
Vortex2D Documentation

Release 1.0

Maximilian Maldacker

Nov 22, 2018

Contents

1	2D real-time fluid engine	1
1.1	Introduction	1
1.2	Table of content	1

1.1 Introduction

This is a 2D engine with the goal of being used in real-time scenarios, e.g. video games, in the same manner as a Box2D.

It is a hybrid engine that uses particles for the advection part (moving the particles given their velocities) and a grid to resolve the incompressible constraints. However, the particles are not visible to the user and you only work with the grid directly.

The engine runs directly on the GPU using Vulkan compute shaders. The rendering is then also done with Vulkan. The grids are represented by textures and operations by the user are all done by drawing shapes on the textures. The engine itself is written in C++ and it provides a simple wrapper around Vulkan and the basic rendering functionalities: shapes, textures, sprites, blending, render textures, etc.

1.2 Table of content

1.2.1 Rendering

Initialization

The rendering API is very basic and supports only the most basic functionality.

Create an instance of `Vortex2D::Renderer::Instance` which is then used to create an instance of `Vortex2D::Renderer::Device`.

The device is then used to create any other object. The main one is the `Vortex2D::Renderer::RenderWindow` which is a window where to render sprites and polygons. The function `Vortex2D::Fluid::RenderWindow::Display()` is then used to present the result to the screen.

```
Vortex2D::Renderer::Instance instance("Application name", extensions); // pass list_
↳of required extensions
Vortex2D::Renderer::Device device(instance.GetPhysicalDevice(), surface);

Vortex2D::Renderer::RenderWindow window(device, surface, width, height);
```

Note that the instance requires a list of extensions necessary to create a window. With GLFW they can be retrieved as:

```
std::vector<const char*> GetGLFWExtensions()
{
    std::vector<const char*> extensions;
    unsigned int glfwExtensionCount = 0;
    const char** glfwExtensions;

    // get the required extensions from GLFW
    glfwExtensions = glfwGetRequiredInstanceExtensions(&glfwExtensionCount);
    for (unsigned int i = 0; i < glfwExtensionCount; i++)
    {
        extensions.push_back(glfwExtensions[i]);
    }

    return extensions;
}
```

In addition, you also need to create a surface which can be also done with the help of GLFW:

```
vk::UniqueSurfaceKHR GetGLFWSurface(GLFWwindow* window, vk::Instance instance)
{
    // create surface
    VkSurfaceKHR surface;
    if (glfwCreateWindowSurface(static_cast<VkInstance>(instance), window, nullptr, &
↳surface) != VK_SUCCESS)
    {
        throw std::runtime_error("failed to create window surface!");
    }

    return vk::UniqueSurfaceKHR(surface, vk::SurfaceKHRDeleter{instance});
}
```

Rendering shapes

We are now ready to draw things on the screen. Let's start with some shapes like rectangles and circles:

```
Vortex2D::Renderer::Rectangle rectangle(device, {100.0f, 100.0f});
Vortex2D::Renderer::Ellipse circle(device, {50.0f, 50.0f});

rectangle.Colour = {1.0f, 0.0f, 0.0f, 1.0f};
rectangle.Position = {200.0f, 100.0f};

circle.Colour = {0.0f, 0.0f, 1.0f, 1.0f};
circle.Position = {500.0f, 400.0f};

auto blendMode = vk::PipelineColorBlendAttachmentState()
    .setBlendEnable(true)
    .setAlphaBlendOp(vk::BlendOp::eAdd)
    .setColorBlendOp(vk::BlendOp::eAdd)
```

(continues on next page)

(continued from previous page)

```

        .setSrcColorBlendFactor(vk::BlendFactor::eSrcAlpha)
        .setSrcAlphaBlendFactor(vk::BlendFactor::eOne)
        .setDstColorBlendFactor(vk::BlendFactor::eOneMinusSrcAlpha)
        .setDstAlphaBlendFactor(vk::BlendFactor::eZero);

// note that rectangle, circle and render need to be alive for the duration of the_
↳rendering
auto render = renderTarget.Record({rectangle, circle}, blendMode);
render.Submit();

```

Rendering textures

Of course we can also render textures, using sprites.

```

Vortex2D::Renderer::Texture texture(device, 100, 100, vk::Format::eR8G8B8A8Unorm);
Vortex2D::Renderer::Sprite sprite(device, texture);

```

1.2.2 Level sets

A level set is a signed distance field. It's a field containing positive or negative value, where the values are 0 represent a contour, or border. This is used to represent shapes, the numbers give you the distance to the shape border. It's the fundamental way that we represent the area of a fluid and the area of the obstacles, i.e. the boundaries.

Signed distance field

Basic shapes

There are several shapes that will set a level set, we draw them like we draw any shape. The format of the `Vortex2D::Renderer::RenderTarget()` must be `vk::Format::eR32Sfloat`. The class `Vortex2D::Fluid::LevelSet()` fulfills this.

```

Vortex2D::Fluid::Rectangle rectangle(device, {100.0f, 100.0f});
rectangle.Position = {40.0f, 60.0f};

Vortex2D::Fluid::LevelSet levelSet(device, {400, 400});
auto renderCmd = levelSet.Record({rectangle});
renderCmd.Submit(); // note that renderCmd and rectangle have to be alive untill the_
↳rendering is done

```

Re-initialization

We can also draw any other shape, and then transform the data into a correct level set. This operation is call re-initialization.

1.2.3 World

The world classes are the centre of the engine, where the fluid gets animated.

Velocity Field

The velocity field is field where each entry gives the current speed of the fluid and its direction. This field is used to evolve the fluid, which means that additional velocity can be added (or subtracted) to it.

Smoke World

This is a type of fluid simulation where the fluid area doesn't move.

Water World

This is a classical water type of fluid simulation.

1.2.4 Rigid body

Rigid bodies are the way to have dynamic interactions with the fluid (other then changing the velocity field directly)

Static bodies

Static bodies act on the fluid, but the fluid doesn't act on the fluid. This is called a weak coupling between the rigid body and the fluid. They have a velocity that is imparted on the fluid. Think of motorized objects pushing through the fluid.

Dynamic bodies

Dynamic bodies are affected by the fluid. They can also in turn, affect the fluid, which is called a strong coupling with the fluid.

1.2.5 Renderer API reference

Classes

- *Vortex2D::Renderer::Clear*
- *Vortex2D::Renderer::Drawable*
- *Vortex2D::Renderer::Ellipse*
- *Vortex2D::Renderer::GenericBuffer*
- *Vortex2D::Renderer::IndirectBuffer*
- *Vortex2D::Renderer::Instance*
- *Vortex2D::Renderer::IntRectangle*
- *Vortex2D::Renderer::Rectangle*
- *Vortex2D::Renderer::RenderState*
- *Vortex2D::Renderer::RenderTarget*
- *Vortex2D::Renderer::RenderTexture*
- *Vortex2D::Renderer::RenderWindow*

- `Vortex2D::Renderer::Sprite`
- `Vortex2D::Renderer::Timer`
- `Vortex2D::Renderer::Transformable`
- `Vortex2D::Renderer::UniformBuffer`
- `Vortex2D::Renderer::VertexBuffer`
- `Vortex2D::Renderer::Work`

API Reference

namespace `Renderer`

Typedefs

```
using Vortex2D::Renderer::DescriptorTypeBindings = typedef std::map<uint32_t, vk::Desc
typedef std::vector<glm::vec2> Path
```

Functions

void **BufferBarrier** (vk::Buffer *buffer*, vk::CommandBuffer *commandBuffer*, vk::AccessFlags *oldAccess*, vk::AccessFlags *newAccess*)
 Inserts a barrier for the given buffer, command buffer and access.

Parameters

- *buffer*: the vulkan buffer handle
- *commandBuffer*: the command buffer to inserts the barrier
- *oldAccess*: old access
- *newAccess*: new access

```
template <typename > class BufferType, typename T>
void CopyTo (BufferType<T> &buffer, T &t)
  Copy the content of a buffer in an object.
```

```
template <typename > class BufferType, typename T>
void CopyTo (BufferType<T> &buffer, std::vector<T> &t)
  Copy the content of a buffer to a vector. Vector needs to have the correct size already.
```

```
template <typename > class BufferType, typename T>
void CopyFrom (BufferType<T> &buffer, const T &t)
  Copy the content of an object to the buffer.
```

```
template <typename > class BufferType, typename T>
void CopyFrom (BufferType<T> &buffer, const std::vector<T> &t)
  Copy the content of a vector to the buffer.
```

void **VORTEX2D_API Vortex2D::Renderer::ExecuteCommand**(const Device & device, CommandB
 Runs immediately a set of commands and waits for them to finish.

Parameters

- `device`: vulkan device
- `commandFn`: lambda that runs the commands.

`bool operator==(const ShaderLayout &left, const ShaderLayout &right)`

`bool operator==(const PipelineLayout &left, const PipelineLayout &right)`

`vk::DescriptorType GetDescriptorType (uint32_t bind, const PipelineLayout &layout)`

VORTEX2D_API void Vortex2D::Renderer::Bind(const Device & device, vk::DescriptorSet & dstSet)
Bind the resources (buffer or texture/sampler) to a *DescriptorSet*.

Parameters

- `device`: vulkan device
- `dstSet`: vulkan descriptor set
- `layout`: pipeline layout
- `bindingInputs`: list of resources (buffer or texture/sampler)

`bool HasLayer(const char *extension, const std::vector<vk::LayerProperties> &availableExtensions)`

`bool HasExtension(const char *extension, const std::vector<vk::ExtensionProperties> &availableExtensions)`

VORTEX2D_API vk::UniquePipeline Vortex2D::Renderer::MakeComputePipeline(vk::Device dev,

`bool operator==(const RenderState &left, const RenderState right)`

VORTEX2D_API ComputeSize Vortex2D::Renderer::MakeStencilComputeSize(const glm::ivec2 &size)
Create a *ComputeSize* for a stencil type shader.

Return calculate *ComputeSize*

Parameters

- `size`: the domain size
- `radius`: the stencil size

VORTEX2D_API ComputeSize Vortex2D::Renderer::MakeCheckerboardComputeSize(const glm::ivec2 &size)
Create a *ComputeSize* for a checkerboard type shader.

Return calculate *ComputeSize*

Parameters

- `size`: the domain size

class AbstractShape : public Vortex2D::Renderer::Drawable, public Vortex2D::Renderer::Transformable
`#include <Shapes.h>` An polygonal shape where the fragment shader can be specified for customisation.

Subclassed by *Vortex2D::Renderer::IntRectangle*, *Vortex2D::Renderer::Rectangle*

Public Functions

AbstractShape (**const** *Device* &device, **const** *SpirvBinary* &fragShader, **const** std::vector<glm::vec2> &vertices)

AbstractShape (*AbstractShape* &&other)

virtual VORTEX2D_API ~**AbstractShape** ()

void **Initialize** (**const** *RenderState* &renderState)

Initialize the drawable for a particular state. This might include creating the correct pipeline. If it was already initialized, it will do nothing.

Parameters

- renderState: the state to initialize with.

void **Update** (**const** glm::mat4 &projection, **const** glm::mat4 &view)

Update the MVP matrix of the drawable.

Parameters

- projection: the projection matrix
- view: the view matrix

void **Draw** (vk::CommandBuffer commandBuffer, **const** *RenderState* &renderState)

Draw for the given render state. This has to be initialized before.

Parameters

- commandBuffer: the command buffer to record into.
- renderState: the render state to use.

Public Members

glm::vec4 Colour

class AbstractSprite : **public** Vortex2D::Renderer::Drawable, **public** Vortex2D::Renderer::Transformable
#include <Sprite.h> a *Sprite*, i.e. a drawable that can render a texture. The fragment shader can be specified for customisation.

Subclassed by *Vortex2D::Fluid::DistanceField*, *Vortex2D::Renderer::Sprite*

Public Functions

AbstractSprite (**const** *Device* &device, **const** *SpirvBinary* &fragShaderName, *Texture* &texture)

AbstractSprite (*AbstractSprite* &&other)

virtual VORTEX2D_API ~**AbstractSprite** ()

void **Initialize** (**const** *RenderState* &renderState)

Initialize the drawable for a particular state. This might include creating the correct pipeline. If it was already initialized, it will do nothing.

Parameters

- renderState: the state to initialize with.

void **Update** (**const** glm::mat4 &*projection*, **const** glm::mat4 &*view*)
Update the MVP matrix of the drawable.

Parameters

- *projection*: the projection matrix
- *view*: the view matrix

void **Draw** (vk::CommandBuffer *commandBuffer*, **const** *RenderState* &*renderState*)
Draw for the given render state. This has to be initialized before.

