
            unsigned numBytes,
            const void * data,
            DXGI_FORMAT format )
```

Creates a dynamic index buffer.

The user can update the data on a per-frame basis.
If the specified key is already mapped to a dynamic buffer, this function does nothing.

Parameters

in	*dynamicBufferKey*	The key to map the dynamic buffer to.
in	*numBytes*	The number of bytes to allocate for the dynamic buffer.
in	*data*	The data you want to copy into the dynamic buffer. Pass in nullptr if you don't want to copy data into the buffer on creation.
in	*format*	The number of bytes to get from one element to the next element.

5.11.2.15 CreateDynamicBuffer() [2/4]

```
void RenderingEngine::RenderScene::CreateDynamicBuffer (
            std::wstring_view dynamicBufferKey,
            unsigned numBytes,
            const void * data,
            unsigned int stride )
```

Creates a dynamic vertex buffer or a dynamic constant buffer.

The user can update the data on a per-frame basis.
If the specified key is already mapped to a dynamic buffer, this function does nothing.

Parameters

in	*dynamicBufferKey*	The key to map the dynamic buffer to.
in	*numBytes*	The number of bytes to allocate for the dynamic buffer.
in	*data*	The data you want to copy into the dynamic buffer. Pass in nullptr if you don't want to copy data into the buffer on creation.
in	*stride*	The number of bytes to get from one element to the next element.

5.11.2.16 CreateDynamicBuffer() [3/4]

```
void RenderingEngine::RenderScene::CreateDynamicBuffer (
            unsigned int dynamicBufferKey,
            unsigned numBytes,
            const void * data,
            DXGI_FORMAT format )
```

Creates a dynamic index buffer.

The user can update the data on a per-frame basis.
If the specified key is already mapped to a dynamic buffer, this function does nothing.

Parameters

in	*dynamicBufferKey*	The key to map the dynamic buffer to.
in	*numBytes*	The number of bytes to allocate for the dynamic buffer.
in	*data*	The data you want to copy into the dynamic buffer. Pass in nullptr if you don't want to copy data into the buffer on creation.
in	*format*	The number of bytes to get from one element to the next element.

5.11.2.17 CreateDynamicBuffer() [4/4]

```
void RenderingEngine::RenderScene::CreateDynamicBuffer (
            unsigned int dynamicBufferKey,
            unsigned numBytes,
            const void * data,
            unsigned int stride )
```

Creates a dynamic vertex buffer or a dynamic constant buffer.

The user can update the data on a per-frame basis.
If the specified key is already mapped to a dynamic buffer, this function does nothing.

Parameters

in	*dynamicBufferKey*	The key to map the dynamic buffer to.
in	*numBytes*	The number of bytes to allocate for the dynamic buffer.
in	*data*	The data you want to copy into the dynamic buffer. Pass in nullptr if you don't want to copy data into the buffer on creation.
in	*stride*	The number of bytes to get from one element to the next element.

5.11.2.18 CreateInputElementDescription() [1/2]

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

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

Parameters

in	*key*	The key to a mapped array to store the created input element description.
in	*semanticName*	The name of the application variable linked to a shader variable.
in	*semanticIndex*	The index to attach to the semanticName.
in	*format*	The data type of input element being described.
in	*inputSlot*	The input slot the input element will come from.
in	*byteOffset*	The offset in bytes to get to the input element being described.
	[in,optional]	inputSlotClass The data class for an input slot. Used for instancing.
	[in,optional]	instanceStepRate The number of instances to render. Used for instancing.

5.11.2.19 CreateInputElementDescription() [2/2]

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

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

Parameters

in	*key*	The key to a mapped array to store the created input element description.
in	*semanticName*	The name of the application variable linked to a shader variable.
in	*semanticIndex*	The index to attach to the semanticName.
in	*format*	The data type of input element being described.
in	*inputSlot*	The input slot the input element will come from.
in	*byteOffset*	The offset in bytes to get to the input element being described.
	[in,optional]	inputSlotClass The data class for an input slot. Used for instancing.
	[in,optional]	instanceStepRate The number of instances to render. Used for instancing.

5.11.2.20 CreatePSO() [1/2]

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

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

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

Parameters

in	*psoKey*	The key to map the created PSO to.
in	*fillMode*	The fill mode to use when rendering triangles. Use D3D12_FILL_MODE_WIREFRAME for wireframe and D3D12_FILL_MODE_SOLID for solid.
in	*enableMultisample*	Pass in TRUE to use multi-sampling, FALSE otherwise.

Parameters

in	*vsKey*	A key to a mapped vertex shader.
in	*psKey*	A key to a mapped pixel shader.
in	*inputElementDescriptionsKey*	A key to a mapped array of input element descriptions for the specified vertex and pixel shaders.
in	*rootSigKey*	A key to a mapped root signature.
in	*primitiveType*	The type of primitive to connect vertices into.
	[in,optional]	sampleCount The number of samples. If enableMultiSample is TRUE pass in 4. All other values will cause an error.

5.11.2.21 CreatePSO() [2/2]

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

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

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

Parameters

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

5.11.2.22 CreateRootConstants() [1/2]

```
void RenderingEngine::RenderScene::CreateRootConstants (
            std::wstring_view rootParameterKey,
            unsigned int shaderRegister,
            unsigned int numValues )
```

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

Parameters

in	*rootParameterKey*	The key to a mappped array to store the created root parameter in.
in	*shaderRegister*	The register where constant data will be stored.
in	*numValues*	The number of 32-bit values.

5.11.2.23 CreateRootConstants() [2/2]

```
void RenderingEngine::RenderScene::CreateRootConstants (
            unsigned int rootParameterKey,
            unsigned int shaderRegister,
            unsigned int numValues )
```

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

Parameters

in	*rootParameterKey*	The key to a mappped array to store the created root parameter in.
in	*shaderRegister*	The register where constant data will be stored.
in	*numValues*	The number of 32-bit values.

5.11.2.24 CreateRootDescriptor() [1/2]

```
void RenderingEngine::RenderScene::CreateRootDescriptor (
            std::wstring_view rootParameterKey,
            unsigned int shaderRegister )
```

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

Parameters

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

5.11.2.25 CreateRootDescriptor() [2/2]

```
void RenderingEngine::RenderScene::CreateRootDescriptor (
            unsigned int rootParameterKey,
            unsigned int shaderRegister )
```

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

Parameters

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

5.11.2.26 CreateRootSignature() [1/4]

```
void RenderingEngine::RenderScene::CreateRootSignature (
            std::wstring_view rootSigKey,
            std::wstring_view rootParametersKey )
```

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

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

Parameters

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

5.11.2.27 CreateRootSignature() [2/4]

```
void RenderingEngine::RenderScene::CreateRootSignature (
            std::wstring_view rootSigKey,
            std::wstring_view rootParametersKey,
            std::wstring_view staticsSamplerKey )
```

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

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

Parameters

in	*rootSigKey*	The key to map the created root signature to.
in	*rootParameterKey*	The key to a mapped array of root parameters.
in	*numStaticSamplers*	The number of static samplers.
in	*staticsSamplerKey*	The key to an array of static samplers.

5.11.2.28 CreateRootSignature() [3/4]

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

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

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

Parameters

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

5.11.2.29 CreateRootSignature() [4/4]

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

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

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

Parameters

in	*rootSigKey*	The key to map the created root signature to.
in	*rootParameterKey*	The key to a mapped array of root parameters.
in	*numStaticSamplers*	The number of static samplers.
in	*staticsSamplerKey*	The key to an array of static samplers.

5.11.2.30 CreateStaticBuffer() [1/6]

```
void RenderingEngine::RenderScene::CreateStaticBuffer (
            std::wstring_view staticBufferKey,
            const void * data,
            unsigned numBytes,
            DXGI_FORMAT format )
```

Creates a static index buffer and stores the specified *data* in the buffer.

The user cannot update/change the data once it is stored in the buffer.
If the specified key is already mapped to a static buffer, this function does nothing.

Parameters

in	*staticBufferKey*	The key to map the static buffer to.
in	*numBytes*	The number of bytes to allocate for the static buffer.
in	*format*	The number of bytes to get from one element to the next element.

5.11.2.31 CreateStaticBuffer() [2/6]

```
void RenderingEngine::RenderScene::CreateStaticBuffer (
            std::wstring_view staticBufferKey,
            const void * data,
            unsigned numBytes,
            unsigned int stride )
```

Creates a static vertex buffer and stores the specified *data* in the buffer.

The user cannot update/change the data once it is stored in the buffer.
If the specified key is already mapped to a static buffer, this function does nothing.

Parameters

in	*staticBufferKey*	The key to map the static buffer to.
in	*numBytes*	The number of bytes to allocate for the static buffer.
in	*stride*	The number of bytes to get from one element to the next element.

5.11.2.32 CreateStaticBuffer() [3/6]

```
void RenderingEngine::RenderScene::CreateStaticBuffer (
            std::wstring_view staticBufferKey,
            const wchar_t * filename,
            unsigned int texType,
            unsigned int index )
```

Creates a static texture buffer, stores the data from the file into the buffer and creates a view of the texture.

The user cannot update/change the data once it is stored in the buffer.
If the specified key is already mapped to a static buffer, this function does nothing.

Parameters

in	*staticBufferKey*	The key to map the static buffer to.
in	*numBytes*	The number of bytes to allocate for the static buffer.
in	*filename*	The filename of the texture.
in	*texType*	The type of texture. Pass in FARender::Tex2D for a 2D texture or FARender::Tex2D_MS for a multi-sampled 2D texture.
in	*index*	Where to store the description (view) of the texture in a shader resource view heap.

5.11.2.33 CreateStaticBuffer() [4/6]