Parameters

- *commandBuffer*: the command buffer to record into.
- *renderState*: the render state to use.

template <typename T>
void **PushConstant** (vk::CommandBuffer *commandBuffer*, uint32_t *offset*, **const** T &*data*)

Public Members

glm::vec4 **Colour** = { 1.0f, 1.0f, 1.0f, 1.0f }

struct BindingInput
#include <*DescriptorSet.h*> The texture/sampler or buffer that can be binded to a shader.

Public Functions

BindingInput (*Renderer::GenericBuffer* &*buffer*, uint32_t *bind* = *DefaultBind*)

BindingInput (*Renderer::Texture* &*texture*, uint32_t *bind* = *DefaultBind*)

BindingInput (vk::Sampler *sampler*, *Renderer::Texture* &*texture*, uint32_t *bind* = *DefaultBind*)

Public Members

uint32_t **Bind**

mpark::variant<*Renderer::GenericBuffer* *, *DescriptorImage*> **Input**

Public Static Attributes

constexpr uint32_t **DefaultBind** = static_cast<uint32_t>(-1)

template <typename T>
class Buffer : **public** Vortex2D::*Renderer::GenericBuffer*
#include <*Buffer.h*> a storage buffer type of buffer

Public Functions

Buffer (**const** *Device* &*device*, std::size_t *size* = 1, VmaMemoryUsage *memoryUsage* = VMA_MEMORY_USAGE_GPU_ONLY)

class Clear : **public** Vortex2D::*Renderer::Drawable*
#include <*Shapes.h*> A drawable that simply clears the target.

Public Functions

Clear (**const** glm::vec4 &colour)

void **Initialize** (**const** *RenderState* &renderState)

Initialize the drawable for a particular state. This might include creating the correct pipeline. If it was already initialized, it will do nothing.

Parameters

- renderState: the state to initialize with.

void **Update** (**const** glm::mat4 &projection, **const** glm::mat4 &view)

Update the MVP matrix of the drawable.

Parameters

- projection: the projection matrix
- view: the view matrix

void **Draw** (vk::CommandBuffer *commandBuffer*, **const** *RenderState* &renderState)

Draw for the given render state. This has to be initialized before.

Parameters

- commandBuffer: the command buffer to record into.
- renderState: the render state to use.

class CommandBuffer

#include <CommandBuffer.h> Can record commands, then submit them (multiple times). A fence can be used to wait on the completion of the commands.

Public Types

using CommandFn = std::function<void (vk::CommandBuffer) >

Public Functions

CommandBuffer (**const** *Device* &device, bool *synchronise* = true)

Creates a command buffer which can be synchronized.

Parameters

- device: vulkan device
- synchronise: flag to determine if the command buffer can be waited on.

~CommandBuffer ()

CommandBuffer (*CommandBuffer* &&*other*)

CommandBuffer &**operator=** (*CommandBuffer* &&*other*)

void **Record** (*CommandFn* *commandFn*)

Record some commands. The commands are recorded in the lambda which is immediately executed.

Parameters

- commandFn: a functor, or simply a lambda, where commands are recorded.

void **Record** (**const** *RenderTarget* &renderTarget, vk::Framebuffer framebuffer, *CommandFn* *commandFn*)

Record some commands inside a render pass. The commands are recorded in the lambda which is immediately executed.

Parameters

- renderTarget: the render target which contains the render pass to record into
- framebuffer: the frame buffer where the render pass will render.
- commandFn: a functor, or simply a lambda, where commands are recorded.

void **Wait** ()

Wait for the command submit to finish. Does nothing if the synchronise flag was false.

void **Reset** ()

Reset the command buffer so it can be recorded again.

void **Submit** (**const** std::initializer_list<vk::Semaphore> &waitSemaphores = {}, **const** std::initializer_list<vk::Semaphore> &signalSemaphores = {})

submit the command buffer

operator bool () **const**

explicit conversion operator to bool, indicates if the command was properly recorded and can be submitted.

struct ComputeSize

#include <Work.h> Used for a compute shader, and defines the group size, local size and domain size.

Public Functions

ComputeSize (**const** glm::ivec2 &size)

Creates a *ComputeSize* using a 2D domain size and the default 2D local size.

Parameters

- size: the domain size

ComputeSize (int size)

Creates a *ComputeSize* using a 1D domain size and the default 1D local size.

Parameters

- size: the domain size

Public Members

glm::ivec2 **DomainSize**

glm::ivec2 **WorkSize**

glm::ivec2 **LocalSize**

Public Static Functions

glm::ivec2 **GetLocalSize2D** ()

The default local size for 2D compute shaders.

Return a 2d vector

int **GetLocalSize1D** ()

The default local size for 1D compute shaders.

Return a integer value

glm::ivec2 **GetWorkSize** (const glm::ivec2 &size)

Computes the 2D group size given a domain size.

Return the group size

Parameters

- size: the domain size of the shader

glm::ivec2 **GetWorkSize** (int size)

Computes the 1D group size given a domain size.

Return the group size

Parameters

- size: the domain size of the shader

ComputeSize **Default2D** ()

A default *ComputeSize* using the default 2D local size. The domain size is (1,1)

Return a default compute size

ComputeSize **Default1D** ()

A default *ComputeSize* using the default 1D local size. The domain size is (1,1)

Return a default compute size

struct DescriptorImage

#include <DescriptorSet.h> The texture or sampler that can be binded to a shader.

Public Functions

DescriptorImage (vk::Sampler sampler, *Renderer::Texture* &texture)

DescriptorImage (*Renderer::Texture* &texture)

Public Members

vk::Sampler **Sampler**

Renderer::Texture ***Texture**

struct DescriptorSet

#include <DescriptorSet.h> The binding of an object for a shader.

Public Members

vk::UniqueDescriptorSet **descriptorSet**

vk::PipelineLayout **pipelineLayout**

vk::DescriptorSetLayout **descriptorSetLayout**

class Device

#include <Device.h> Encapsulation around the vulkan device. Allows to create command buffers, layout, bindings, memory and shaders.

Public Functions

```
Device (vk::PhysicalDevice physicalDevice, bool validation = true)

Device (vk::PhysicalDevice physicalDevice, vk::SurfaceKHR surface, bool validation = true)

Device (vk::PhysicalDevice physicalDevice, int familyIndex, bool validation = true)

~Device ()

Device (Device&&)

Device &operator= (Device&&)

vk::Device Handle () const

vk::Queue Queue () const

vk::PhysicalDevice GetPhysicalDevice () const

int GetFamilyIndex () const

std::vector<vk::CommandBuffer> CreateCommandBuffers (uint32_t size) const

void FreeCommandBuffers (vk::ArrayProxy<const vk::CommandBuffer> commandBuffers)
                        const

VmaAllocator Allocator () const

LayoutManager &GetLayoutManager () const

vk::ShaderModule GetShaderModule (const SpirvBinary &spirv) const

struct DispatchParams
    #include <Work.h> Parameters for indirect compute: group size, local size, etc.
```

Public Functions

```
DispatchParams (int count)
```

Public Members

```
vk::DispatchIndirectCommand workSize

uint32_t count

struct Drawable
    #include <Drawable.h> Interface of a drawable object.

    Subclassed by Vortex2D::Fluid::Circle, Vortex2D::Fluid::Polygon, Vortex2D::Renderer::AbstractShape,
    Vortex2D::Renderer::AbstractSprite, Vortex2D::Renderer::Clear, Vortex2D::Renderer::Ellipse
```


Public Functions

virtual ~Drawable()

virtual void Initialize (**const** *RenderState* &renderState) = 0

Initialize the drawable for a particular state. This might include creating the correct pipeline. If it was already initialized, it will do nothing.

Parameters

- renderState: the state to initialize with.

virtual void Update (**const** glm::mat4 &projection, **const** glm::mat4 &view) = 0

Update the MVP matrix of the drawable.

Parameters

- projection: the projection matrix
- view: the view matrix

virtual void Draw (vk::CommandBuffer commandBuffer, **const** *RenderState* &renderState) = 0

Draw for the given render state. This has to be initialized before.

Parameters

- commandBuffer: the command buffer to record into.
- renderState: the render state to use.

class Ellipse: **public** Vortex2D::*Renderer::Drawable*, **public** Vortex2D::*Renderer::Transformable*
#include <Shapes.h> A solid colour ellipse. Implements the *Drawable* interface and *Transformable* interface.

Public Functions

Ellipse (**const** *Device* &device, **const** glm::vec2 &radius)

void Initialize (**const** *RenderState* &renderState)

Initialize the drawable for a particular state. This might include creating the correct pipeline. If it was already initialized, it will do nothing.

Parameters

- renderState: the state to initialize with.

void Update (**const** glm::mat4 &projection, **const** glm::mat4 &view)

Update the MVP matrix of the drawable.

Parameters

- projection: the projection matrix
- view: the view matrix

void Draw (vk::CommandBuffer commandBuffer, **const** *RenderState* &renderState)

Draw for the given render state. This has to be initialized before.

Parameters

- commandBuffer: the command buffer to record into.
- renderState: the render state to use.

Public Members

glm::vec4 **Colour**

class GenericBuffer

#include <Buffer.h> A vulkan buffer which can be on the host or the device.

Subclassed by *Vortex2D::Renderer::Buffer< float >*, *Vortex2D::Renderer::Buffer< glm::ivec2 >*, *Vortex2D::Renderer::Buffer< glm::vec2 >*, *Vortex2D::Renderer::Buffer< int >*, *Vortex2D::Renderer::Buffer< Renderer::DispatchParams >*, *Vortex2D::Renderer::Buffer< Vortex2D::Fluid::Particle >*, *Vortex2D::Renderer::Buffer< Vortex2D::Fluid::RigidBody::Velocity >*, *Vortex2D::Renderer::Buffer< Vortex2D::Fluid::RigidBody::Vortex2D::Fluid::Velocity >*, *Vortex2D::Renderer::IndirectBuffer< Renderer::DispatchParams >*, *Vortex2D::Renderer::UniformBuffer< glm::mat4 >*, *Vortex2D::Renderer::UniformBuffer< glm::vec4 >*, *Vortex2D::Renderer::UniformBuffer< Size >*, *Vortex2D::Renderer::UniformBuffer< Vortex2D::Fluid::RigidBody::Velocity >*, *Vortex2D::Renderer::VertexBuffer< glm::vec2 >*, *Vortex2D::Renderer::VertexBuffer< Vortex2D::Renderer::AbstractSprite::Vertex >*, *Vortex2D::Renderer::Buffer< T >*, *Vortex2D::Renderer::IndirectBuffer< T >*, *Vortex2D::Renderer::UniformBuffer< T >*, *Vortex2D::Renderer::VertexBuffer< T >*

Public Functions

GenericBuffer (**const** *Device* &device, vk::BufferUsageFlags *usageFlags*, VmaMemoryUsage *memoryUsage*, vk::DeviceSize *deviceSize*)

~GenericBuffer ()

GenericBuffer (*GenericBuffer* &&*other*)

void **CopyFrom** (vk::CommandBuffer *commandBuffer*, *GenericBuffer* &*srcBuffer*)

Copy a buffer to this buffer.

Parameters

- *commandBuffer*: command buffer to run the copy on.
- *srcBuffer*: the source buffer.

void **CopyFrom** (vk::CommandBuffer *commandBuffer*, *Texture* &*srcTexture*)

Copy a texture to this buffer.

Parameters

- *commandBuffer*: command buffer to run the copy on.
- *srcTexture*: the source texture

vk::Buffer **Handle** () **const**

The vulkan handle.

vk::DeviceSize **Size** () **const**

The size in bytes of the buffer.

void **Barrier** (vk::CommandBuffer *commandBuffer*, vk::AccessFlags *oldAccess*, vk::AccessFlags *newAccess*)

Inserts a barrier for this buffer.

Parameters

- *commandBuffer*: the command buffer to run the barrier
- *oldAccess*: old access
- *newAccess*: new access

void **Clear** (vk::CommandBuffer *commandBuffer*)

Clear the buffer with 0.

Parameters

- *commandBuffer*: the command buffer to clear on

Friends

template <template< typename > class *BufferType*, **typename** T>

void **CopyFrom** (*BufferType*<T> &*buffer*, **const** T &*t*)

Copy the content of an object to the buffer.

template <template< typename > class *BufferType*, **typename** T>

void **CopyTo** (*BufferType*<T> &*buffer*, T &*t*)

Copy the content of a buffer in an object.

template <template< typename > class *BufferType*, **typename** T>

void **CopyTo** (*BufferType*<T> &*buffer*, std::vector<T> &*t*)

Copy the content of a buffer to a vector. Vector needs to have the correct size already.

template <template< typename > class *BufferType*, **typename** T>

void **CopyFrom** (*BufferType*<T> &*buffer*, **const** std::vector<T> &*t*)

Copy the content of a vector to the buffer.

class **GraphicsPipeline**

#include <*Pipeline.h*> graphics pipeline which caches the pipeline per render states.