```
void RenderingEngine::RenderScene::CreateStaticBuffer (
            unsigned int staticBufferKey,
            const void * data,
            unsigned numBytes,
            DXGI_FORMAT format )
```

Creates a static index buffer and stores the specified *data* in the buffer.

The user cannot update/change the data once it is stored in the buffer.
If the specified key is already mapped to a static buffer, this function does nothing.

Parameters

in	*staticBufferKey*	The key to map the static buffer to.
in	*numBytes*	The number of bytes to allocate for the static buffer.
in	*format*	The number of bytes to get from one element to the next element.

5.11.2.34 CreateStaticBuffer() [5/6]

```
void RenderingEngine::RenderScene::CreateStaticBuffer (
            unsigned int staticBufferKey,
            const void * data,
            unsigned numBytes,
            unsigned int stride )
```

Creates a static vertex buffer and stores the specified *data* in the buffer.

The user cannot update/change the data once it is stored in the buffer.
If the specified key is already mapped to a static buffer, this function does nothing.

Parameters

in	*staticBufferKey*	The key to map the static buffer to.
in	*numBytes*	The number of bytes to allocate for the static buffer.
in	*stride*	The number of bytes to get from one element to the next element.

5.11.2.35 CreateStaticBuffer() [6/6]

```
void RenderingEngine::RenderScene::CreateStaticBuffer (
            unsigned int staticBufferKey,
            const wchar_t * filename,
            unsigned int texType,
            unsigned int index )
```

Creates a static texture buffer, stores the data from the file into the buffer and creates a view of the texture.

The user cannot update/change the data once it is stored in the buffer.
If the specified key is already mapped to a static buffer, this function does nothing.

Parameters

in	*staticBufferKey*	The key to map the static buffer to.
in	*numBytes*	The number of bytes to allocate for the static buffer.
in	*filename*	The filename of the texture.
in	*texType*	The type of texture. Pass in FARender::Tex2D for a 2D texture or FARender::Tex2D_MS for a multi-sampled 2D texture.
in	*index*	Where to store the description (view) of the texture in a shader resource view heap.

5.11.2.36 CreateStaticSampler() [1/2]

```
void RenderingEngine::RenderScene::CreateStaticSampler (
            std::wstring_view staticSamplerKey,
            D3D12_FILTER filter,
            D3D12_TEXTURE_ADDRESS_MODE u,
            D3D12_TEXTURE_ADDRESS_MODE v,
            D3D12_TEXTURE_ADDRESS_MODE w,
            unsigned int shaderRegister )
```

Creates a static sampler and stores in an an array mapped to the specified key.

Parameters

in	*staticSamplerKey*	The key to an array of static samplers.
in	*filter*	The filtering method to use when sampling a texture.
in	*u*	The address mode for the u texture coordinate.
in	*v*	The address mode for the v texture coordinate.
in	*w*	The address mode for the w texture coordinate.
in	*shaderRegister*	The register the sampler is linked to.

5.11.2.37 CreateStaticSampler() [2/2]

```
void RenderingEngine::RenderScene::CreateStaticSampler (
            unsigned int staticSamplerKey,
            D3D12_FILTER filter,
            D3D12_TEXTURE_ADDRESS_MODE u,
            D3D12_TEXTURE_ADDRESS_MODE v,
            D3D12_TEXTURE_ADDRESS_MODE w,
            unsigned int shaderRegister )
```

Creates a static sampler and stores in an an array mapped to the specified key.

Parameters

in	*staticSamplerKey*	The key to an array of static samplers.
in	*filter*	The filtering method to use when sampling a texture.
in	*u*	The address mode for the u texture coordinate.
in	*v*	The address mode for the v texture coordinate.
in	*w*	The address mode for the w texture coordinate.
in	*shaderRegister*	The register the sampler is linked to.

5.11.2.38 CreateTextureViewHeap()

```
void RenderingEngine::RenderScene::CreateTextureViewHeap (
            unsigned int numDescriptors )
```

Creates a descriptor heap to store views of textures.

Parameters

in	*numDescriptors*	The number of views to be stored in the heap.

5.11.2.39 ExecuteAndFlush()

```
void RenderingEngine::RenderScene::ExecuteAndFlush ( )
```

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

5.11.2.40 LinkDynamicBuffer() [1/2]

```
void RenderingEngine::RenderScene::LinkDynamicBuffer (
            unsigned int bufferType,
            std::wstring_view dynamicBufferKey,
            unsigned int indexConstantData = 0,
            unsigned int rootParameterIndex = 0 )
```

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

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

Parameters

in		

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

Parameters

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

The parameters indexConstantData rootParameterIndex are used if the dynamic buffer is a constant buffer.

5.11.2.41 LinkDynamicBuffer() [2/2]

```
void RenderingEngine::RenderScene::LinkDynamicBuffer (
            unsigned int bufferType,
            unsigned int dynamicBufferKey,
            unsigned int indexConstantData = 0,
            unsigned int rootParameterIndex = 0 )
```

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

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

Parameters

in		

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

Parameters

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

The parameters indexConstantData rootParameterIndex are used if the dynamic buffer is a constant buffer.

5.11.2.42 LinkPSOAndRootSignature() [1/2]

```
void RenderingEngine::RenderScene::LinkPSOAndRootSignature (
            std::wstring_view psoKey,
            std::wstring_view rootSigKey )
```

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

Parameters

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

5.11.2.43 LinkPSOAndRootSignature() [2/2]

```
void RenderingEngine::RenderScene::LinkPSOAndRootSignature (
            unsigned int psoKey,
            unsigned int rootSigKey )
```

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

Parameters

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

5.11.2.44 LinkStaticBuffer() [1/2]

```
void RenderingEngine::RenderScene::LinkStaticBuffer (
            unsigned int bufferType,
            std::wstring_view staticBufferKey )
```

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

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

Parameters

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

5.11.2.45 LinkStaticBuffer() [2/2]

```
void RenderingEngine::RenderScene::LinkStaticBuffer (
            unsigned int bufferType,
            unsigned int staticBufferKey )
```

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

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

Parameters

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

5.11.2.46 LinkTexture() [1/2]

```
void RenderingEngine::RenderScene::LinkTexture (
            unsigned int rootParameterIndex )
```

Links the set of textures in the descriptor table to the pipeline.

Parameters

in	*rootParameterIndex*	The index of the root parameter in the root signature that has the register the texture will be stored in.

5.11.2.47 LinkTexture() [2/2]

```
void RenderingEngine::RenderScene::LinkTexture (
            unsigned int rootParameterIndex,
            unsigned int textureViewIndex )
```

Links a texture to the pipeline.

Parameters

in	*rootParameterIndex*	The index of the root parameter in the root signature that has the register the texture will be stored in.
in	*textureViewIndex*	The index of the view to the texture in a shader resource view heap.

5.11.2.48 LinkTextureViewHeap()

```
void RenderingEngine::RenderScene::LinkTextureViewHeap ( )
```

Links the texture view heap to the pipeline.

5.11.2.49 LoadShader() [1/2]

```
void RenderingEngine::RenderScene::LoadShader (
            std::wstring_view shaderKey,
            std::wstring_view filename )
```

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

Parameters

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

5.11.2.50 LoadShader() [2/2]

```
void RenderingEngine::RenderScene::LoadShader (
            unsigned int shaderKey,
            std::wstring_view filename )
```

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

Parameters

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

5.11.2.51 RemoveShader() [1/2]

```
void RenderingEngine::RenderScene::RemoveShader (
            std::wstring_view shaderKey )
```

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

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

5.11.2.52 RemoveShader() [2/2]

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

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

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

5.11.2.53 RenderObject()

```
void RenderingEngine::RenderScene::RenderObject (
            unsigned int indexCount,
            unsigned int locationFirstIndex,
            int indexOfFirstVertex,
            D3D_PRIMITIVE_TOPOLOGY primitive )
```

Renders an object with the specified draw arguments.

Call in between a BeforeRenderObjects function and a AfterRenderObjects function.

Ex.
BeforeRenderObjects()
RenderObject()
RenderObject()
AfterRenderObjects()

Parameters

in	*indexCount*	The number of indices used to connect the vertices of the objects.
in	*locationFirstIndex*	The location of the first index of the object in an index buffer.
in	*indexOfFirstVertex*	The index of the first vertex of the object in a vertex buffer.
in	*primitive*	The primitve used to render the object.

5.11.2.54 RenderText()

```
void RenderingEngine::RenderScene::RenderText (
            const vec4 & textLocation,
            const Color & textColor,
            float textSize,
            const std::wstring & textString,
            DWRITE_PARAGRAPH_ALIGNMENT alignment = DWRITE_PARAGRAPH_ALIGNMENT_CENTER )
```

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

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

Parameters

in	*textLocation*	The location of the text. The first 2 values are the top left corner and last two values are bottom right corner.
in	*textColor*	The color of the text.
in	*textSize*	The size of the size.
in	*textString*	The text to render.
in	*alignment*	Where you want the text to start at in the rectangle.

5.11.2.55 Resize()

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

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

Parameters

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

5.11.2.56 SetConstants()