Public Functions

GraphicsPipeline ()

GraphicsPipeline (*Builder* *builder*)

void **Create** (vk::Device *device*, **const** *RenderState* &*renderState*)

void **Bind** (vk::CommandBuffer *commandBuffer*, **const** *RenderState* &*renderState*)

class **Builder**

#include <*Pipeline.h*> *Builder* for graphics pipeline.

Public Functions

Builder ()

GraphicsPipeline::Builder &**Shader** (vk::ShaderModule *shader*, vk::ShaderStageFlagBits *shaderStage*)

Set the shader.

Return *this

Parameters

- *shader*: the loaded shader
- *shaderStage*: shader state (vertex, fragment or compute)

GraphicsPipeline::Builder &**VertexAttribute** (uint32_t *location*, uint32_t *binding*, vk::Format *format*, uint32_t *offset*)

Sets the vertex attributes.

Return *this

Parameters

- **location**: location in the shader
- **binding**: binding in the shader
- **format**: vertex format
- **offset**: offset in the vertex

```
GraphicsPipeline::Builder &VertexBinding (uint32_t binding,      uint32_t stride,  
                                           vk::VertexInputRate inputRate      =  
                                           vk::VertexInputRate::eVertex)
```

Sets the vertex binding.

Return *this

Parameters

- **binding**: binding in the shader
- **stride**: stride in bytes
- **inputRate**: input rate

```
GraphicsPipeline::Builder &Topology (vk::PrimitiveTopology topology)
```

```
GraphicsPipeline::Builder &Layout (vk::PipelineLayout pipelineLayout)
```

```
vk::UniquePipeline Create (vk::Device device, const RenderState &renderState)
```

```
template <typename T>
```

```
class IndirectBuffer : public Vortex2D::Renderer::GenericBuffer
```

```
    #include <Buffer.h> an indirect buffer type of buffer, used for compute indirect dispatch
```

Public Functions

```
IndirectBuffer (const Device &device,      VmaMemoryUsage memoryUsage      =  
              VMA_MEMORY_USAGE_GPU_ONLY)
```

```
class Instance
```

```
    #include <Instance.h> Vulkan instance, which extensions enabled.
```

Public Functions

```
Instance (const std::string &name, std::vector<const char *> extensions, bool validation)
```

```
~Instance ()
```

```
vk::PhysicalDevice GetPhysicalDevice () const
```

```
vk::Instance GetInstance () const
```

```
class IntRectangle : public Vortex2D::Renderer::AbstractShape
```

```
    #include <Shapes.h> A solid colour rectangle as Rectangle, however uses integer colors and is meant to  
    be drawn to a framebuffer wiht integer colours.
```

Public Functions

```
IntRectangle (const Device &device, const glm::vec2 &size)
```

```
class LayoutManager
```

```
    #include <DescriptorSet.h> Caches and creates layouts and bindings.
```

Public Functions

```
LayoutManager (const Device &device)
```

```
void CreateDescriptorPool ()
```

```
    Create or re-create the descriptor pool, will render invalid existing descriptor sets.
```

```
DescriptorSet MakeDescriptorSet (const PipelineLayout &layout)
```

```
    Create the descriptor set given the layout.
```

```
    Return built descriptor set
```

```
    Parameters
```

- layout: pipeline/shader layout

```
vk::DescriptorSetLayout GetDescriptorSetLayout (const PipelineLayout &layout)
```

```
    Create, cache and return a descriptor layout given the pipeline layout.
```

```
    Return cached descriptor set layout
```

```
    Parameters
```

- layout: pipeline layout

```
vk::PipelineLayout GetPipelineLayout (const PipelineLayout &layout)
```

```
    create, cache and return a vulkan pipeline layout given the layout
```

```
    Return vulkan pipeline layout
```

```
    Parameters
```

- layout: pipeline layout

```
struct PipelineLayout
```

```
    #include <DescriptorSet.h> Represents the layout of a pipeline: vertex + fragment or compute.
```

Public Members

```
std::vector<ShaderLayout> layouts
```

```
template <typename T>
```

```
class property
```

```
    #include <Transformable.h> Simple class to simulate properties like in C#.
```

Public Functions

```
property (Transformable &t)
```

```
T &operator= (const T &i)
```

```
operator T const& () const
```

```
class Rectangle : public Vortex2D::Renderer::AbstractShape
```

```
    #include <Shapes.h> A solid colour rectangle defined by two triangles. Implements the Drawable interface and Transformable interface.
```

Public Functions

Rectangle (**const** *Device* &device, **const** glm::vec2 &size)

class **RenderCommand**

#include <CommandBuffer.h> A special command buffer that has been recorded by a *RenderTarget*. It can be used to submit the rendering. The object has to stay alive until rendering is complete.

Public Functions

RenderCommand ()

~RenderCommand ()

RenderCommand (*RenderCommand* &&other)

RenderCommand &**operator=** (*RenderCommand* &&other)

void **Submit** (**const** glm::mat4 &view = glm::mat4())
Submit the render command with a transform matrix.

Parameters

- view: a transform matrix

Friends

friend Vortex2D::Renderer::RenderTexture

friend Vortex2D::Renderer::RenderWindow

class **RenderpassBuilder**

#include <RenderTarget.h> Factory for a vulkan render pass.

Public Functions

RenderpassBuilder &**Attachement** (vk::Format format)
Format of the render pass.

Return

Parameters

- format:

RenderpassBuilder &**AttachementLoadOp** (vk::AttachmentLoadOp value)
operation to perform when loading the framebuffer (clear, load, etc)

Return

Parameters

- value:

RenderpassBuilder &**AttachementStoreOp** (vk::AttachmentStoreOp value)
operation to perform when storing the framebuffer (clear, save, etc)

Return

Parameters

- value:

RenderpassBuilder & **AttachementInitialLayout** (vk::ImageLayout *layout*)

Layout of the image to be before render pass.

Return

Parameters

- layout:

RenderpassBuilder & **AttachementFinalLayout** (vk::ImageLayout *layout*)

Layout of the image to be after render pass.

Return

Parameters

- layout:

RenderpassBuilder & **Subpass** (vk::PipelineBindPoint *bindPoint*)

Define subpass of the render pass.

Return

Parameters

- bindPoint:

RenderpassBuilder & **SubpassColorAttachment** (vk::ImageLayout *layout*, uint32_t *attachment*)

Set the color attachment with index.

Return

Parameters

- layout:
- attachment: index of the attachment

RenderpassBuilder & **Dependency** (uint32_t *srcSubpass*, uint32_t *dstSubpass*)

Dependency of the subpasses.

Return

Parameters

- srcSubpass:
- dstSubpass:

RenderpassBuilder & **DependencySrcStageMask** (vk::PipelineStageFlags *value*)

RenderpassBuilder & **DependencyDstStageMask** (vk::PipelineStageFlags *value*)

RenderpassBuilder & **DependencySrcAccessMask** (vk::AccessFlags *value*)

RenderpassBuilder & **DependencyDstAccessMask** (vk::AccessFlags *value*)

vk::UniqueRenderPass **Create** (vk::Device *device*)

Create the render pass.

Return

Parameters

- device:

class RenderState

#include <RenderState.h> the various state to render to a target: size, render pass and blend.

Public Functions

RenderState (**const** *RenderTarget* &*renderTarget*)

Initialize for a render target with default blend.

Parameters

- *renderTarget*:

RenderState (**const** *RenderTarget* &*renderTarget*, vk::PipelineColorBlendAttachmentState *colorBlend*)

Initialize for a render target with a given blend.

Parameters

- *renderTarget*:
- *colorBlend*:

Public Members

uint32_t **Width**

uint32_t **Height**

vk::RenderPass **RenderPass**

vk::PipelineColorBlendAttachmentState **ColorBlend**

struct RenderTarget

#include <RenderTarget.h> A target that can be rendered to. This is implemented by the *RenderWindow* and the *RenderTexture*.

Subclassed by *Vortex2D::Renderer::RenderTexture*, *Vortex2D::Renderer::RenderWindow*

Public Types

using DrawableList = std::initializer_list<std::reference_wrapper<*Drawable*>>

Public Functions

RenderTarget (uint32_t *width*, uint32_t *height*)

RenderTarget (*RenderTarget* &&*other*)

~RenderTarget ()

virtual VORTEX2D_API RenderCommand Vortex2D::Renderer::RenderTarget::Record (DrawableList &&*drawables*)

virtual VORTEX2D_API void Vortex2D::Renderer::RenderTarget::Submit (RenderCommand &&*command*)

Public Members

uint32_t **Width**

uint32_t **Height**

glm::mat4 **Orth**

glm::mat4 **View**

vk::UniqueRenderPass **RenderPass**

class RenderTexture : public Vortex2D::Renderer::RenderTarget, public Vortex2D::Renderer::Texture
#include <RenderTexture.h> A render target that renders into a texture.

Subclassed by *Vortex2D::Fluid::Density*, *Vortex2D::Fluid::LevelSet*, *Vortex2D::Fluid::ParticleCount*,
Vortex2D::Fluid::Velocity

Public Functions

RenderTexture (const *Device* &device, uint32_t width, uint32_t height, vk::Format format)

RenderCommand **Record** (DrawableList drawables, vk::PipelineColorBlendAttachmentState
blendMode = {})

void **Submit** (*RenderCommand* &renderCommand)

class RenderWindow : public Vortex2D::Renderer::RenderTarget
#include <RenderWindow.h> Render to a swapchain, i.e. to the window/surface.

Public Functions

RenderWindow (const *Device* &device, vk::SurfaceKHR surface, uint32_t width, uint32_t
height)

Initialize with a given surface and size.

Parameters

- device: vulkan device
- surface: vulkan surface
- width:
- height:

RenderCommand **Record** (DrawableList drawables, vk::PipelineColorBlendAttachmentState
blendMode = {})

void **Submit** (*RenderCommand* &renderCommand)

void **Display** ()

Submits all the render command and present the surface for display.

class SamplerBuilder
#include <Texture.h> Factory for a vullkan sampler.

Public Functions

SamplerBuilder ()

SamplerBuilder &**AddressMode** (vk::SamplerAddressMode mode)
Mode of the sampler: repeat, clamp, etc.

Return *this

Parameters

- mode: vulkan mode

SamplerBuilder &**Filter** (vk::Filter *filter*)

Filter of the sampler: linear, nearest, etc.

Return *this

Parameters

- *filter*: vulkan filter

vk::UniqueSampler **Create** (vk::Device *device*)

Create the vulkan sampler.

Return unique sampler

Parameters

- *device*: vulkan device

struct ShaderLayout

#include <DescriptorSet.h> Represents the layout of a shader (vertex, fragment or compute)

Public Functions

ShaderLayout (const SPIRV::Reflection &*reflection*)

Public Members

vk::ShaderStageFlags **shaderStage**

DescriptorTypeBindings **bindings**

unsigned **pushConstantSize**

class SpirvBinary

#include <Device.h> A binary SPIRV shader, to be feed to vulkan.

Public Functions

template <std::size_t *N*>

SpirvBinary (const uint32_t (&*spirv*)[*N*])

const uint32_t ***data** () const

std::size_t **size** () const

std::size_t **words** () const

class Sprite : public Vortex2D::Renderer::AbstractSprite

#include <Sprite.h> A sprite that renders a texture with a simple pass-through fragment shader.

Subclassed by *Vortex2D::Fluid::Density*

Public Functions

Sprite (const *Device* &*device*, *Texture* &*texture*)

struct SwapChainSupportDetails

Public Functions

SwapChainSupportDetails (vk::PhysicalDevice *device*, vk::SurfaceKHR *surface*)

bool **IsValid**() const

Public Members

vk::SurfaceCapabilitiesKHR **capabilities**

std::vector<vk::SurfaceFormatKHR> **formats**

std::vector<vk::PresentModeKHR> **presentModes**

class Texture

#include <Texture.h> A texture, or in vulkan terms, an image.