```
void RenderingEngine::RenderScene::SetConstants (
            unsigned int rootParameterIndex,
            unsigned int numValues,
            void * data,
            unsigned int index )
```

Links an array of 32-bit values to the pipeline.

Parameters

in	*rootParameterIndex*	The index of the root parameter in the root signature that has the register the texture will be stored in.
in	*numValues*	The number of 32-bit values.
in	*data*	Pointer to an array of 32-bit values.
in	*index*	The index of the the first 32-bit value in the hlsl constant buffer.

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

- RenderScene.h

5.12 RenderingEngine::RenderTargetBuffer Class Reference

A wrapper for render target buffer resources. Uses DirectD 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

`#include "Buffer.h"`

Public Member Functions

- **RenderTargetBuffer** (const RenderTargetBuffer &)=delete
- RenderTargetBuffer & **operator=** (const RenderTargetBuffer &)=delete
- RenderTargetBuffer ()

    Creates a render target buffer object. No memory is allocated. Called the CreateRenderTargetBufferAndView() function to allocate memory for the buffer.
- RenderTargetBuffer (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::↩ ComPtr< ID3D12DescriptorHeap > &rtvHeap, unsigned int index, unsigned int rtvSize, unsigned int width, unsigned int height, DXGI_FORMAT format=DXGI_FORMAT_R8G8B8A8_UNORM, unsigned int sample↩ Count=1)

    Creates the render target buffer and view.
- DXGI_FORMAT GetRenderTargetFormat () const

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

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

    Returns a constant reference to the render target buffer.
- void CreateRenderTargetBufferAndView (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &rtvHeap, unsigned int index, unsigned int rtvSize, unsigned int width, unsigned int height, DXGI_FORMAT format=DXGI_FORMAT_R8G8B8A8_UNORM, unsigned int sampleCount=1)

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

    Frees the memory of the buffer.
- void ClearRenderTargetBuffer (const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &rtvHeap, unsigned int index↩ OfView, unsigned int rtvSize, const float ∗clearValue)

    Clears the render target buffer with the specified clear value.

5.12.1 Detailed Description

A wrapper for render target buffer resources. Uses DirectD 12 API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 RenderTargetBuffer() `[1/2]`

```
RenderingEngine::RenderTargetBuffer::RenderTargetBuffer ( )
```

Creates a render target buffer object. No memory is allocated. Called the CreateRenderTargetBufferAndView() function to allocate memory for the buffer.

5.12.2.2 RenderTargetBuffer() [2/2]

```
RenderingEngine::RenderTargetBuffer::RenderTargetBuffer (
            const Microsoft::WRL::ComPtr< ID3D12Device > & device,
            const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & rtvHeap,
            unsigned int index,
            unsigned int rtvSize,
            unsigned int width,
            unsigned int height,
            DXGI_FORMAT format = DXGI_FORMAT_R8G8B8A8_UNORM,
            unsigned int sampleCount = 1 )
```

Creates the render target buffer and view.

Parameters

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

5.12.3 Member Function Documentation

5.12.3.1 ClearRenderTargetBuffer()

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

Clears the render target buffer with the specified clear value.

Parameters

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

5.12.3.2 CreateRenderTargetBufferAndView()

```
void RenderingEngine::RenderTargetBuffer::CreateRenderTargetBufferAndView (
            const Microsoft::WRL::ComPtr< ID3D12Device > & device,
            const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & rtvHeap,
            unsigned int index,
            unsigned int rtvSize,
            unsigned int width,
            unsigned int height,
            DXGI_FORMAT format = DXGI_FORMAT_R8G8B8A8_UNORM,
            unsigned int sampleCount = 1 )
```

Creates the render target buffer and view.

Parameters

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

5.12.3.3 GetRenderTargetBuffer() [1/2]

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

Returns a reference to the render target buffer.

5.12.3.4 GetRenderTargetBuffer() [2/2]

```
const Microsoft::WRL::ComPtr< ID3D12Resource > & RenderingEngine::RenderTargetBuffer::Get↩
RenderTargetBuffer ( ) const
```

Returns a constant reference to the render target buffer.

5.12.3.5 GetRenderTargetFormat()

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

Returns the format of the render target buffer.

5.12.3.6 ReleaseBuffer()

```
void RenderingEngine::RenderTargetBuffer::ReleaseBuffer ( )
```

Frees the memory of the buffer.

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

- Buffer.h

5.13 RenderingEngine::StaticBuffer Class Reference

This class stores data in a Direct3D 12 default buffer. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

```
#include "Buffer.h"
```

Public Member Functions

- **StaticBuffer** (const StaticBuffer &)=delete
- StaticBuffer & **operator=** (const StaticBuffer &)=delete
- StaticBuffer ()

  Creates a static buffer object. No memory is allocated for the buffer. Call one of the CreateStaticBuffer() functions to allocate memory for the buffer and store data in the buffer.
- StaticBuffer (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const void ∗data, unsigned int numBytes, unsigned int stride)

  Creates a static vertex buffer and stores all of the specified data in the buffer.
- StaticBuffer (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const void ∗data, unsigned int numBytes, DXGI_FORMAT format)

  Creates a static index buffer and stores all of the specified data in the buffer.
- StaticBuffer (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > &commandList, const wchar_t ∗filename)

  Creates a static texture buffer and stores all of the data from the file in the buffer.
- void CreateStaticBuffer (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL↩ ::ComPtr< ID3D12GraphicsCommandList > &commandList, const void ∗data, unsigned int numBytes, unsigned int stride)

  Creates a static vertex buffer and stores all of the specified data in the buffer.
- void CreateStaticBuffer (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL↩ ::ComPtr< ID3D12GraphicsCommandList > &commandList, const void ∗data, unsigned int numBytes, DXGI_FORMAT format)

  Creates a static index buffer and stores all of the specified data in the buffer.
- void CreateStaticBuffer (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL↩ ::ComPtr< ID3D12GraphicsCommandList > &commandList, const wchar_t ∗filename)

  Creates a static texture buffer and stores all of the data from the file in the buffer.
- const D3D12_VERTEX_BUFFER_VIEW GetVertexBufferView () const

  Returns the vertex buffer view of the static buffer.
- const D3D12_INDEX_BUFFER_VIEW GetIndexBufferView () const

  Returns the index buffer view of the static buffers.

- void CreateTexture2DView (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::↩ WRL::ComPtr< ID3D12DescriptorHeap > &srvHeap, unsigned int srvSize, unsigned int index)

    Creates a 2D texture view and stores it in the specified heap.
- void CreateTexture2DMSView (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft↩ ::WRL::ComPtr< ID3D12DescriptorHeap > &srvHeap, unsigned int srvSize, unsigned int index)

    Creates a multi-sampled 2D texture view and stores it in the specified heap.
- void ReleaseBuffer ()

    Frees the static buffer memory.

5.13.1 Detailed Description

This class stores data in a Direct3D 12 default buffer. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 StaticBuffer() `[1/4]`

```
RenderingEngine::StaticBuffer::StaticBuffer ( )
```

Creates a static buffer object. No memory is allocated for the buffer. Call one of the CreateStaticBuffer() functions to allocate memory for the buffer and store data in the buffer.

5.13.2.2 StaticBuffer() `[2/4]`

```
RenderingEngine::StaticBuffer::StaticBuffer (
            const Microsoft::WRL::ComPtr< ID3D12Device > & device,
            const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & commandList,
            const void * data,
            unsigned int numBytes,
            unsigned int stride )
```

Creates a static vertex buffer and stores all of the specified data in the buffer.

Parameters

in	*device*	A Direct3D 12 device.
in	*commadList*	A Direct3D 12 graphics command list.
in	*data*	The data to store in the static vertex buffer.
in	*numBytes*	The number of bytes to store in the static vertex buffer.
in	*stride*	The number of bytes to get from one element to the next element.

5.13.2.3 StaticBuffer() [3/4]

```
RenderingEngine::StaticBuffer::StaticBuffer (
            const Microsoft::WRL::ComPtr< ID3D12Device > & device,
            const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & commandList,
            const void * data,
            unsigned int numBytes,
            DXGI_FORMAT format )
```

Creates a static index buffer and stores all of the specified data in the buffer.

Parameters

in	*device*	A Direct3D 12 device.
in	*commadList*	A Direct3D 12 graphics command list.
in	*data*	The data to store in the static index buffer.
in	*numBytes*	The number of bytes to store in the static index buffer.
in	*format*	The number of bytes to get from one element to the next element.

5.13.2.4 StaticBuffer() [4/4]

```
RenderingEngine::StaticBuffer::StaticBuffer (
            const Microsoft::WRL::ComPtr< ID3D12Device > & device,
            const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & commandList,
            const wchar_t * filename )
```

Creates a static texture buffer and stores all of the data from the file in the buffer.

Parameters

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

5.13.3 Member Function Documentation

5.13.3.1 CreateStaticBuffer() [1/3]

```
void RenderingEngine::StaticBuffer::CreateStaticBuffer (
            const Microsoft::WRL::ComPtr< ID3D12Device > & device,
            const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & commandList,
            const void * data,
            unsigned int numBytes,
            DXGI_FORMAT format )
```

Creates a static index buffer and stores all of the specified data in the buffer.

Parameters

in	*device*	A Direct3D 12 device.
in	*commadList*	A Direct3D 12 graphics command list.
in	*data*	The data to store in the static index buffer.
in	*numBytes*	The number of bytes to store in the static index buffer.
in	*format*	The number of bytes to get from one element to the next element.

5.13.3.2 CreateStaticBuffer() [2/3]

```
void RenderingEngine::StaticBuffer::CreateStaticBuffer (
            const Microsoft::WRL::ComPtr< ID3D12Device > & device,
            const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & commandList,
            const void * data,
            unsigned int numBytes,
            unsigned int stride )
```

Creates a static vertex buffer and stores all of the specified data in the buffer.

Parameters

in	*device*	A Direct3D 12 device.
in	*commadList*	A Direct3D 12 graphics command list.
in	*data*	The data to store in the static vertex buffer.
in	*numBytes*	The number of bytes to store in the static vertex buffer.
in	*stride*	The number of bytes to get from one element to the next element.

5.13.3.3 CreateStaticBuffer() [3/3]

```
void RenderingEngine::StaticBuffer::CreateStaticBuffer (
            const Microsoft::WRL::ComPtr< ID3D12Device > & device,
            const Microsoft::WRL::ComPtr< ID3D12GraphicsCommandList > & commandList,
            const wchar_t * filename )
```

Creates a static texture buffer and stores all of the data from the file in the buffer.

Parameters

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

5.13.3.4 CreateTexture2DMSView()

```
void RenderingEngine::StaticBuffer::CreateTexture2DMSView (
            const Microsoft::WRL::ComPtr< ID3D12Device > & device,
            const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & srvHeap,
            unsigned int srvSize,
            unsigned int index )
```

Creates a multi-sampled 2D texture view and stores it in the specified heap.

Parameters

in	*device*	A Direct3D 12 device.
in	*srvHeap*	A shader resource view heap.
in	*srvSize*	The size of a shader resource view.
in	*index*	The index of where to store the texture view in the shader resource view heap.

5.13.3.5 CreateTexture2DView()

```
void RenderingEngine::StaticBuffer::CreateTexture2DView (
            const Microsoft::WRL::ComPtr< ID3D12Device > & device,
            const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & srvHeap,
            unsigned int srvSize,
            unsigned int index )
```

Creates a 2D texture view and stores it in the specified heap.

Parameters

in	*device*	A Direct3D 12 device.
in	*srvHeap*	A shader resource view heap.
in	*srvSize*	The size of a shader resource view.
in	*index*	The index of where to store the texture view in the shader resource view heap.

5.13.3.6 GetIndexBufferView()

```
const D3D12_INDEX_BUFFER_VIEW RenderingEngine::StaticBuffer::GetIndexBufferView ( ) const
```

Returns the index buffer view of the static buffers.

5.13.3.7 GetVertexBufferView()

```
const D3D12_VERTEX_BUFFER_VIEW RenderingEngine::StaticBuffer::GetVertexBufferView ( ) const
```

Returns the vertex buffer view of the static buffer.

5.13.3.8 ReleaseBuffer()

```
void RenderingEngine::StaticBuffer::ReleaseBuffer ( )
```

Frees the static buffer memory.

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

- Buffer.h

5.14 RenderingEngine::SwapChain Class Reference

A wrapper for swap chain resources. Uses DirectD 12 API and DXGI API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

```
#include "SwapChain.h"
```

Public Member Functions

- **SwapChain** (const SwapChain &)=delete
- SwapChain & **operator=** (const SwapChain &)=delete
- SwapChain ()

  Creates a swap chain object. Call the CreateSwapChain() function to create a swap chain.
- SwapChain (const Microsoft::WRL::ComPtr< IDXGIFactory4 > &dxgiFactory, const Microsoft::WRL↩ ::ComPtr< ID3D12CommandQueue > &commandQueue, HWND windowHandle, DXGI_FORMAT rtFormat=DXGI_FORMAT_R8G8B8A8_UNORM, DXGI_FORMAT dsFormat=DXGI_FORMAT_D24_↩ UNORM_S8_UINT, unsigned int numRenderTargetBuffers=2)

  Creates a swap chain.
- void CreateSwapChain (const Microsoft::WRL::ComPtr< IDXGIFactory4 > &dxgiFactory, const Microsoft↩ ::WRL::ComPtr< ID3D12CommandQueue > &commandQueue, HWND windowHandle, DXGI_FORMAT rtFormat=DXGI_FORMAT_R8G8B8A8_UNORM, DXGI_FORMAT dsFormat=DXGI_FORMAT_D24_↩ UNORM_S8_UINT, unsigned int numRenderTargetBuffers=2)

  Creates a swap chain.
- const Microsoft::WRL::ComPtr< ID3D12Resource > & GetCurrentBackBuffer () const

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

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

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

  Returns the format of the swap chain.

- DXGI_FORMAT GetDepthStencilFormat () const

  Returns the format of the depth stencil buffer.
- const std::vector< std::unique_ptr< RenderTargetBuffer > > & **GetRenderTargetBuffers** ()
- void ReleaseBuffers ()

  Frees the memory of the render target and depth stencil buffers.
- void CreateRenderTargetBuffersAndViews (const Microsoft::WRL::ComPtr< ID3D12Device > &device, const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > &rtvHeap, unsigned int index, unsigned int rtvSize, unsigned width, unsigned height)

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

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

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

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

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

  Swaps the front and back buffers.

5.14.1 Detailed Description

A wrapper for swap chain resources. Uses DirectD 12 API and DXGI API. The copy constructor and assignment operators are explicitly deleted. This makes this class non-copyable.

5.14.2 Constructor & Destructor Documentation

5.14.2.1 SwapChain() [1/2]

```
RenderingEngine::SwapChain::SwapChain ( )
```

Creates a swap chain object. Call the CreateSwapChain() function to create a swap chain.

5.14.2.2 SwapChain() [2/2]

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

Creates a swap chain.

Parameters

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

5.14.3 Member Function Documentation

5.14.3.1 ClearCurrentBackBuffer()

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

Clears the current render target buffer.

Parameters

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

5.14.3.2 ClearDepthStencilBuffer()

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

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

Parameters

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

5.14.3.3 CreateDepthStencilBufferAndView()

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

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

Parameters

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

5.14.3.4 CreateRenderTargetBuffersAndViews()

```
void RenderingEngine::SwapChain::CreateRenderTargetBuffersAndViews (
            const Microsoft::WRL::ComPtr< ID3D12Device > & device,
            const Microsoft::WRL::ComPtr< ID3D12DescriptorHeap > & rtvHeap,
            unsigned int index,
            unsigned int rtvSize,
            unsigned width,
            unsigned height )
```

Creates the swap chains render target buffers and views to them.

Parameters

in	*device*	A Direct3D 12 device.
in	*rtvHeap*	A descriptor heap for storing render target descriptors.

Parameters

in	*index*	The index of where to store the created descriptor in the descriptor heap.
in	*rtvSize*	The size of a render target descriptor.
in	*width*	The width of the render target buffers.
in	*height*	The height of the render target buffers.

5.14.3.5 CreateSwapChain()

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

Creates a swap chain.

Parameters

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

5.14.3.6 GetBackBufferFormat()

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

Returns the format of the swap chain.

5.14.3.7 GetCurrentBackBuffer()

```
const Microsoft::WRL::ComPtr< ID3D12Resource > & RenderingEngine::SwapChain::GetCurrentBack←
Buffer ( ) const
```

Returns a constant reference to the current render target buffer.

5.14.3.8 GetCurrentBackBufferIndex()

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

Returns the current back buffer index.

5.14.3.9 GetDepthStencilFormat()

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

Returns the format of the depth stencil buffer.

5.14.3.10 GetNumRenderTargetBuffers()

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

Returns the number of swap chain buffers.

5.14.3.11 Present()

```
void RenderingEngine::SwapChain::Present ( )
```

Swaps the front and back buffers.

5.14.3.12 ReleaseBuffers()

```
void RenderingEngine::SwapChain::ReleaseBuffers ( )
```

Frees the memory of the render target and depth stencil buffers.

5.14.3.13 Transition()

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

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

Parameters

`in`	*commandList*	A Direct3D 12 graphics command list.

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

- SwapChain.h

5.15 RenderingEngine::Text Class Reference

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

```
#include "Text.h"
```

Public Member Functions

- Text (const vec4 &textLocation, const std::wstring &textString, float textSize, const Color &textColor)

  Constructs a Text object.
- const vec4 & GetTextLocation () const

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

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

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

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

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

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

  Changes the text string to the specified textString.
- void SetTextLocation (const vec4 &textLocation)

  Changes the text location to the specified textLocation.

5.15.1 Detailed Description

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

5.15.2 Constructor & Destructor Documentation

5.15.2.1 Text()

```
RenderingEngine::Text::Text (
            const vec4 & textLocation,
            const std::wstring & textString,
            float textSize,
            const Color & textColor )
```

Constructs a Text object.

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

Parameters

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

5.15.3 Member Function Documentation

5.15.3.1 GetTextColor()

```
const Color & RenderingEngine::Text::GetTextColor ( ) const
```

Returns a constant reference to the text color.

5.15.3.2 GetTextLocation()

```
const vec4 & RenderingEngine::Text::GetTextLocation ( ) const
```

Returns a constant reference to the text location.

5.15.3.3 GetTextSize()

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

Returns the text size.

5.15.3.4 GetTextString()

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

Returns a constant reference to the text string.

5.15.3.5 SetTextColor()

```
void RenderingEngine::Text::SetTextColor (
             const Color & textColor )
```

Changes the text color to the specified *textColor*.

5.15.3.6 SetTextLocation()

```
void RenderingEngine::Text::SetTextLocation (
             const vec4 & textLocation )
```

Changes the text location to the specified *textLocation*.

5.15.3.7 SetTextSize()

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

Changes the text size to the specified *textSize*.

5.15.3.8 SetTextString()

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

Changes the text string to the specified *textString*.

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

- Text.h

5.16 RenderingEngine::TextResources Class Reference

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

```
#include "TextResources.h"
```

Public Member Functions

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

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

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

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

  Resets the text buffers.
- void ResizeBuffers (const std::vector< std::unique_ptr< RenderTargetBuffer > > &renderTargetBuffers, HWND windowHandle)

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

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

  Executes text commands.

5.16.1 Detailed Description

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

5.16.2 Constructor & Destructor Documentation

5.16.2.1 TextResources()

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

Initializes the text resources.

Parameters

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

5.16.3 Member Function Documentation

5.16.3.1 AfterRenderText()

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

Executes text commands.

Parameters

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

5.16.3.2 BeforeRenderText()

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

Prepares to render text.

Parameters

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

5.16.3.3 GetDirect2DDeviceContext()

```
const Microsoft::WRL::ComPtr< ID2D1DeviceContext > & RenderingEngine::TextResources::Get←
Direct2DDeviceContext ( ) const
```

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

5.16.3.4 GetDirectWriteFactory()

```
const Microsoft::WRL::ComPtr< IDWriteFactory > & RenderingEngine::TextResources::GetDirect←
WriteFactory ( ) const
```

Returns a constant reference to the direct direct write factory.

5.16.3.5 ResetBuffers()

```
void RenderingEngine::TextResources::ResetBuffers ( )
```

Resets the text buffers.

5.16.3.6 ResizeBuffers()

```
void RenderingEngine::TextResources::ResizeBuffers (
             const std::vector< std::unique_ptr< RenderTargetBuffer > > & renderTarget←
Buffers,
             HWND windowHandle )
```

Resizes the buffers.

Parameters

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

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

- TextResources.h

5.17 RenderingEngine::Time Struct Reference

A struct that holds the properties for time.