Subclassed by *Vortex2D::Renderer::RenderTexture*

Public Functions

Texture (const *Device* &*device*, uint32_t *width*, uint32_t *height*, vk::Format *format*, VmaMemoryUsage *memoryUsage* = VMA_MEMORY_USAGE_GPU_ONLY)

~**Texture** ()

Texture (*Texture* &&*other*)

template <typename T>

void **CopyFrom** (const std::vector<T> &*data*)

template <typename T>

void **CopyTo** (std::vector<T> &*data*)

void **CopyFrom** (vk::CommandBuffer *commandBuffer*, *Texture* &*srcImage*)

void **Barrier** (vk::CommandBuffer *commandBuffer*, vk::ImageLayout *oldLayout*, vk::AccessFlags *oldAccess*, vk::ImageLayout *newLayout*, vk::AccessFlags *newAccess*)

vk::ImageView **GetView** () const

uint32_t **GetWidth** () const

uint32_t **GetHeight** () const

vk::Format **GetFormat** () const

void **Clear** (vk::CommandBuffer *commandBuffer*, const std::array<int, 4> &*colour*)

void **Clear** (vk::CommandBuffer *commandBuffer*, const std::array<float, 4> &*colour*)

Friends

friend Vortex2D::Renderer::GenericBuffer

class Timer

#include <Timer.h> Calculates the elapsed time on the GPU.

Public Functions

Timer (**const** *Device* &device)

void **Start** ()

Start the timer after the current last command buffer.

void **Stop** ()

Stop the timer after the current last command buffer.

uint64_t **GetElapsedNs** ()

Get the elapsed time between the Start and Stop calls. Blocking function which will download the timestamps from the GPU.

Return timestamp in nanoseconds.

struct Transformable

#include <Transformable.h> Class to represent the transformation of an object: position, scale, rotation and anchor.

Subclassed by *Vortex2D::Fluid::Circle*, *Vortex2D::Fluid::Polygon*, *Vortex2D::Fluid::RigidBody*, *Vortex2D::Renderer::AbstractShape*, *Vortex2D::Renderer::AbstractSprite*, *Vortex2D::Renderer::Ellipse*

Public Functions

Transformable ()

virtual ~Transformable ()

const glm::mat4 &**GetTransform** () **const**

Returns the transform matrix.

void **Update** ()

Update the transform matrix with the position, scale, rotation and anchor.

Public Members

property<glm::vec2> **Position**

absolute position

property<glm::vec2> **Scale**

scale for the x and y components

property<float> **Rotation**

Rotation in radians.

property<glm::vec2> **Anchor**

An offset to the position (used for centering a shape)

template <typename T>

class UniformBuffer : **public** Vortex2D::Renderer::GenericBuffer

#include <Buffer.h> a uniform buffer type of buffer

Public Functions

```
UniformBuffer(const Device &device, VmaMemoryUsage memoryUsage =
              VMA_MEMORY_USAGE_GPU_ONLY)

template <typename T>
class VertexBuffer: public Vortex2D::Renderer::GenericBuffer
    #include <Buffer.h> a vertex buffer type of buffer
```

Public Functions

```
VertexBuffer(const Device &device, std::size_t size, VmaMemoryUsage memoryUsage =
              VMA_MEMORY_USAGE_GPU_ONLY)

class Work
    #include <Work.h> Represents a compute shader. It simplifies the process of binding, setting push con-
    stants and recording.
```

Public Functions

```
Work(const Device &device, const ComputeSize &computeSize, const SpirvBinary &spirv)
    Constructs an object using a SPIRV binary. It is not bound to any buffers or textures.
```

Parameters

- device: vulkan device
- computeSize: the compute size. Can be a default one with size (1,1) or one with an actual size.
- spirv: binary spirv

```
Work::Bound Bind(const std::vector<BindingInput> &inputs)
```

Bind the buffers and/or textures.

Return a bound object, ready to be recorded in a command buffer.

Parameters

- inputs: a list of buffers and/or textures

```
VORTEX2D_API Bound Vortex2D::Renderer::Work::Bind(ComputeSize computeSize, const st
```

Bind the buffers and/or textures. This overrides the provided compute size in *Work*.

Return a bound object, ready to be recorded in a command buffer.

Parameters

- computeSize: the compute shader compute size.
- inputs: a list of buffers and/or textures

```
class Bound
```

#include <Work.h> Is a bound version of *Work*. This means a buffer or texture was bound and this can be recorded in a command buffer.

Public Functions

```
Bound()
```

```
template <typename T>
```

void **PushConstant** (vk::CommandBuffer *commandBuffer*, uint32_t *offset*, **const** T &*data*)
Adds a constant value, i.e. a push constant.

Parameters

- *commandBuffer*: the command buffer where the compute work will also be recorded.
- *offset*: the offset in which the push constant is to be placed.
- *data*: the data to push. A total of 128 bytes can be used.

void **Record** (vk::CommandBuffer *commandBuffer*)
Record the compute work in this command buffer. This will also set two additional push constants: the 2D domain size.

Parameters

- *commandBuffer*: the command buffer to record into.

void **RecordIndirect** (vk::CommandBuffer *commandBuffer*, *Indirect-Buffer*<*DispatchParams*> &*dispatchParams*)
Record the compute work in this command buffer. Use the provided parameters to run the compute shader.

Parameters

- *commandBuffer*: the command buffer to record into.
- *dispatchParams*: the indirect buffer containing the parameters.

void **Dispatch** (vk::CommandBuffer *commandBuffer*)
Record the compute work in this command buffer.

Parameters

- *commandBuffer*: the command buffer to record into.

Friends

friend Vortex2D::Renderer::Work

1.2.6 Fluid API reference

Classes

- *Vortex2D::Fluid::Advection*
- *Vortex2D::Fluid::Circle*
- *Vortex2D::Fluid::ConjugateGradient*
- *Vortex2D::Fluid::Density*
- *Vortex2D::Fluid::Depth*
- *Vortex2D::Fluid::Diagonal*
- *Vortex2D::Fluid::Dimensions*
- *Vortex2D::Fluid::DistanceField*
- *Vortex2D::Fluid::Extrapolation*
- *Vortex2D::Fluid::GaussSeidel*
- *Vortex2D::Fluid::IncompletePoisson*

- `Vortex2D::Fluid::Jacobi`
- `Vortex2D::Fluid::LevelSet`
- `Vortex2D::Fluid::LinearSolver`
- `Vortex2D::Fluid::LocalGaussSeidel`
- `Vortex2D::Fluid::Multigrid`
- `Vortex2D::Fluid::ParticleCount`
- `Vortex2D::Fluid::Polygon`
- `Vortex2D::Fluid::Preconditioner`
- `Vortex2D::Fluid::Pressure`
- `Vortex2D::Fluid::Rectangle`
- `Vortex2D::Fluid::Reduce`
- `Vortex2D::Fluid::ReduceJ`
- `Vortex2D::Fluid::ReduceMax`
- `Vortex2D::Fluid::ReduceSum`
- `Vortex2D::Fluid::RigidBody`
- `Vortex2D::Fluid::SmokeWorld`
- `Vortex2D::Fluid::Transfer`
- `Vortex2D::Fluid::Velocity`
- `Vortex2D::Fluid::WaterWorld`
- `Vortex2D::Fluid::World`

API Reference

namespace Fluid

Functions

Renderer::ComputeSize **MakeLocalSize** (`const glm::ivec2 &size`)

Variables

```
.setBlendEnable(true)          .setColorBlendOp(vk::BlendOp::eMax)          .setSrcColorBlendFactor(vk::BlendFactor::eOne)
                               .setDstColorBlendFactor(vk::BlendFactor::eOne)          .setColorWriteMask(vk::ColorComponentFlagBits::eR) ]
```

```
.setBlendEnable(true)          .setColorBlendOp(vk::BlendOp::eMin)          .setSrcColorBlendFactor(vk::BlendFactor::eOne)
                               .setDstColorBlendFactor(vk::BlendFactor::eOne)          .setColorWriteMask(vk::ColorComponentFlagBits::eR) ]
```

VORTEX2D_API **Renderer::Clear** **Vortex2D::Fluid::BoundariesClear** = *Vortex2D::Renderer::Clear* ({1000

class Advection

#include <Advection.h> Advects particles, velocity field or any field using a velocity field.

Public Functions

Advection (*const* *Renderer::Device* &device, *const* glm::ivec2 &size, float dt, *Velocity* &velocity)

Initialize advection kernels and related object.

Parameters

- device: vulkan device
- size: size of velocity field
- dt: delta time for integration
- velocity: velocity field

void **AdvectVelocity** ()

Self advect velocity.

void **AdvectBind** (*Density* &density)

Binds a density field to be advected.

Parameters

- density: density field

void **Advect** ()

Performs an advection of the density field. Asynchronous operation.

void **AdvectParticleBind** (*Renderer::GenericBuffer* &particles, *Renderer::Texture* &levelSet, *Renderer::IndirectBuffer*<*Renderer::DispatchParams*> &dispatchParams)

Binds particles to be advected. Also use a level set to project out the particles if they enter it.

Parameters

- particles: particles to be advected
- levelSet: level set to project out particles
- dispatchParams: contains number of particles

void **AdvectParticles** ()

Advect particles. Asynchronous operation.

class Circle : public Vortex2D::Renderer::Transformable, public Vortex2D::Renderer::Drawable
#include <Boundaries.h> Signed distance field of circle.

Public Functions

Circle (*const* *Renderer::Device* &device, float radius, float extent = 10.0f)

Initialize the circle with radius and extend of signed distance.

Parameters

- device: vulkan device.
- radius: radius of circle.
- extent: extend how far from the circle the signed distance field is calculated.

void **Initialize** (*const* *Renderer::RenderState* &renderState)

Initialize the drawable for a particular state. This might include creating the correct pipeline. If it was already initialized, it will do nothing.

Parameters

- renderState: the state to initialize with.

void **Update** (**const** glm::mat4 &*projection*, **const** glm::mat4 &*view*)
Update the MVP matrix of the drawable.

Parameters

- *projection*: the projection matrix
- *view*: the view matrix

void **Draw** (vk::CommandBuffer *commandBuffer*, **const** *Renderer::RenderState* &*renderState*)
Draw for the given render state. This has to be initialized before.

Parameters

- *commandBuffer*: the command buffer to record into.
- *renderState*: the render state to use.

class ConjugateGradient : **public** Vortex2D::Fluid::LinearSolver
#include <ConjugateGradient.h> An iterative preconditioned conjugate linear solver. The preconditioner can be specified.

Public Functions

ConjugateGradient (**const** *Renderer::Device* &*device*, **const** glm::ivec2 &*size*, *Preconditioner* &*preconditioner*)
Initialize the solver with a size and preconditioner.

Parameters

- *device*: vulkan device
- *size*:
- *preconditioner*:

void **Bind** (*Renderer::GenericBuffer* &*d*, *Renderer::GenericBuffer* &*l*, *Renderer::GenericBuffer* &*b*, *Renderer::GenericBuffer* &*x*)
Bind the buffers for the linear solver.

Parameters

- *d*: the diagonal of the matrix
- *l*: the lower matrix
- *b*: the right hand side
- *x*: the unknowns

void **Solve** (Parameters &*params*)
Solve iteratively solve the linear equations in data.

class Density : **public** Vortex2D::Renderer::RenderTexture, **public** Vortex2D::Renderer::Sprite
#include <Density.h> *Density* field, used to represent smoke swirling.

Public Functions

Density (**const** *Renderer::Device* &*device*, **const** glm::ivec2 &*size*, vk::Format *format*)

Friends

friend Vortex2D::Fluid::Advection

class Depth
#include <Multigrid.h> Contains the sizes of the multigrid hierarchy.

Public Functions

Depth (**const** glm::ivec2 &*size*)

Initialize with the finest size.

Parameters

- *size*: the base size.

int **GetMaxDepth** () **const**

The calculated depth of the multigrid.

Return the depth.

glm::ivec2 **GetDepthSize** (std::size_t *i*) **const**

Gets the depth for a given level.

Return the size

Parameters

- *i*: the level

class Diagonal : **public** Vortex2D::Fluid::Preconditioner

#include <Diagonal.h> *Diagonal* preconditioner. Simplest of preconditioner, useful to verify if the preconditioned conjugate gradient works.

Public Functions

Diagonal (**const** *Renderer::Device* &*device*, **const** glm::ivec2 &*size*)

void **Bind** (*Renderer::GenericBuffer* &*d*, *Renderer::GenericBuffer* &*l*, *Renderer::GenericBuffer* &*b*,
Renderer::GenericBuffer &*x*)

Bind the linear equation buffers.

Parameters

- *d*: the diagonal of the matrix
- *l*: the lower matrix
- *b*: the right hand side
- *x*: the unknown buffer

void **Record** (vk::CommandBuffer *commandBuffer*)

Record the preconditioner.

Parameters

- *commandBuffer*: the command buffer to record into.

struct Dimensions

#include <Size.h> Represents the size of the fluid grid. This is useful if we want a different resolution or scale than one grid cell per pixel.

Public Functions

Dimensions (**const** glm::ivec2 &*size*, **const** float *scale*)

The grid size will be: size/scale.