```
#include "GameTime.h"
```

Public Attributes

- __int64 **previousTime** = 0
- __int64 **currentTime** = 0
- double **deltaTime** = 0
- double **secondsPerCount** = 0.0
- bool **stopped** = false

5.17.1 Detailed Description

A struct that holds the properties for time.

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

- GameTime.h

5.18 RenderingEngine::Window Struct Reference

The window struct is used to make a Window using Win32 API.

```
#include "Window.h"
```

Public Attributes

- HWND **windowHandle**
- WNDCLASSEX **windowClass**

5.18.1 Detailed Description

The window struct is used to make a Window using Win32 API.

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

- Window.h

Chapter 6

File Documentation

6.1 Buffer.h

```
#pragma once

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

namespace RenderingEngine
{
    enum BufferTypes { VERTEX_BUFFER, INDEX_BUFFER, CONSTANT_BUFFER, TEXTURE_BUFFER };
    enum TextureTypes { TEX2D, TEX2D_MS };

    class RenderTargetBuffer
    {
    public:

        //No copying
        RenderTargetBuffer(const RenderTargetBuffer&) = delete;
        RenderTargetBuffer& operator=(const RenderTargetBuffer&) = delete;

        RenderTargetBuffer();

        RenderTargetBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
            const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int index, unsigned int rtvSize,
            unsigned int width, unsigned int height, DXGI_FORMAT format = DXGI_FORMAT_R8G8B8A8_UNORM, unsigned int sampleCount = 1);

        DXGI_FORMAT GetRenderTargetFormat() const;

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

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

        void CreateRenderTargetBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
            const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& rtvHeap, unsigned int index, unsigned int rtvSize,
            unsigned int width, unsigned int height, DXGI_FORMAT format = DXGI_FORMAT_R8G8B8A8_UNORM, unsigned int sampleCount = 1);

        void ReleaseBuffer();

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

    private:
        Microsoft::WRL::ComPtr<ID3D12Resource> mRenderTargetBuffer;
    };

    class DepthStencilBuffer
    {
    public:

         //No copying
         DepthStencilBuffer(const DepthStencilBuffer&) = delete;
         DepthStencilBuffer& operator=(const DepthStencilBuffer&) = delete;

```

```
      DepthStencilBuffer();

      DepthStencilBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
          const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int index, unsigned int dsvSize,
          unsigned int width, unsigned int height, DXGI_FORMAT format = DXGI_FORMAT_D24_UNORM_S8_UINT, unsigned int sampleCount = 1);

      DXGI_FORMAT GetDepthStencilFormat() const;

      void CreateDepthStencilBufferAndView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
          const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& dsvHeap, unsigned int index, unsigned int dsvSize,
          unsigned int width, unsigned int height, DXGI_FORMAT format = DXGI_FORMAT_D24_UNORM_S8_UINT, unsigned int sampleCount = 1);

      void ReleaseBuffer();

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

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


  class StaticBuffer
  {
  public:

      //No copying
      StaticBuffer(const StaticBuffer&) = delete;
      StaticBuffer& operator=(const StaticBuffer&) = delete;

      StaticBuffer();

      StaticBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
          const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const void* data,
          unsigned int numBytes, unsigned int stride);

      StaticBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
          const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const void* data,
          unsigned int numBytes, DXGI_FORMAT format);

      StaticBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
          const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const wchar_t* filename);

      void CreateStaticBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
          const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const void* data,
          unsigned int numBytes, unsigned int stride);

      void CreateStaticBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
          const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const void* data,
          unsigned int numBytes, DXGI_FORMAT format);

      void CreateStaticBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
          const Microsoft::WRL::ComPtr<ID3D12GraphicsCommandList>& commandList, const wchar_t* filename);

      const D3D12_VERTEX_BUFFER_VIEW GetVertexBufferView() const;

      const D3D12_INDEX_BUFFER_VIEW GetIndexBufferView() const;

      void CreateTexture2DView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
          const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& srvHeap, unsigned int srvSize, unsigned int index);

      void CreateTexture2DMSView(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
          const Microsoft::WRL::ComPtr<ID3D12DescriptorHeap>& srvHeap, unsigned int srvSize, unsigned int index);

      void ReleaseBuffer();

  private:
      Microsoft::WRL::ComPtr<ID3D12Resource> mStaticDefaultBuffer;
      Microsoft::WRL::ComPtr<ID3D12Resource> mStaticUploadBuffer;

      union
      {
          unsigned int mStride;
          DXGI_FORMAT mFormat;
      };
  };
```

```
    class DynamicBuffer
    {
    public:

        //No copying
        DynamicBuffer(const DynamicBuffer&) = delete;
        DynamicBuffer& operator=(const DynamicBuffer&) = delete;

        DynamicBuffer();

        DynamicBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device, unsigned int numOfBytes,
      unsigned int stride);

        DynamicBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device, unsigned int numOfBytes,
      DXGI_FORMAT format);

        ~DynamicBuffer();

        void CreateDynamicBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device, unsigned int
      numOfBytes, unsigned int stride);

        void CreateDynamicBuffer(const Microsoft::WRL::ComPtr<ID3D12Device>& device, unsigned int
      numOfBytes, DXGI_FORMAT format);

        const D3D12_GPU_VIRTUAL_ADDRESS GetGPUAddress(unsigned int index) const;

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

        const D3D12_VERTEX_BUFFER_VIEW GetVertexBufferView();

        const D3D12_INDEX_BUFFER_VIEW GetIndexBufferView();

        void CopyData(unsigned int index, const void* data, unsigned long long numOfBytes);

        void ReleaseBuffer();

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

        union
        {
            UINT mStride;
            DXGI_FORMAT mFormat;
        };
    };
}
```

6.2 Camera.h

```
#pragma once


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


namespace RenderingEngine
{
    struct Camera
    {
        //camera position in world coordinates
        vec3 position;

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

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

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

        //stores the world to camera transform
        mat4 viewMatrix;

        //the velocities of the camera.
        float linearSpeed = 0.0f;
        float angularSpeed = 0.0f;
    };

    void SetProperties(Camera& camera, vec3 position, vec3 x, vec3 y, vec3 z, float linearSpeed, float
   angularSpeed);

    void LookAt(Camera& camera, vec3 position, vec3 target, vec3 up);

    void UpdateViewMatrix(Camera& camera);

    void Left(Camera& camera, float dt);

    void Right(Camera& camera, float dt);

    void Foward(Camera& camera, float dt);

    void Backward(Camera& camera, float dt);

    void Up(Camera& camera, float dt);

    void Down(Camera& camera, float dt);

    void RotateCameraLeftRight(Camera& camera, float xDiff);

    void RotateCameraUpDown(Camera& camera, float yDiff);
}
```

6.3 Color.h

```
#pragma once

#include "MathEngine.h"

namespace RenderingEngine
{
    class Color
    {
    public:

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

        Color(const vec4& color);

        const vec4& GetColor() const;

        float GetRed() const;

        float GetGreen() const;

        float GetBlue() const;

        float GetAlpha() const;

        void SetColor(const vec4& color);

        void SetRed(float r);

        void SetGreen(float g);

        void SetBlue(float b);

        void SetAlpha(float a);

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

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

        Color& operator*=(float k);

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

    private:
        vec4 mColor;
    };

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

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

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

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

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

6.4 DDSTextureLoader.h

```
//--------------------------------------------------------------------------------------
// File:  DDSTextureLoader.h
//
// Functions for loading a DDS texture and creating a Direct3D 11 runtime resource for it
//
// Note these functions are useful as a light-weight runtime loader for DDS files.  For
// a full-featured DDS file reader, writer, and texture processing pipeline see
// the 'Texconv' sample and the 'DirectXTex' library.
//
// THIS CODE AND INFORMATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF
// ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO
// THE IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A
// PARTICULAR PURPOSE.
//
// Copyright (c) Microsoft Corporation.  All rights reserved.
//
// http://go.microsoft.com/fwlink/?LinkId=248926
// http://go.microsoft.com/fwlink/?LinkId=248929
//--------------------------------------------------------------------------------------

#ifdef _MSC_VER
#pragma once
#endif

#include <wrl.h>
#include <d3d11_1.h>
#include "d3dx12.h"

#pragma warning(push)
#pragma warning(disable :  4005)
#include <stdint.h>

#pragma warning(pop)

#if defined(_MSC_VER) && (_MSC_VER<1610) && !defined(_In_reads_)
#define _In_reads_(exp)
#define _Out_writes_(exp)
#define _In_reads_bytes_(exp)
#define _In_reads_opt_(exp)
#define _Outptr_opt_
#endif

#ifndef _Use_decl_annotations_
#define _Use_decl_annotations_
#endif

namespace DirectX
{
    enum DDS_ALPHA_MODE
    {
        DDS_ALPHA_MODE_UNKNOWN       = 0,
        DDS_ALPHA_MODE_STRAIGHT      = 1,
        DDS_ALPHA_MODE_PREMULTIPLIED = 2,
        DDS_ALPHA_MODE_OPAQUE        = 3,
        DDS_ALPHA_MODE_CUSTOM        = 4,
    };

    HRESULT CreateDDSTextureFromMemory12(_In_ ID3D12Device* device,
                                         _In_ ID3D12GraphicsCommandList* cmdList,
                                         _In_reads_bytes_(ddsDataSize) const uint8_t* ddsData,
                                         _In_ size_t ddsDataSize,
                                         _Out_ Microsoft::WRL::ComPtr<ID3D12Resource>& texture,
                                         _Out_ Microsoft::WRL::ComPtr<ID3D12Resource>& textureUploadHeap,
                                         _In_ size_t maxsize = 0,
                                         _Out_opt_ DDS_ALPHA_MODE* alphaMode = nullptr
                                         );

    HRESULT CreateDDSTextureFromFile12(_In_ ID3D12Device* device,
        _In_ ID3D12GraphicsCommandList* cmdList,
        _In_z_ const wchar_t* szFileName,
        _Out_ Microsoft::WRL::ComPtr<ID3D12Resource>& texture,
        _Out_ Microsoft::WRL::ComPtr<ID3D12Resource>& textureUploadHeap,
        _In_ size_t maxsize = 0,
        _Out_opt_ DDS_ALPHA_MODE* alphaMode = nullptr
    );
}
```

6.5 DeviceResources.h

```
#pragma once

```

```
#include <wrl.h>
#include <d3dx12.h>
#include <dxgi1_4.h>
#include "SwapChain.h"
#include "Multisampling.h"
#include "TextResources.h"

namespace RenderingEngine
{
    class DeviceResources
    {
    public:

        //No copying
        DeviceResources(const DeviceResources&) = delete;
        DeviceResources& operator=(const DeviceResources&) = delete;

        static const unsigned int NUM_OF_FRAMES{ 3 };

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

        ~DeviceResources();

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

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

        DXGI_FORMAT GetBackBufferFormat() const;

        DXGI_FORMAT GetDepthStencilFormat() const;

        unsigned int GetCBVSRVUAVSize() const;

        unsigned int GetCurrentFrame() const;

        const TextResources& GetTextResources() const;

        void UpdateCurrentFrameFenceValue();

        void FlushCommandQueue();

        void WaitForGPU() const;

        void Signal();

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

        void RTBufferTransition(bool isMSAAEnabled, bool isTextEnabled);

        void BeforeTextDraw();

        void AfterTextDraw();

        void Execute() const;

        void Present();

        /*@brief Calls the necessary functions to let the user draw their objects.
*
*  @param[in] isMSAAEnabled Pass in true if MSAA enabled, false otherwise.
*/
        void Draw(bool isMSAAEnabled);

        void NextFrame();

    private:

        DeviceResources(unsigned int width, unsigned int height, HWND windowHandle,
            bool isMSAAEnabled, bool isTextEnabled);

        unsigned int mCurrentFrameIndex;

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

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

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

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

        UINT mRTVSize;
        UINT mDSVSize;
        UINT mCBVSize;

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

        SwapChain mSwapChain;

        MultiSampling mMultiSampling;

        D3D12_VIEWPORT mViewport{};
        D3D12_RECT mScissor{};

        TextResources mTextResources;

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

6.6 Direct3DLink.h

```
#pragma once

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

6.7 DirectXException.h

```
#pragma once

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

inline std::wstring AnsiToWString(const std::string& str)
{
    WCHAR buffer[2048];
    MultiByteToWideChar(CP_ACP, 0, str.c_str(), -1, buffer, 1024);
    return std::wstring(buffer);
}

class DirectXException
{
public:

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

    std::wstring ErrorMsg() const;

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

#ifndef ThrowIfFailed
#define ThrowIfFailed(x)                                                  \
{                                                                         \
HRESULT hr = (x);                                                       \
std::wstring filename(AnsiToWString(__FILE__));                            \
if (FAILED(hr)) { throw DirectXException(hr, L#x, filename, __LINE__); }    \
}
#endif


inline void CreateInfoQueue(Microsoft::WRL::ComPtr<IDXGIInfoQueue>& infoQueue)
{
#if defined(_DEBUG) || defined(DEBUG)
    //define function signature
    typedef HRESULT(WINAPI* dxgiDebugInterface)(REFIID, void**);

    //Get a handle to the dll file
    HMODULE dxgiDebugHandle;
    GetModuleHandleEx(GET_MODULE_HANDLE_EX_FLAG_UNCHANGED_REFCOUNT, L"Dxgidebug.dll", &dxgiDebugHandle);

    //get the address of the function DXGIGetDebugInterface in the dll file
    dxgiDebugInterface DXGIGetDebugInterface = (dxgiDebugInterface)GetProcAddress(dxgiDebugHandle, "DXGIGetDebugInterface");
    if (DXGIGetDebugInterface == nullptr)
    {
        exit(-1);
    }

    //create a DXGIInfoQueue object.
    DXGIGetDebugInterface(IID_PPV_ARGS(&infoQueue));
#endif
}



inline std::wstring ErrorMessage(HRESULT errorCode, const std::wstring& functionName, const std::wstring& filename, int lineNumber,
    const Microsoft::WRL::ComPtr<IDXGIInfoQueue>& infoQueue)
{
    //the _com_error class lets us retrieve the error message associated with the HRESULT error code
    _com_error error(errorCode);
    std::wstring msg = error.ErrorMessage();

    //Get the hex value of the error code
    std::stringstream ss;
    ss « std::hex « errorCode;
    std::wstring hrHex{ AnsiToWString(ss.str()) };

    std::wstring eCode(std::to_wstring(errorCode));


    std::wstring errorMessage{ L"File Name:  " + filename + L"\n\n" + L"Function Name:  " + functionName + L"\n\n" +
        L"Line Number:  " + std::to_wstring(lineNumber) + L"\n\n" + L"Error Code:  " + eCode +
        L"(0x" + hrHex + L")" + L"\n\n" + L"Error Code Description:  " + msg };

    std::vector<std::wstring> messages;

    if (infoQueue != nullptr)
    {
        //Get the number of messages in the queue.
        UINT64 numOfMessages = infoQueue->GetNumStoredMessages(DXGI_DEBUG_ALL);

        for (UINT64 i = 0; i < numOfMessages; ++i)
        {
            //Get the length of the current message.
            SIZE_T messageLength{ 0 };
            infoQueue->GetMessage(DXGI_DEBUG_ALL, i, nullptr, &messageLength);

            //Allocate enough memory to store the message.
            std::unique_ptr<unsigned char[]> bytes = std::make_unique<unsigned char[]>(messageLength);
            DXGI_INFO_QUEUE_MESSAGE* pMsg = (DXGI_INFO_QUEUE_MESSAGE*)bytes.get();

            //Retrieve the message.  It will be stored in pMsg.
            infoQueue->GetMessage(DXGI_DEBUG_ALL, i, pMsg, &messageLength);

            //Store the message.
            std::string tempMessage{ pMsg->pDescription };
            messages.emplace_back(AnsiToWString(tempMessage));
        }
    }

    for (int i = 0; i < messages.size(); ++i)
    {
        errorMessage += L"\n";
        errorMessage += messages[i];
    }

    return errorMessage;
}
```

```
#ifndef ExitIfFailed
#define ExitIfFailed(x)                                                                          \
{                                                                                                \
HRESULT hr = (x);                                                                            \
if (FAILED(hr))                                                                              \
{                                                                                            \
Microsoft::WRL::ComPtr<IDXGIInfoQueue> infoQueue;                                        \
CreateInfoQueue(infoQueue);                                                              \
std::wstring filename(AnsiToWString(__FILE__));                                          \
std::wstring errMsg = ErrorMessage(hr, L#x, filename, __LINE__, infoQueue);              \
MessageBox(nullptr, errMsg.c_str(), L"DirectX Error", MB_OK);                            \
        exit(-1);                                                                                \
    }                                                                                            \
}
#endif
```

6.8 DrawArguments.h

```
#pragma once

#include <string>
#include <d3dcommon.h>

namespace RenderingEngine
{
    struct DrawArguments
    {
        unsigned int indexCount = 0;
        unsigned int locationOfFirstIndex = 0;
        int indexOfFirstVertex = 0;
        unsigned int indexOfConstantData = 0;
        unsigned int rootParameterIndex = 0;
        std::wstring constantBufferKey = L"";
        D3D_PRIMITIVE_TOPOLOGY primtive = D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST;
    };
}
```

6.9 GameTime.h

```
#pragma once

#include <Windows.h>

namespace RenderingEngine
{
    struct Time
    {
        __int64 previousTime = 0;
        __int64 currentTime = 0;
        double deltaTime = 0;
        double secondsPerCount = 0.0;
        bool stopped = false;
    };

    void InitializeTime(Time& time);

    void Reset(Time& time);

    void Tick(Time& time);

    void Start(Time& time);

    void Stop(Time& time);
}
```

6.10 Multisampling.h

```
#pragma once

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

namespace RenderingEngine
{
    class MultiSampling
    {
    public:

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

        MultiSampling();

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

        void CheckMultiSamplingSupport(const Microsoft::WRL::ComPtr<ID3D12Device>& device,
            DXGI_FORMAT rtFormat, unsigned int sampleCount);

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

        DXGI_FORMAT GetRenderTargetFormat();

        DXGI_FORMAT GetDepthStencilFormat();

        void ReleaseBuffers();

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

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

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

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

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

    private:
        RenderTargetBuffer mMSAARenderTargetBuffer;
        DepthStencilBuffer mMSAADepthStencilBuffer;
        unsigned int mSampleCount;
    };
}
```

6.11 OrthographicProjection.h

```
#pragma once

#include "MathEngine.h"

namespace RenderingEngine
{
    struct OrthogrpahicProjection
    {
        float width = 0.0f;
        float height = 0.0f;
        float znear = 0.0f;
        float zfar = 0.0f;
        mat4 projectionMatrix;
    };

    void SetProperties(OrthogrpahicProjection& ortho, float width, float height, float znear, float zfar);

    void UpdateProjectionMatrix(OrthogrpahicProjection& ortho);
}
```