Parameters

- *size*: the size at which we will draw boundaries, forces, etc
- *scale*: the scale of the actual grid

Public Members

```
const float Scale
const glm::ivec2 Size
const glm::mat4 InvScale
```

```
class DistanceField : public Vortex2D::Renderer::AbstractSprite
#include <Boundaries.h> Sprite of a distance field.
```

Public Functions

```
DistanceField(const Renderer::Device &device, Renderer::RenderTexture &levelSet, float
              scale = 1.0f)
Initialize the price with the level set and scale.
```

Parameters

- device: vulkan device
- levelSet: level set to use as sprite
- scale: scale of the level set

```
void Draw (vk::CommandBuffer commandBuffer, const Renderer::RenderState &renderState)
Draw for the given render state. This has to be initialized before.
```

Parameters

- commandBuffer: the command buffer to record into.
- renderState: the render state to use.

```
class Extrapolation
#include <Extrapolation.h> Class to extrapolate values into the neumann and/or dirichlet boundaries.
```

Public Functions

```
Extrapolation(const Renderer::Device &device, const glm::ivec2 &size, Ren-
              derer::GenericBuffer &valid, Velocity &velocity, int iterations = 10)
```

```
void Extrapolate ()
Will extrapolate values from buffer into the dirichlet and neumann boundaries.
```

```
void ConstrainBind (Renderer::Texture &solidPhi)
Binds a solid level set to use later and constrain the velocity against.
```

Parameters

- solidPhi: solid level set

```
void ConstrainVelocity ()
Constrain the velocity, i.e. ensure that the velocity normal to the solid level set is 0.
```

```
class GaussSeidel : public Vortex2D::Fluid::LinearSolver, public Vortex2D::Fluid::Preconditioner
#include <GaussSeidel.h> An iterative black and red successive over relaxation linear solver.
```

Public Functions

GaussSeidel (**const** *Renderer::Device* &device, **const** glm::ivec2 &size)

void **Bind** (*Renderer::GenericBuffer* &d, *Renderer::GenericBuffer* &l, *Renderer::GenericBuffer* &b,
Renderer::GenericBuffer &x)

Bind the buffers for the linear solver.

Parameters

- d: the diagonal of the matrix
- l: the lower matrix
- b: the right hand side
- x: the unknowns

void **Solve** (*Parameters* ¶ms)

Iterative solving of the linear equations in data.

void **Record** (vk::CommandBuffer *commandBuffer*)

Record the preconditioner.

Parameters

- *commandBuffer*: the command buffer to record into.

void **Record** (vk::CommandBuffer *commandBuffer*, int *iterations*)

Record a determined number of iterations.

Parameters

- *commandBuffer*:
- *iterations*:

void **SetW** (float *w*)

Set the *w* factor of the GS iterations : $x_{\text{new}} = w * x_{\text{new}} + (1-w) * x_{\text{old}}$.

Parameters

- *w*:

void **SetPreconditionerIterations** (int *iterations*)

set number of iterations to be used when GS is a preconditioner

Parameters

- *iterations*:

class IncompletePoisson : public Vortex2D::Fluid::Preconditioner

#include <IncompletePoisson.h> Incomplete poisson preconditioner. Slightly better than a simple diagonal preconditioner.

Public Functions

IncompletePoisson (**const** *Renderer::Device* &device, **const** glm::ivec2 &size)

void **Bind** (*Renderer::GenericBuffer* &d, *Renderer::GenericBuffer* &l, *Renderer::GenericBuffer* &b,
Renderer::GenericBuffer &x)

Bind the linear equation buffers.

Parameters

- d: the diagonal of the matrix
- l: the lower matrix
- b: the right hand side

- x: the unknown buffer

void **Record** (vk::CommandBuffer *commandBuffer*)

Record the preconditioner.

Parameters

- *commandBuffer*: the command buffer to record into.

struct J

Public Members

glm::vec2 **linear**

float **angular**

class Jacobi : public Vortex2D::Fluid::Preconditioner

#include <Jacobi.h> An iterative jacobi linear solver.

Public Functions

Jacobi (const *Renderer::Device* &*device*, const glm::ivec2 &*size*)

void **Bind** (*Renderer::GenericBuffer* &*d*, *Renderer::GenericBuffer* &*l*, *Renderer::GenericBuffer* &*b*,
Renderer::GenericBuffer &*x*)

Bind the linear equation buffers.

Parameters

- *d*: the diagonal of the matrix
- *l*: the lower matrix
- *b*: the right hand side
- *x*: the unknown buffer

void **Record** (vk::CommandBuffer *commandBuffer*)

Record the preconditioner.

Parameters

- *commandBuffer*: the command buffer to record into.

void **Record** (vk::CommandBuffer *commandBuffer*, int *iterations*)

void **SetW** (float *w*)

Set the w factor of the GS iterations : $x_{\text{new}} = w * x_{\text{new}} + (1-w) * x_{\text{old}}$.

Parameters

- *w*:

void **SetPreconditionerIterations** (int *iterations*)

set number of iterations to be used when GS is a preconditioner

Parameters

- *iterations*:

class LevelSet : public Vortex2D::Renderer::RenderTexture

#include <LevelSet.h> A signed distance field, which can be re-initialized. In other words, a level set.

Public Functions

LevelSet (**const** *Renderer::Device* &device, **const** glm::ivec2 &size, int *reinitializeIterations* = 100)

void **ExtrapolateBind** (*Renderer::Texture* &solidPhi)

Bind a solid level set, which will be used to extrapolate into this level set.

Parameters

- solidPhi:

void **Reinitialise** ()

Reinitialise the level set, i.e. ensure it is a correct signed distance field.

void **Extrapolate** ()

Extrapolate this level set into the solid level set it was attached to. This only performs a single cell extrapolation.

void **ExtrapolateRecord** (vk::CommandBuffer *commandBuffer*)

Same as *Extrapolate* but to be recorded as part of other commands.

Parameters

- commandBuffer:

struct LinearSolver

#include <LinearSolver.h> An interface to represent a linear solver.

Subclassed by *Vortex2D::Fluid::ConjugateGradient*, *Vortex2D::Fluid::GaussSeidel*

Public Functions

virtual ~LinearSolver ()

virtual void Bind (*Renderer::GenericBuffer* &d, *Renderer::GenericBuffer* &l, *Renderer::GenericBuffer* &b, *Renderer::GenericBuffer* &x) = 0

Bind the buffers for the linear solver.

Parameters

- d: the diagonal of the matrix
- l: the lower matrix
- b: the right hand side
- x: the unknowns

virtual void Solve (*Parameters* ¶ms) = 0

Solves the linear equations.

struct Data

#include <LinearSolver.h> The various parts of linear equations.

Public Functions

Data (**const** *Renderer::Device* &device, **const** glm::ivec2 &size, VmaMemoryUsage *memoryUsage* = VMA_MEMORY_USAGE_GPU_ONLY)

Public Members

Renderer::Buffer<float> **Diagonal**

Renderer::Buffer<glm::vec2> **Lower**

Renderer::Buffer<float> **B**

Renderer::Buffer<float> **X**

struct Parameters

#include <LinearSolver.h> *Parameters* for an iterative linear solvers.

Public Functions

Parameters (unsigned *iterations*, float *errorTolerance* = 0.0f)

Construct parameters with max iterations and max error.

Parameters

- *iterations*: max number of iterations to perform
- *errorTolerance*: solver stops when the error is smaller than this.

bool **IsFinished** (unsigned *iterations*, float *error*) **const**

Checks if we've reached the parameters.

Return if we can stop the linear solver.

Parameters

- *iterations*: the current number of iterations
- *error*: the current error

Public Members

unsigned **Iterations**

float **ErrorTolerance**

unsigned **OutIterations**

float **OutError**

class LocalGaussSeidel : **public** Vortex2D::Fluid::Preconditioner

#include <GaussSeidel.h> A version of the gauss seidel that can only be applied on sizes (16,16) or smaller.

Public Functions

LocalGaussSeidel (**const** *Renderer::Device* &device, **const** glm::ivec2 &size)

void **Bind** (*Renderer::GenericBuffer* &d, *Renderer::GenericBuffer* &l, *Renderer::GenericBuffer* &b, *Renderer::GenericBuffer* &x)

Bind the linear equation buffers.

Parameters

- d: the diagonal of the matrix
- l: the lower matrix
- b: the right hand side

- `x`: the unknown buffer

void **Record** (vk::CommandBuffer *commandBuffer*)

Record the preconditioner.

Parameters

- `commandBuffer`: the command buffer to record into.

class Multigrid: public Vortex2D::Fluid::Preconditioner

#include <Multigrid.h> *Multigrid* preconditioner. It creates a hierarchy of twice as small set of linear equations. It applies a few iterations of jacobi on each level and transfers the error on the level above. It then copies the error down, adds to the current solution and apply a few more iterations of jacobi.

Public Functions

Multigrid (const *Renderer::Device* &device, const glm::ivec2 &size, float delta)

Initialize multigrid for given size and delta.

Parameters

- `device`: vulkan device
- `size`: of the linear equations
- `delta`: timestep delta

void **Bind** (*Renderer::GenericBuffer* &d, *Renderer::GenericBuffer* &l, *Renderer::GenericBuffer* &b,
Renderer::GenericBuffer &x)

Bind the linear equation buffers.

Parameters

- `d`: the diagonal of the matrix
- `l`: the lower matrix
- `b`: the right hand side
- `x`: the unknown buffer

void **BuildHierarchiesBind** (*Pressure* &pressure, *Renderer::Texture* &solidPhi, *Renderer::Texture* &liquidPhi)

Bind the level sets from which the hierarchy is built.

Parameters

- `pressure`: The current linear equations
- `solidPhi`: the solid level set
- `liquidPhi`: the liquid level set

void **BuildHierarchies** ()

Computes the hierarchy to be used by the multigrid. Asynchronous operation.

void **Record** (vk::CommandBuffer *commandBuffer*)

Record the preconditioner.

Parameters

- `commandBuffer`: the command buffer to record into.

struct Particle

Public Members

glm::vec2 **Position**

glm::vec2 **Velocity**

class ParticleCount : public Vortex2D::Renderer::RenderTexture
#include <Particles.h> Container for particles used in the advection of the fluid simulation. Also a level set that is built from the particles.

Public Functions

ParticleCount (const *Renderer::Device* &device, const glm::ivec2 &size, *Renderer::GenericBuffer* &particles, const *Renderer::DispatchParams* ¶ms = {0}, float *alpha* = 1.0f)

void **Scan** ()
 Count the number of particles and update the internal data structures.

int **GetTotalCount** ()
 Calculate the total number of particles and return it.

Return

Renderer::IndirectBuffer<*Renderer::DispatchParams*> &**GetDispatchParams** ()
 Calculate the dispatch parameters to use on the particle buffer.

Return

void **LevelSetBind** (*LevelSet* &levelSet)
 Bind a solid level set, which will be used to interpolate the particles out of.

Parameters

- levelSet:

void **Phi** ()
 Calculate the level set from the particles.

void **VelocitiesBind** (*Velocity* &velocity, *Renderer::GenericBuffer* &valid)
 Bind the velocities, used for advection of the particles.

Parameters

- velocity:
- valid:

void **TransferToGrid** ()
 Interpolate the velocities of the particles to the velocities field.

void **TransferFromGrid** ()
 Interpolate the velocities field in to the particles' velocity.

class Polygon : public Vortex2D::Renderer::Transformable, public Vortex2D::Renderer::Drawable
#include <Boundaries.h> Signed distance field of a polygon.

Subclassed by *Vortex2D::Fluid::Rectangle*

Public Functions

Polygon (const *Renderer::Device* &device, std::vector<glm::vec2> *points*, bool *inverse* = false, float *extent* = 10.0f)
 Initialize polygon with set of points and extent of signed distance.

Parameters

- `device`: vulkan device
- `points`: clockwise oriented set of points (mininum 3).
- `inverse`: flag if the distance field should be inversed.
- `extent`: extend how far from the poylon the signed distance field is calculated.

void **Initialize** (`const Renderer::RenderState &renderState`)

Initialize the drawable for a particular state. This might include creating the correct pipeline. If it was already initialized, it will do nothing.

Parameters

- `renderState`: the state to initialize with.

void **Update** (`const glm::mat4 &projection`, `const glm::mat4 &view`)

Update the MVP matrix of the drawable.

Parameters

- `projection`: the projection matrix
- `view`: the view matrix

void **Draw** (`vk::CommandBuffer commandBuffer`, `const Renderer::RenderState &renderState`)

Draw for the given render state. This has to be initialized before.

Parameters

- `commandBuffer`: the command buffer to record into.
- `renderState`: the render state to use.

struct Preconditioner

#include <Preconditioner.h> An interface to represent a linear solver preconditioner.