6.12 PerspectiveProjection.h

```
#pragma once
```

```

#include "MathEngine.h"


namespace RenderingEngine
{
    struct PerspectiveProjection
    {
        float znear = 0.0f;
        float zfar = 0.0f;
        float verticalFov = 0.0f;
        float aspectRatio = 0.0f;
        mat4 projectionMatrix;
    };

    void SetProperties(PerspectiveProjection& perspective, float znear, float zfar, float vFov, float
    aspectRatio);

    void UpdateProjectionMatrix(PerspectiveProjection& perspective);
}
```

6.13 RenderScene.h

```
#pragma once


#include <d3dcompiler.h>
#include <unordered_map>
#include "DeviceResources.h"
#include "Buffer.h"
#include "Color.h"
#include <string_view>

namespace RenderingEngine
{
    class RenderScene
    {
    public:


    //-----------------------------------------------------------------------------------------------------------------------
        //CONSTRUCTORS

        //No copying
        RenderScene(const RenderScene&) = delete;
        RenderScene operator=(const RenderScene&) = delete;

        /*@brief Creates a RenderScene object.  Does not create the necessary resources to render a
    scene.
* Call the function CreateDeviceResources() to initialize all necessary resources.
*/
        RenderScene();

        /*@brief Initializes all necessary resources.
*
* @param[in] width The width of a window.
* @param[in] height The height of a window.
* @param[in] windowHandle A handle to a window.
* @param[in, optional] isMSAAEnabled Pass in true if you want to have MSAA enabled, false otherwise.
* @param[in, optional] isTextEnabled Pass in true if you want to have text enabled, false otherwise.
*/
        RenderScene(unsigned int width, unsigned int height, HWND windowHandle,
            bool isMSAAEnabled = false, bool isTextEnabled = false);

        /*@brief Initializes all necessary resources.
*
* @param[in] width The width of a window.
* @param[in] height The height of a window.
* @param[in] windowHandle A handle to a window.
* @param[in, optional] isMSAAEnabled Pass in true if you want to have MSAA enabled, false otherwise.
* @param[in, optional] isTextEnabled Pass in true if you want to have text enabled, false otherwise.
*/
        void CreateDeviceResources(unsigned int width, unsigned int height, HWND windowHandle,
            bool isMSAAEnabled = false, bool isTextEnabled = false);


    //-----------------------------------------------------------------------------------------------------------------------







    //-----------------------------------------------------------------------------------------------------------------------
```

```
     //SHADER FUNCTIONS

     void LoadShader(unsigned int shaderKey, std::wstring_view filename);

     void CompileShader(unsigned int shaderKey, std::wstring_view filename,
         std::string_view entryPointName, std::string_view target);

     void RemoveShader(unsigned int shaderKey);



     void LoadShader(std::wstring_view shaderKey, std::wstring_view filename);

      void CompileShader(std::wstring_view shaderKey, std::wstring_view filename,
          std::string_view entryPointName, std::string_view target);

      void RemoveShader(std::wstring_view shaderKey);

//-----------------------------------------------------------------------------------------------------------------------






//-----------------------------------------------------------------------------------------------------------------------
      //INPUT ELEMENT DESCRIPTION FUNCTIONS

      void CreateInputElementDescription(unsigned int key, const char* semanticName, unsigned int semanticIndex,
          DXGI_FORMAT format, unsigned int inputSlot, unsigned int byteOffset,
          D3D12_INPUT_CLASSIFICATION inputSlotClass = D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA,
          unsigned int instanceStepRate = 0);



      void CreateInputElementDescription(std::wstring_view key, const char* semanticName, unsigned int semanticIndex,
          DXGI_FORMAT format, unsigned int inputSlot, unsigned int byteOffset,
          D3D12_INPUT_CLASSIFICATION inputSlotClass = D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA,
          unsigned int instanceStepRate = 0);


//-----------------------------------------------------------------------------------------------------------------------






//-----------------------------------------------------------------------------------------------------------------------
      //ROOT PARAMETER FUNCTIONS

      void CreateRootDescriptor(unsigned int rootParameterKey, unsigned int shaderRegister);

      void CreateDescriptorRange(unsigned int descriptorRangeKey,
          D3D12_DESCRIPTOR_RANGE_TYPE type, unsigned int numDescriptors, unsigned int shaderRegister, unsigned int registerSpace,
          unsigned int offset);

      void CreateDescriptorTable(unsigned int rootParameterKey, unsigned int descriptorRangeKey);

      void CreateRootConstants(unsigned int rootParameterKey, unsigned int shaderRegister, unsigned int numValues);



      void CreateRootDescriptor(std::wstring_view rootParameterKey, unsigned int shaderRegister);

      void CreateDescriptorRange(std::wstring_view descriptorRangeKey,
          D3D12_DESCRIPTOR_RANGE_TYPE type, unsigned int numDescriptors, unsigned int shaderRegister, unsigned int registerSpace,
          unsigned int offset);

      void CreateDescriptorTable(std::wstring_view rootParameterKey, unsigned int descriptorRangeKey);

      void CreateRootConstants(std::wstring_view rootParameterKey, unsigned int shaderRegister, unsigned int numValues);


//-----------------------------------------------------------------------------------------------------------------------




```

```

//------------------------------------------------------------------------------------------------------------------
      //ROOT SIGNATURE FUNCTIONS

      void CreateRootSignature(unsigned int rootSigKey, unsigned int rootParametersKey);

      void CreateRootSignature(unsigned int rootSigKey, unsigned int rootParametersKey,
          unsigned int staticsSamplerKey);

      void CreateStaticSampler(unsigned int staticSamplerKey, D3D12_FILTER filter,
          D3D12_TEXTURE_ADDRESS_MODE u, D3D12_TEXTURE_ADDRESS_MODE v, D3D12_TEXTURE_ADDRESS_MODE w,
unsigned int shaderRegister);



      void CreateRootSignature(std::wstring_view rootSigKey, std::wstring_view rootParametersKey);

      void CreateRootSignature(std::wstring_view rootSigKey, std::wstring_view rootParametersKey,
          std::wstring_view staticsSamplerKey);

      void CreateStaticSampler(std::wstring_view staticSamplerKey, D3D12_FILTER filter,
          D3D12_TEXTURE_ADDRESS_MODE u, D3D12_TEXTURE_ADDRESS_MODE v, D3D12_TEXTURE_ADDRESS_MODE w,
unsigned int shaderRegister);


//------------------------------------------------------------------------------------------------------------------






//------------------------------------------------------------------------------------------------------------------
      //PIPELINE STATE OBJECT FUNCTIONS

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

      void LinkPSOAndRootSignature(unsigned int psoKey, unsigned int rootSigKey);



      void CreatePSO(std::wstring_view psoKey, D3D12_FILL_MODE fillMode, BOOL enableMultisample,
          std::wstring_view vsKey, std::wstring_view psKey, std::wstring_view
inputElementDescriptionsKey,
          std::wstring_view rootSigKey,
          const D3D12_PRIMITIVE_TOPOLOGY_TYPE& primitiveType, UINT sampleCount = 1);

      void LinkPSOAndRootSignature(std::wstring_view psoKey, std::wstring_view rootSigKey);


//------------------------------------------------------------------------------------------------------------------






//------------------------------------------------------------------------------------------------------------------
      //STATIC BUFFER FUNCTIONS

      void CreateStaticBuffer(unsigned int staticBufferKey, const void* data, unsigned numBytes,
unsigned int stride);

      void CreateStaticBuffer(unsigned int staticBufferKey, const void* data, unsigned numBytes,
DXGI_FORMAT format);

      void CreateStaticBuffer(unsigned int staticBufferKey, const wchar_t* filename, unsigned int
texType, unsigned int index);

      void LinkStaticBuffer(unsigned int bufferType, unsigned int staticBufferKey);



      void CreateStaticBuffer(std::wstring_view staticBufferKey, const void* data, unsigned numBytes,
unsigned int stride);

      void CreateStaticBuffer(std::wstring_view staticBufferKey, const void* data, unsigned numBytes,
DXGI_FORMAT format);

      void CreateStaticBuffer(std::wstring_view staticBufferKey, const wchar_t* filename, unsigned int
texType, unsigned int index);

      void LinkStaticBuffer(unsigned int bufferType, std::wstring_view staticBufferKey);