Subclassed by *Vortex2D::Fluid::Diagonal*, *Vortex2D::Fluid::GaussSeidel*, *Vortex2D::Fluid::IncompletePoisson*, *Vortex2D::Fluid::Jacobi*, *Vortex2D::Fluid::LocalGaussSeidel*, *Vortex2D::Fluid::Multigrid*

Public Functions

virtual `~Preconditioner()`

virtual void **Bind** (`Renderer::GenericBuffer &d`, `Renderer::GenericBuffer &l`, `Renderer::GenericBuffer &b`, `Renderer::GenericBuffer &x`) = 0

Bind the linear equation buffers.

Parameters

- `d`: the diagonal of the matrix
- `l`: the lower matrix
- `b`: the right hand side
- `x`: the unknown buffer

virtual void **Record** (`vk::CommandBuffer commandBuffer`) = 0

Record the preconditioner.

Parameters

- `commandBuffer`: the command buffer to record into.

class PrefixScan

#include <PrefixScan.h> The prefix sum operator.

```
void PrefixSym(int input[], int n, int output[])
{
    output[0] = input[0];

    for (int i = 1; i < n; i++)
        output[i] = output[i-1] + input[i];
}
```

Public Functions

PrefixScan (const *Renderer::Device* &device, const glm::ivec2 &size)

PrefixScan::Bound **Bind** (*Renderer::GenericBuffer* &input, *Renderer::GenericBuffer* &output, *Renderer::GenericBuffer* &dispatchParams)

class Bound

#include <PrefixScan.h> A prefix scan object bound with input/output buffers, ready to be dispatched.

Public Functions

void **Record** (vk::CommandBuffer *commandBuffer*)

Friends

friend Vortex2D::Fluid::PrefixScan

class Pressure

#include <Pressure.h> build the linear equation and compute the divergence from the resulting solution.

Public Functions

Pressure (const *Renderer::Device* &device, float *dt*, const glm::ivec2 &size, *LinearSolver::Data* &data, *Velocity* &velocity, *Renderer::Texture* &solidPhi, *Renderer::Texture* &liquidPhi, *Renderer::GenericBuffer* &valid)

Renderer::Work::Bound **BindMatrixBuild** (const glm::ivec2 &size, *Renderer::GenericBuffer* &diagonal, *Renderer::GenericBuffer* &lower, *Renderer::Texture* &liquidPhi, *Renderer::Texture* &solidPhi)

Bind the various buffes for the linear system $Ax = b$.

Return

Parameters

- size: size of the linear system
- diagonal: diagonal of A
- lower: lower matrix of A
- liquidPhi: liquid level set
- solidPhi: solid level set

void **BuildLinearEquation** ()

Build the matrix A and right hand side b.

void **ApplyPressure** ()

Apply the solution of the equation $Ax = b$, i.e. the pressure to the velocity to make it non-divergent.

class Rectangle : public Vortex2D::Fluid::Polygon

#include <Boundaries.h> Signed distance field of a rectangle.

Public Functions

Rectangle (const *Renderer::Device* &device, const glm::vec2 &size, bool *inverse* = false, float *extent* = 10.0f)

Initialize rectangle with size and extend of signed distance.

Parameters

- *device*: vulkan device.
- *size*: rectangle size
- *inverse*: flag if the distance field should be inverted.
- *extent*: extent how far from the rectangle the signed distance field is calculated.

class Reduce

#include <Reduce.h> Parallel reduction of a buffer into one value. The operator and type of data is specified by inheriting the class.

Subclassed by *Vortex2D::Fluid::ReduceJ*, *Vortex2D::Fluid::ReduceMax*, *Vortex2D::Fluid::ReduceSum*

Public Functions

virtual ~Reduce ()

Reduce::Bound **Bind** (*Renderer::GenericBuffer* &input, *Renderer::GenericBuffer* &output)

Bind the reduce operation.

Return a bound object that can be recorded in a command buffer.

Parameters

- *input*: input buffer
- *output*: output buffer

class Bound

#include <Reduce.h> *Bound* input and output buffer for a reduce operation.

Public Functions

Bound ()

void **Record** (vk::CommandBuffer *commandBuffer*)

Record the reduce operation.

Parameters

- *commandBuffer*: the command buffer to record into.

Friends

friend Vortex2D::Fluid::Reduce

```
class ReduceJ : public Vortex2D::Fluid::Reduce
    #include <Reduce.h> Reduce operation on a struct with a 2d vector and 1 float (i.e. 3 floats) with addition.
```

Public Functions

```
ReduceJ (const Renderer::Device &device, const glm::ivec2 &size)
    Initialize reduce with device and 2d size.
```

Parameters

- device:
- size:

```
class ReduceMax : public Vortex2D::Fluid::Reduce
    #include <Reduce.h> Reduce operation on float with max of absolute.
```

Public Functions

```
ReduceMax (const Renderer::Device &device, const glm::ivec2 &size)
    Initialize reduce with device and 2d size.
```

Parameters

- device:
- size:

```
class ReduceSum : public Vortex2D::Fluid::Reduce
    #include <Reduce.h> Reduce operation on float with addition.
```

Public Functions

```
ReduceSum (const Renderer::Device &device, const glm::ivec2 &size)
    Initialize reduce with device and 2d size.
```

Parameters

- device:
- size:

```
class RigidBody : public Vortex2D::Renderer::Transformable
    #include <Rigidbody.h> Rigidbody that can interact with the fluid: either be push by it, or influence it, or both.
```

Public Types

```
enum Type
```

Values:

```
eStatic = 0x01
```

```
eWeak = 0x02
```

```
eStrong = 0x3
```

Public Functions

RigidBody (*const* *Renderer::Device* &device, *const* *Dimensions* &dimensions, *Renderer::Drawable* &drawable, *const* glm::vec2 ¢re, *Renderer::RenderTexture* &phi, vk::Flags<*Type*> type)

void **SetVelocities** (*const* glm::vec2 &velocity, float angularVelocity)
sets the velocities and angular velocities of the body

Parameters

- velocity:
- angularVelocity:

void **UpdatePosition** ()
Upload the transform matrix to the GPU.

void **RenderPhi** ()
Render the current object orientation in an internal texture and the external one.

void **BindDiv** (*Renderer::GenericBuffer* &div, *Renderer::GenericBuffer* &diagonal, *Renderer::Texture* &fluidLevelSet)
Bind a the right hand side and diagonal of the linear system $Ax = b$. This is to apply the rigid body influence to the system.

Parameters

- div: right hand side of the linear system $Ax=b$
- diagonal: diagonal of matrix A
- fluidLevelSet: fluid level set

void **BindVelocityConstrain** (*Fluid::Velocity* &velocity)
Bind velocities to constrain based on the body's velocity.

Parameters

- velocity:

void **BindPressure** (*Renderer::Texture* &fluidLevelSet, *Renderer::GenericBuffer* &pressure, *Renderer::GenericBuffer* &force)
Bind pressure, to have the pressure update the body's forces.

Parameters

- fluidLevelSet: fluid level set, to know if the pressure is applicable
- pressure: solved pressure buffer
- force: a scratch buffer where the forces will be set

void **Div** ()
Apply the body's velocities to the linear equations matrix A and right hand side b.

void **Pressure** ()
Apply the pressure to body, updating its forces.

void **VelocityConstrain** ()
Constrain the velocities field based on the body's velocity.

RigidBody::Velocity **GetForces** ()
Download the forces from the GPU and return them.

Return

vk::Flags<*RigidBody::Type*> **GetType** ()
Type of this body.

Return

Renderer::RenderTexture &**Phi** ()
the local level set of the body

Return

struct Velocity

Public Members

glm::vec2 **velocity**

float **angular_velocity**

class SmokeWorld: public Vortex2D::Fluid::World

#include <World.h> A concrete implementation of *World* to simulate ‘smoke’, or more accurately dye in a liquid. The liquid cannot change location or size.

Public Functions

SmokeWorld (const *Renderer::Device* &device, *Dimensions* dimensions, float dt)

void **Solve** ()
Perform one step of the simulation.

void **FieldBind** (*Density* &density)
Bind a density field to be moved around with the fluid.

Parameters

- density: the density field

class Transfer

#include <Transfer.h> Prolongates or restrict a level set on a finer or coarser level set.

Public Functions

Transfer (const *Renderer::Device* &device)
Initialize prolongate and restrict compute pipelines.

Parameters

- device:

void **ProlongateBind** (std::size_t level, const glm::ivec2 &fineSize, *Renderer::GenericBuffer* &fine, *Renderer::GenericBuffer* &fineDiagonal, *Renderer::GenericBuffer* &coarse, *Renderer::GenericBuffer* &coarseDiagonal)

Prolongate a level set on a finer level set. Setting the 4 cells to the value of the coarser grid. Multiple level sets can be bound and indexed.

Parameters

- level: the index of the bound level set to prolongate
- fineSize: size of the finer level set
- fine: the finer level set
- fineDiagonal: the diagonal of the linear equation matrix at size fineSize
- coarse: the coarse level set

- `coarseDiagonal`: the diagonal of the linear equation matrix at size half of `fineSize`

void **RestrictBind** (std::size_t *level*, const glm::ivec2 &*fineSize*, *Renderer::GenericBuffer* &*fine*, *Renderer::GenericBuffer* &*fineDiagonal*, *Renderer::GenericBuffer* &*coarse*, *Renderer::GenericBuffer* &*coarseDiagonal*)

Restricting the level set on a coarser level set. Averages 4 cells into one. Multiple level sets can be bound and indexed.

Parameters

- `level`: the index of the bound level set to prolongate
- `fineSize`: size of the finer level set
- `fine`: the finer level set
- `fineDiagonal`: the diagonal of the linear equation matrix at size `fineSize`
- `coarse`: the coarse level set
- `coarseDiagonal`: the diagonal of the linear equation matrix at size half of `fineSize`

void **Prolongate** (vk::CommandBuffer *commandBuffer*, std::size_t *level*)

Prolongate the level set, using the bound level sets at the specified index.

Parameters

- `commandBuffer`: command buffer to record into.
- `level`: index of bound level sets.

void **Restrict** (vk::CommandBuffer *commandBuffer*, std::size_t *level*)

Restrict the level set, using the bound level sets at the specified index.

Parameters

- `commandBuffer`: command buffer to record into.
- `level`: index of bound level sets.

class Velocity : public Vortex2D::Renderer::RenderTexture

#include <Velocity.h> The *Velocity* field. Can be used to calculate a difference between different states. Contains three fields: input and output, used for ping-pong algorithms, and d, the difference between two velocity fields.

Public Functions

Velocity (const *Renderer::Device* &*device*, const glm::ivec2 &*size*)

Renderer::Texture &**Output** ()

An output texture used for algorithms that used the velocity as input and need to create a new velocity field.

Return

Renderer::Texture &**D** ()

A difference velocity field, calculated with the difference between this velocity field, and the output velocity field.

Return

void **CopyBack** (vk::CommandBuffer *commandBuffer*)

Copy the output field to the main field.

Parameters

- `commandBuffer`:

void **Clear** (vk::CommandBuffer *commandBuffer*)
Clear the velocity field.

Parameters

- *commandBuffer*:

void **SaveCopy** ()
Copy to the difference field.

void **VelocityDiff** ()
Calculate the difference between the difference field and this velocity field, store it in the difference field.

class WaterWorld: public Vortex2D::Fluid::World
#include <World.h> A concrete implementation of *World* to simulate water.

Public Functions

WaterWorld (const *Renderer::Device* &device, *Dimensions* dimensions, float dt)

void **Solve** ()
Perform one step of the simulation.

Renderer::RenderCommand **RecordParticleCount** (*Renderer::RenderTarget::DrawableList* drawables)

The water simulation uses particles to define the water area. In fact, the level set is built from the particles. This means to be able to set an area, we can't use *RecordLiquidPhi*. To define the particle area, simply draw a regular shape. The colour *r* is used to determine if we add or remove particles, use *r* = 4 to add and *r* = -4 to remove.

Return render command

Parameters

- *drawables*: list of drawables object with colour 4 or -4

class World
#include <World.h> The main class of the framework. Each instance manages a grid and this class is used to set forces, define boundaries, solve the incompressibility equations and do the advection.

Subclassed by *Vortex2D::Fluid::SmokeWorld*, *Vortex2D::Fluid::WaterWorld*

Public Functions

World (const *Renderer::Device* &device, *Dimensions* dimensions, float dt)
Construct an Engine with a size and time step.

Parameters

- *device*: vulkan device
- *dimensions*: dimensions of the simulation
- *dt*: timestamp of the simulation, e.g. 0.016 for 60FPS simulations.

virtual ~World ()

virtual void Solve () = 0
Perform one step of the simulation.