```

```
//------------------------------------------------------------------------------------------------------------------------







//------------------------------------------------------------------------------------------------------------------------
      //DYNAMIC BUFFER FUNCTIONS

      void CreateDynamicBuffer(unsigned int dynamicBufferKey, unsigned numBytes, const void* data, unsigned int stride);

      void CreateDynamicBuffer(unsigned int dynamicBufferKey, unsigned numBytes, const void* data, DXGI_FORMAT format);

      void LinkDynamicBuffer(unsigned int bufferType, unsigned int dynamicBufferKey, unsigned int indexConstantData = 0,
          unsigned int rootParameterIndex = 0);

      void CopyDataIntoDynamicBuffer(unsigned int dynamicBufferKey, unsigned int index, const void* data, UINT64 numOfBytes);



      void CreateDynamicBuffer(std::wstring_view dynamicBufferKey, unsigned numBytes, const void* data, unsigned int stride);

      void CreateDynamicBuffer(std::wstring_view dynamicBufferKey, unsigned numBytes, const void* data, DXGI_FORMAT format);

      void LinkDynamicBuffer(unsigned int bufferType, std::wstring_view dynamicBufferKey, unsigned int indexConstantData = 0,
          unsigned int rootParameterIndex = 0);

      void CopyDataIntoDynamicBuffer(std::wstring_view dynamicBufferKey, unsigned int index, const void* data, UINT64 numOfBytes);


//------------------------------------------------------------------------------------------------------------------------







//------------------------------------------------------------------------------------------------------------------------
      //TEXTURE FUNCTIONS

      void CreateTextureViewHeap(unsigned int numDescriptors);

      void LinkTextureViewHeap();

      void LinkTexture(unsigned int rootParameterIndex);

      void LinkTexture(unsigned int rootParameterIndex, unsigned int textureViewIndex);


//------------------------------------------------------------------------------------------------------------------------


//------------------------------------------------------------------------------------------------------------------------
      //RENDER OBJECTS FUNCTONS

      void BeforeRenderObjects(bool isMSAAEnabled = false);

      void RenderObject(unsigned int indexCount, unsigned int locationFirstIndex, int indexOfFirstVertex,
          D3D_PRIMITIVE_TOPOLOGY primitive);

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


//------------------------------------------------------------------------------------------------------------------------







//------------------------------------------------------------------------------------------------------------------------
      //RENDER TEXT FUNCTIONS

      void BeforeRenderText();

      void RenderText(const vec4& textLocation, const Color& textColor, float textSize,
            const std::wstring& textString, DWRITE_PARAGRAPH_ALIGNMENT alignment =
DWRITE_PARAGRAPH_ALIGNMENT_CENTER);

        void AfterRenderText();


//-----------------------------------------------------------------------------------------------------------------------






//-----------------------------------------------------------------------------------------------------------------------
        //MISCELLANEOUS FUNCTIONs

        void AfterRender();

        void ExecuteAndFlush();

        void Resize(unsigned int width, unsigned int height, HWND windowHandle, bool isMSAAEnabled =
false, bool isTextEnabled = false);

        void SetConstants(unsigned int rootParameterIndex, unsigned int numValues, void* data, unsigned
int index);


//-----------------------------------------------------------------------------------------------------------------------

   private:

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



//-----------------------------------------------------------------------------------------------------------------------
        //SHADER HASH MAPS

        //Stores all of the shaders for this scene.
        std::unordered_map<unsigned int, Microsoft::WRL::ComPtr<ID3DBlob» mShaders;

        //Stores all of the shaders for this scene.
        std::unordered_map<std::wstring_view, Microsoft::WRL::ComPtr<ID3DBlob» mShadersStr;


//-----------------------------------------------------------------------------------------------------------------------






//-----------------------------------------------------------------------------------------------------------------------
        //INPUT ELEMENT DESCRIPTION HASH MAPS

        //Stores input element descriptions for a set of shaders.
        std::unordered_map<unsigned int, std::vector<D3D12_INPUT_ELEMENT_DESC»
mInputElementDescriptions;

        //Stores input element descriptions for a set of shaders.
        std::unordered_map<std::wstring_view, std::vector<D3D12_INPUT_ELEMENT_DESC»
mInputElementDescriptionsStr;


//-----------------------------------------------------------------------------------------------------------------------






//-----------------------------------------------------------------------------------------------------------------------
        //ROOT PARAMETER HASH MAPS

        //Stores root parameters for root signatures.
        std::unordered_map<unsigned int, std::vector<D3D12_ROOT_PARAMETER» mRootParameters;

        //Stores descriptor ranges for descriptor tables.
        std::unordered_map<unsigned int, std::vector<D3D12_DESCRIPTOR_RANGE» mDescriptorRanges;

        //Stores root parameters for root signatures.
        std::unordered_map<std::wstring_view, std::vector<D3D12_ROOT_PARAMETER» mRootParametersStr;

        //Stores descriptor ranges for descriptor tables.
        std::unordered_map<std::wstring_view, std::vector<D3D12_DESCRIPTOR_RANGE» mDescriptorRangesStr;

```

```
//----------------------------------------------------------------------------------------------------------------------





//----------------------------------------------------------------------------------------------------------------------
    //ROOT SIGNATURE HASH MAPS

    //The root signatures for the scene.
    //Describes all of the constant data that is expected in a set of shaders.
    //Microsoft::WRL::ComPtr<ID3D12RootSignature> mRootSignature;
    std::unordered_map<unsigned int, Microsoft::WRL::ComPtr<ID3D12RootSignature» mRootSignatures;

    //Stores static samplers.
    std::unordered_map<unsigned int, std::vector<D3D12_STATIC_SAMPLER_DESC» mStaticSamplers;

    //The root signatures for the scene.
    //Describes all of the constant data that is expected in a set of shaders.
    //Microsoft::WRL::ComPtr<ID3D12RootSignature> mRootSignature;
    std::unordered_map<std::wstring_view, Microsoft::WRL::ComPtr<ID3D12RootSignature» mRootSignaturesStr;

    //Stores static samplers.
    std::unordered_map<std::wstring_view, std::vector<D3D12_STATIC_SAMPLER_DESC» mStaticSamplersStr;


//----------------------------------------------------------------------------------------------------------------------





//----------------------------------------------------------------------------------------------------------------------
    //PIPELINE STATE OBJECT HASH MAPS

    //Stores pipeline state objects.
    std::unordered_map<unsigned int, Microsoft::WRL::ComPtr<ID3D12PipelineState» mPSOs;

    //Stores pipeline state objects.
    std::unordered_map<std::wstring_view, Microsoft::WRL::ComPtr<ID3D12PipelineState» mPSOsStr;

//----------------------------------------------------------------------------------------------------------------------





//----------------------------------------------------------------------------------------------------------------------
    //STATIC BUFFER HASH MAPS

    //Stores data that will not be updated on a per-frame basis.
    std::unordered_map<unsigned int, StaticBuffer> mStaticBuffers;

    //Stores data that will not be updated on a per-frame basis.
    std::unordered_map < std::wstring_view, StaticBuffer> mStaticBuffersStr;


//----------------------------------------------------------------------------------------------------------------------





//----------------------------------------------------------------------------------------------------------------------
    //DYNAMIC BUFFER HASH MAPS

    //Stores data that will be updated on a per-frame basis.
    //We can't update a dynamic buffer until the GPU
    //is done executing all the commands that reference it, so each frame needs its own dynamic buffer.
    std::unordered_map<unsigned int, DynamicBuffer[DeviceResources::NUM_OF_FRAMES]> mDynamicBuffers;

    //Stores data that will be updated on a per-frame basis.
    //We can't update a dynamic buffer until the GPU
    //is done executing all the commands that reference it, so each frame needs its own dynamic buffer.
    std::unordered_map<std::wstring_view, DynamicBuffer[DeviceResources::NUM_OF_FRAMES]> mDynamicBuffersStr;


//----------------------------------------------------------------------------------------------------------------------
```

```


        //Used to store descriptors of textures.
        Microsoft::WRL::ComPtr<ID3D12DescriptorHeap> mTextureViewHeap;

    };
}
```

6.14 SwapChain.h

```
#pragma once

#include <wrl.h>
#include "d3dx12.h"
#include <dxgi1_4.h>
#include <vector>
#include <memory>
#include "Buffer.h"

namespace RenderingEngine
{
    class SwapChain
    {
    public:

        //No copying
        SwapChain(const SwapChain&) = delete;
        SwapChain& operator=(const SwapChain&) = delete;

        SwapChain();

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

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

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

        unsigned int GetNumRenderTargetBuffers() const;

        unsigned int GetCurrentBackBufferIndex() const;

        DXGI_FORMAT GetBackBufferFormat() const;

        DXGI_FORMAT GetDepthStencilFormat() const;

        const std::vector<std::unique_ptr<RenderTargetBuffer>>& GetRenderTargetBuffers();

        void ReleaseBuffers();

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

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

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

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

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

        void Present();

```

```cpp
    private:
        unsigned int mNumRenderTargetBuffers;
        unsigned int mCurrentBackBufferIndex;

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

        DepthStencilBuffer mDepthStencilBuffer;
    };
}
```

6.15 Text.h

```cpp
#pragma once


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

namespace RenderingEngine
{
    class Text
    {
    public:

        Text() = default;

        Text(const vec4& textLocation, const std::wstring& textString, float textSize, const Color& textColor);

        const vec4& GetTextLocation() const;

        const std::wstring& GetTextString() const;

        float GetTextSize() const;

        const Color& GetTextColor() const;

        void SetTextSize(float textSize);

        void SetTextColor(const Color& textColor);

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

        void SetTextLocation(const vec4& textLocation);

    private:

        vec4 mTextLocation;
        std::wstring mText;
        float mTextSize{ 0.0f };
        Color mTextColor;
    };
}
```

6.16 TextResources.h

```cpp
#pragma once

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

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

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

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

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

        void ResetBuffers();

        void ResizeBuffers(const std::vector<std::unique_ptr<RenderTargetBuffer»& renderTargetBuffers, HWND windowHandle);

        void BeforeRenderText(unsigned int currentBackBuffer);

        void AfterRenderText(unsigned int currentBackBuffer);

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

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

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

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

6.17 Window.h

```
#pragma once

#include <Windows.h>
#include <string>
#include "Color.h"

namespace RenderingEngine
{
    struct Window
    {
        HWND windowHandle;
        WNDCLASSEX windowClass;
    };

    void CreateParentWindow(Window& window, const HINSTANCE& hInstance, WNDPROC windowProcedure, const RenderingEngine::Color& backgroundColor,
        const std::wstring& windowClassName, const std::wstring& windowName, unsigned int styles,
        unsigned int x, unsigned int y, unsigned int width, unsigned int height, void* additionalData = nullptr);

    void CreateChildWindow(Window& window, const HINSTANCE& hInstance, HWND parent, unsigned long long int identifier,
        WNDPROC windowProcedure, const RenderingEngine::Color& backgroundColor,
        const std::wstring& windowClassName, const std::wstring& windowName, unsigned int styles,
        unsigned int x, unsigned int y, unsigned int width, unsigned int height, void* additionalData = nullptr);

    void CreateControlWindow(Window& window, const HINSTANCE& hInstance, HWND parent, unsigned long long int identifier,
        const std::wstring& windowClassName, const std::wstring& windowName, unsigned int styles,
        unsigned int x, unsigned int y, unsigned int width, unsigned int height, void* additionalData = nullptr);

    unsigned int GetWidth(const Window& window);

    unsigned int GetHeight(const Window& window);

    unsigned int GetX(const Window& window);

    unsigned int GetY(const Window& window);
}
```

Index