Renderer::RenderCommand **RecordVelocity** (*Renderer::RenderTarget::DrawableList* drawables)

Record drawables to the velocity field. The colour (r,g) will be used as the velocity (x, y)

Return render command

Parameters

- drawables: a list of drawable field

void **SubmitVelocity** (*Renderer::RenderCommand* &renderCommand)

submit the render command created with *RecordVelocity*

Parameters

- renderCommand: the render command

Renderer::RenderCommand **RecordLiquidPhi** (*Renderer::RenderTarget::DrawableList* drawables)

Record drawables to the liquid level set, i.e. to define the fluid area. The drawables need to make a signed distance field, if not the result is undefined.

Return render command

Parameters

- drawables: a list of signed distance field drawables

Renderer::RenderCommand **RecordStaticSolidPhi** (*Renderer::RenderTarget::DrawableList* drawables)

Record drawables to the solid level set, i.e. to define the boundary area. The drawables need to make a signed distance field, if not the result is undefined.

Return render command

Parameters

- drawables: a list of signed distance field drawables

DistanceField **LiquidDistanceField** ()

Create sprite that can be rendered to visualize the liquid level set.

Return a sprite

DistanceField **SolidDistanceField** ()

Create sprite that can be rendered to visualize the solid level set.

Return a sprite

RigidBody ***CreateRigidbody** (vk::Flags<*RigidBody::Type*> type, *Renderer::Drawable* &drawable, const glm::vec2 ¢re)

Create a rigid body and add it to the *World*. The lifetime of the rigidbody is tied to the lifetime of the *World*.

Return a pointer to the rigid body.

Parameters

- type: type of the rigidbody: static, weak or strong
- drawable: a drawable that renders a signed distance field. This can be the sprite of a level set.
- centre: the centre of the drawable. Use for rotations.

C

CopyFrom (C++ function), 15

CopyTo (C++ function), 15

V

Vortex2D::Fluid (C++ type), 27

Vortex2D::Fluid::Advection (C++ class), 27

Vortex2D::Fluid::Advection::Advect (C++ function), 28

Vortex2D::Fluid::Advection::AdvectBind (C++ function), 28

Vortex2D::Fluid::Advection::Advection (C++ function), 28

Vortex2D::Fluid::Advection::AdvectParticleBind (C++ function), 28

Vortex2D::Fluid::Advection::AdvectParticles (C++ function), 28

Vortex2D::Fluid::Advection::AdvectVelocity (C++ function), 28

Vortex2D::Fluid::Circle (C++ class), 28

Vortex2D::Fluid::Circle::Circle (C++ function), 28

Vortex2D::Fluid::Circle::Draw (C++ function), 29

Vortex2D::Fluid::Circle::Initialize (C++ function), 28

Vortex2D::Fluid::Circle::Update (C++ function), 28

Vortex2D::Fluid::ConjugateGradient (C++ class), 29

Vortex2D::Fluid::ConjugateGradient::Bind (C++ function), 29

Vortex2D::Fluid::ConjugateGradient::ConjugateGradient (C++ function), 29

Vortex2D::Fluid::ConjugateGradient::Solve (C++ function), 29

Vortex2D::Fluid::Density (C++ class), 29

Vortex2D::Fluid::Density::Density (C++ function), 29

Vortex2D::Fluid::Depth (C++ class), 29

Vortex2D::Fluid::Depth::Depth (C++ function), 30

Vortex2D::Fluid::Depth::GetDepthSize (C++ function), 30

Vortex2D::Fluid::Depth::GetMaxDepth (C++ function), 30

Vortex2D::Fluid::Diagonal (C++ class), 30

Vortex2D::Fluid::Diagonal::Bind (C++ function), 30

Vortex2D::Fluid::Diagonal::Diagonal (C++ function), 30

Vortex2D::Fluid::Diagonal::Record (C++ function), 30

Vortex2D::Fluid::Dimensions (C++ class), 30

Vortex2D::Fluid::Dimensions::Dimensions (C++ function), 30

Vortex2D::Fluid::Dimensions::InvScale (C++ member), 31

Vortex2D::Fluid::Dimensions::Scale (C++ member), 31

Vortex2D::Fluid::Dimensions::Size (C++ member), 31

Vortex2D::Fluid::DistanceField (C++ class), 31

Vortex2D::Fluid::DistanceField::DistanceField (C++ function), 31

Vortex2D::Fluid::DistanceField::Draw (C++ function), 31

Vortex2D::Fluid::eStatic (C++ enumerator), 41

Vortex2D::Fluid::eStrong (C++ enumerator), 41

Vortex2D::Fluid::eWeak (C++ enumerator), 41

Vortex2D::Fluid::Extrapolation (C++ class), 31

Vortex2D::Fluid::Extrapolation::ConstrainBind (C++ function), 31

Vortex2D::Fluid::Extrapolation::ConstrainVelocity (C++ function), 31

Vortex2D::Fluid::Extrapolation::Extrapolate (C++ function), 31

Vortex2D::Fluid::Extrapolation::Extrapolation (C++ function), 31

Vortex2D::Fluid::GaussSeidel (C++ class), 31

Vortex2D::Fluid::GaussSeidel::Bind (C++ function), 32

Vortex2D::Fluid::GaussSeidel::GaussSeidel (C++ function), 32

Vortex2D::Fluid::GaussSeidel::Record (C++ function), 32

Vortex2D::Fluid::GaussSeidel::SetPreconditionerIterations (C++ function), 32

Vortex2D::Fluid::GaussSeidel::SetW (C++ function), 32

Vortex2D::Fluid::GaussSeidel::Solve (C++ function), 32

Vortex2D::Fluid::IncompletePoisson (C++ class), 32

Vortex2D::Fluid::IncompletePoisson::Bind (C++ function), 32

Vortex2D::Fluid::IncompletePoisson::IncompletePoisson (C++ function), 32
Vortex2D::Fluid::IncompletePoisson::Record (C++ function), 33
Vortex2D::Fluid::J (C++ class), 33
Vortex2D::Fluid::J::angular (C++ member), 33
Vortex2D::Fluid::J::linear (C++ member), 33
Vortex2D::Fluid::Jacobi (C++ class), 33
Vortex2D::Fluid::Jacobi::Bind (C++ function), 33
Vortex2D::Fluid::Jacobi::Jacobi (C++ function), 33
Vortex2D::Fluid::Jacobi::Record (C++ function), 33
Vortex2D::Fluid::Jacobi::SetPreconditionerIterations (C++ function), 33
Vortex2D::Fluid::Jacobi::SetW (C++ function), 33
Vortex2D::Fluid::LevelSet (C++ class), 33
Vortex2D::Fluid::LevelSet::Extrapolate (C++ function), 34
Vortex2D::Fluid::LevelSet::ExtrapolateBind (C++ function), 34
Vortex2D::Fluid::LevelSet::ExtrapolateRecord (C++ function), 34
Vortex2D::Fluid::LevelSet::LevelSet (C++ function), 34
Vortex2D::Fluid::LevelSet::Reinitialise (C++ function), 34
Vortex2D::Fluid::LinearSolver (C++ class), 34
Vortex2D::Fluid::LinearSolver::~LinearSolver (C++ function), 34
Vortex2D::Fluid::LinearSolver::Bind (C++ function), 34
Vortex2D::Fluid::LinearSolver::Data (C++ class), 34
Vortex2D::Fluid::LinearSolver::Data::B (C++ member), 35
Vortex2D::Fluid::LinearSolver::Data::Data (C++ function), 34
Vortex2D::Fluid::LinearSolver::Data::Diagonal (C++ member), 35
Vortex2D::Fluid::LinearSolver::Data::Lower (C++ member), 35
Vortex2D::Fluid::LinearSolver::Data::X (C++ member), 35
Vortex2D::Fluid::LinearSolver::Parameters (C++ class), 35
Vortex2D::Fluid::LinearSolver::Parameters::ErrorTolerance (C++ member), 35
Vortex2D::Fluid::LinearSolver::Parameters::IsFinished (C++ function), 35
Vortex2D::Fluid::LinearSolver::Parameters::Iterations (C++ member), 35
Vortex2D::Fluid::LinearSolver::Parameters::OutError (C++ member), 35
Vortex2D::Fluid::LinearSolver::Parameters::OutIterations (C++ member), 35
Vortex2D::Fluid::LinearSolver::Parameters::Parameters (C++ function), 35
Vortex2D::Fluid::LinearSolver::Solve (C++ function), 34
Vortex2D::Fluid::LocalGaussSeidel (C++ class), 35
Vortex2D::Fluid::LocalGaussSeidel::Bind (C++ function), 35
Vortex2D::Fluid::LocalGaussSeidel::LocalGaussSeidel (C++ function), 35
Vortex2D::Fluid::LocalGaussSeidel::Record (C++ function), 36
Vortex2D::Fluid::MakeLocalSize (C++ function), 27
Vortex2D::Fluid::Multigrid (C++ class), 36
Vortex2D::Fluid::Multigrid::Bind (C++ function), 36
Vortex2D::Fluid::Multigrid::BuildHierarchies (C++ function), 36
Vortex2D::Fluid::Multigrid::BuildHierarchiesBind (C++ function), 36
Vortex2D::Fluid::Multigrid::Multigrid (C++ function), 36
Vortex2D::Fluid::Multigrid::Record (C++ function), 36
Vortex2D::Fluid::Particle (C++ class), 36
Vortex2D::Fluid::Particle::Position (C++ member), 36
Vortex2D::Fluid::Particle::Velocity (C++ member), 36
Vortex2D::Fluid::ParticleCount (C++ class), 37
Vortex2D::Fluid::ParticleCount::GetDispatchParams (C++ function), 37
Vortex2D::Fluid::ParticleCount::GetTotalCount (C++ function), 37
Vortex2D::Fluid::ParticleCount::LevelSetBind (C++ function), 37
Vortex2D::Fluid::ParticleCount::ParticleCount (C++ function), 37
Vortex2D::Fluid::ParticleCount::Phi (C++ function), 37
Vortex2D::Fluid::ParticleCount::Scan (C++ function), 37
Vortex2D::Fluid::ParticleCount::TransferFromGrid (C++ function), 37
Vortex2D::Fluid::ParticleCount::TransferToGrid (C++ function), 37
Vortex2D::Fluid::ParticleCount::VelocitiesBind (C++ function), 37
Vortex2D::Fluid::Polygon (C++ class), 37
Vortex2D::Fluid::Polygon::Draw (C++ function), 38
Vortex2D::Fluid::Polygon::Initialize (C++ function), 38
Vortex2D::Fluid::Polygon::Polygon (C++ function), 37
Vortex2D::Fluid::Polygon::Update (C++ function), 38
Vortex2D::Fluid::Preconditioner (C++ class), 38
Vortex2D::Fluid::Preconditioner::~Preconditioner (C++ function), 38
Vortex2D::Fluid::Preconditioner::Bind (C++ function), 38
Vortex2D::Fluid::Preconditioner::Record (C++ function), 38
Vortex2D::Fluid::PrefixScan (C++ class), 38
Vortex2D::Fluid::PrefixScan::Bind (C++ function), 39
Vortex2D::Fluid::PrefixScan::Bound (C++ class), 39
Vortex2D::Fluid::PrefixScan::Bound::Record (C++ function), 39
Vortex2D::Fluid::PrefixScan::PrefixScan (C++ function),

- 39
- Vortex2D::Fluid::Pressure (C++ class), 39
- Vortex2D::Fluid::Pressure::ApplyPressure (C++ function), 39
- Vortex2D::Fluid::Pressure::BindMatrixBuild (C++ function), 39
- Vortex2D::Fluid::Pressure::BuildLinearEquation (C++ function), 39
- Vortex2D::Fluid::Pressure::Pressure (C++ function), 39
- Vortex2D::Fluid::Rectangle (C++ class), 40
- Vortex2D::Fluid::Rectangle::Rectangle (C++ function), 40
- Vortex2D::Fluid::Reduce (C++ class), 40
- Vortex2D::Fluid::Reduce::~~Reduce (C++ function), 40
- Vortex2D::Fluid::Reduce::Bind (C++ function), 40
- Vortex2D::Fluid::Reduce::Bound (C++ class), 40
- Vortex2D::Fluid::Reduce::Bound::Bound (C++ function), 40
- Vortex2D::Fluid::Reduce::Bound::Record (C++ function), 40
- Vortex2D::Fluid::ReduceJ (C++ class), 40
- Vortex2D::Fluid::ReduceJ::ReduceJ (C++ function), 41
- Vortex2D::Fluid::ReduceMax (C++ class), 41
- Vortex2D::Fluid::ReduceMax::ReduceMax (C++ function), 41
- Vortex2D::Fluid::ReduceSum (C++ class), 41
- Vortex2D::Fluid::ReduceSum::ReduceSum (C++ function), 41
- Vortex2D::Fluid::RigidBody (C++ class), 41
- Vortex2D::Fluid::RigidBody::BindDiv (C++ function), 42
- Vortex2D::Fluid::RigidBody::BindPressure (C++ function), 42
- Vortex2D::Fluid::RigidBody::BindVelocityConstrain (C++ function), 42
- Vortex2D::Fluid::RigidBody::Div (C++ function), 42
- Vortex2D::Fluid::RigidBody::GetForces (C++ function), 42
- Vortex2D::Fluid::RigidBody::GetType (C++ function), 42
- Vortex2D::Fluid::RigidBody::Phi (C++ function), 43
- Vortex2D::Fluid::RigidBody::Pressure (C++ function), 42
- Vortex2D::Fluid::RigidBody::RenderPhi (C++ function), 42
- Vortex2D::Fluid::RigidBody::RigidBody (C++ function), 42
- Vortex2D::Fluid::RigidBody::SetVelocities (C++ function), 42
- Vortex2D::Fluid::RigidBody::UpdatePosition (C++ function), 42
- Vortex2D::Fluid::RigidBody::Velocity (C++ class), 43
- Vortex2D::Fluid::RigidBody::Velocity::angular_velocity (C++ member), 43
- Vortex2D::Fluid::RigidBody::Velocity::velocity (C++ member), 43
- Vortex2D::Fluid::RigidBody::VelocityConstrain (C++ function), 42
- Vortex2D::Fluid::SmokeWorld (C++ class), 43
- Vortex2D::Fluid::SmokeWorld::FieldBind (C++ function), 43
- Vortex2D::Fluid::SmokeWorld::SmokeWorld (C++ function), 43
- Vortex2D::Fluid::SmokeWorld::Solve (C++ function), 43
- Vortex2D::Fluid::Transfer (C++ class), 43
- Vortex2D::Fluid::Transfer::Prolongate (C++ function), 44
- Vortex2D::Fluid::Transfer::ProlongateBind (C++ function), 43
- Vortex2D::Fluid::Transfer::Restrict (C++ function), 44
- Vortex2D::Fluid::Transfer::RestrictBind (C++ function), 44
- Vortex2D::Fluid::Transfer::Transfer (C++ function), 43
- Vortex2D::Fluid::Type (C++ type), 41
- Vortex2D::Fluid::Velocity (C++ class), 44
- Vortex2D::Fluid::Velocity::Clear (C++ function), 44
- Vortex2D::Fluid::Velocity::CopyBack (C++ function), 44
- Vortex2D::Fluid::Velocity::D (C++ function), 44
- Vortex2D::Fluid::Velocity::Output (C++ function), 44
- Vortex2D::Fluid::Velocity::SaveCopy (C++ function), 45
- Vortex2D::Fluid::Velocity::Velocity (C++ function), 44
- Vortex2D::Fluid::Velocity::VelocityDiff (C++ function), 45
- Vortex2D::Fluid::WaterWorld (C++ class), 45
- Vortex2D::Fluid::WaterWorld::RecordParticleCount (C++ function), 45
- Vortex2D::Fluid::WaterWorld::Solve (C++ function), 45
- Vortex2D::Fluid::WaterWorld::WaterWorld (C++ function), 45
- Vortex2D::Fluid::World (C++ class), 45
- Vortex2D::Fluid::World::~~World (C++ function), 45
- Vortex2D::Fluid::World::CreateRigidbody (C++ function), 46
- Vortex2D::Fluid::World::LiquidDistanceField (C++ function), 46
- Vortex2D::Fluid::World::RecordLiquidPhi (C++ function), 46
- Vortex2D::Fluid::World::RecordStaticSolidPhi (C++ function), 46
- Vortex2D::Fluid::World::RecordVelocity (C++ function), 45
- Vortex2D::Fluid::World::SolidDistanceField (C++ function), 46
- Vortex2D::Fluid::World::Solve (C++ function), 45
- Vortex2D::Fluid::World::SubmitVelocity (C++ function), 46
- Vortex2D::Fluid::World::World (C++ function), 45
- Vortex2D::Renderer (C++ type), 5
- Vortex2D::Renderer::AbstractShape (C++ class), 6

Vortex2D::Renderer::AbstractShape::~~AbstractShape
(C++ function), 7
Vortex2D::Renderer::AbstractShape::AbstractShape
(C++ function), 7
Vortex2D::Renderer::AbstractShape::Colour (C++ mem-
ber), 7
Vortex2D::Renderer::AbstractShape::Draw (C++ func-
tion), 7
Vortex2D::Renderer::AbstractShape::Initialize (C++
function), 7
Vortex2D::Renderer::AbstractShape::Update (C++ func-
tion), 7
Vortex2D::Renderer::AbstractSprite (C++ class), 7
Vortex2D::Renderer::AbstractSprite::~~AbstractSprite
(C++ function), 7
Vortex2D::Renderer::AbstractSprite::AbstractSprite
(C++ function), 7
Vortex2D::Renderer::AbstractSprite::Colour (C++ mem-
ber), 8
Vortex2D::Renderer::AbstractSprite::Draw (C++ func-
tion), 8
Vortex2D::Renderer::AbstractSprite::Initialize (C++
function), 7
Vortex2D::Renderer::AbstractSprite::PushConstant (C++
function), 8
Vortex2D::Renderer::AbstractSprite::Update (C++ func-
tion), 7
Vortex2D::Renderer::BindingInput (C++ class), 8
Vortex2D::Renderer::BindingInput::Bind (C++ member),
8
Vortex2D::Renderer::BindingInput::BindingInput (C++
function), 8
Vortex2D::Renderer::BindingInput::DefaultBind (C++
member), 8
Vortex2D::Renderer::BindingInput::Input (C++ mem-
ber), 8
Vortex2D::Renderer::Buffer (C++ class), 8
Vortex2D::Renderer::Buffer::Buffer (C++ function), 8
Vortex2D::Renderer::BufferBarrier (C++ function), 5
Vortex2D::Renderer::Clear (C++ class), 8
Vortex2D::Renderer::Clear::Clear (C++ function), 9
Vortex2D::Renderer::Clear::Draw (C++ function), 9
Vortex2D::Renderer::Clear::Initialize (C++ function), 9
Vortex2D::Renderer::Clear::Update (C++ function), 9
Vortex2D::Renderer::CommandBuffer (C++ class), 9
Vortex2D::Renderer::CommandBuffer::~~CommandBuffer
(C++ function), 9
Vortex2D::Renderer::CommandBuffer::CommandBuffer
(C++ function), 9
Vortex2D::Renderer::CommandBuffer::CommandFn
(C++ type), 9
Vortex2D::Renderer::CommandBuffer::operator bool
(C++ function), 10
Vortex2D::Renderer::CommandBuffer::operator= (C++
function), 9
Vortex2D::Renderer::CommandBuffer::Record (C++
function), 9
Vortex2D::Renderer::CommandBuffer::Reset (C++ func-
tion), 10
Vortex2D::Renderer::CommandBuffer::Submit (C++
function), 10
Vortex2D::Renderer::CommandBuffer::Wait (C++ func-
tion), 10
Vortex2D::Renderer::ComputeSize (C++ class), 10
Vortex2D::Renderer::ComputeSize::ComputeSize (C++
function), 10
Vortex2D::Renderer::ComputeSize::Default1D (C++
function), 11
Vortex2D::Renderer::ComputeSize::Default2D (C++
function), 11
Vortex2D::Renderer::ComputeSize::DomainSize (C++
member), 10
Vortex2D::Renderer::ComputeSize::GetLocalSize1D
(C++ function), 10
Vortex2D::Renderer::ComputeSize::GetLocalSize2D
(C++ function), 10
Vortex2D::Renderer::ComputeSize::GetWorkSize (C++
function), 11
Vortex2D::Renderer::ComputeSize::LocalSize (C++
member), 10
Vortex2D::Renderer::ComputeSize::WorkSize (C++
member), 10
Vortex2D::Renderer::CopyFrom (C++ function), 5
Vortex2D::Renderer::CopyTo (C++ function), 5
Vortex2D::Renderer::DescriptorImage (C++ class), 11
Vortex2D::Renderer::DescriptorImage::DescriptorImage
(C++ function), 11
Vortex2D::Renderer::DescriptorImage::Sampler (C++
member), 11
Vortex2D::Renderer::DescriptorImage::Texture (C++
member), 11
Vortex2D::Renderer::DescriptorSet (C++ class), 11
Vortex2D::Renderer::DescriptorSet::descriptorSet (C++
member), 11
Vortex2D::Renderer::DescriptorSet::descriptorSetLayout
(C++ member), 11
Vortex2D::Renderer::DescriptorSet::pipelineLayout
(C++ member), 11
Vortex2D::Renderer::Device (C++ class), 11
Vortex2D::Renderer::Device::~~Device (C++ function), 12
Vortex2D::Renderer::Device::Allocator (C++ function),
12
Vortex2D::Renderer::Device::CreateCommandBuffers
(C++ function), 12
Vortex2D::Renderer::Device::Device (C++ function), 12
Vortex2D::Renderer::Device::FreeCommandBuffers
(C++ function), 12
Vortex2D::Renderer::Device::GetFamilyIndex (C++

- function), 12
- Vortex2D::Renderer::Device::GetLayoutManager (C++ function), 12
- Vortex2D::Renderer::Device::GetPhysicalDevice (C++ function), 12
- Vortex2D::Renderer::Device::GetShaderModule (C++ function), 12
- Vortex2D::Renderer::Device::Handle (C++ function), 12
- Vortex2D::Renderer::Device::operator= (C++ function), 12
- Vortex2D::Renderer::Device::Queue (C++ function), 12
- Vortex2D::Renderer::DispatchParams (C++ class), 12
- Vortex2D::Renderer::DispatchParams::count (C++ member), 12
- Vortex2D::Renderer::DispatchParams::DispatchParams (C++ function), 12
- Vortex2D::Renderer::DispatchParams::workSize (C++ member), 12
- Vortex2D::Renderer::Drawable (C++ class), 12
- Vortex2D::Renderer::Drawable::~~Drawable (C++ function), 13
- Vortex2D::Renderer::Drawable::Draw (C++ function), 13
- Vortex2D::Renderer::Drawable::Initialize (C++ function), 13
- Vortex2D::Renderer::Drawable::Update (C++ function), 13
- Vortex2D::Renderer::Ellipse (C++ class), 13
- Vortex2D::Renderer::Ellipse::Colour (C++ member), 14
- Vortex2D::Renderer::Ellipse::Draw (C++ function), 13
- Vortex2D::Renderer::Ellipse::Ellipse (C++ function), 13
- Vortex2D::Renderer::Ellipse::Initialize (C++ function), 13
- Vortex2D::Renderer::Ellipse::Update (C++ function), 13
- Vortex2D::Renderer::GenericBuffer (C++ class), 14
- Vortex2D::Renderer::GenericBuffer::~~GenericBuffer (C++ function), 14
- Vortex2D::Renderer::GenericBuffer::Barrier (C++ function), 14
- Vortex2D::Renderer::GenericBuffer::Clear (C++ function), 14
- Vortex2D::Renderer::GenericBuffer::CopyFrom (C++ function), 14
- Vortex2D::Renderer::GenericBuffer::GenericBuffer (C++ function), 14
- Vortex2D::Renderer::GenericBuffer::Handle (C++ function), 14
- Vortex2D::Renderer::GenericBuffer::Size (C++ function), 14
- Vortex2D::Renderer::GetDescriptorType (C++ function), 6
- Vortex2D::Renderer::GraphicsPipeline (C++ class), 15
- Vortex2D::Renderer::GraphicsPipeline::Bind (C++ function), 15
- Vortex2D::Renderer::GraphicsPipeline::Builder (C++ class), 15
- Vortex2D::Renderer::GraphicsPipeline::Builder::Builder (C++ function), 15
- Vortex2D::Renderer::GraphicsPipeline::Builder::Create (C++ function), 16
- Vortex2D::Renderer::GraphicsPipeline::Builder::Layout (C++ function), 16
- Vortex2D::Renderer::GraphicsPipeline::Builder::Shader (C++ function), 15
- Vortex2D::Renderer::GraphicsPipeline::Builder::Topology (C++ function), 16
- Vortex2D::Renderer::GraphicsPipeline::Builder::VertexAttribute (C++ function), 15
- Vortex2D::Renderer::GraphicsPipeline::Builder::VertexBinding (C++ function), 16
- Vortex2D::Renderer::GraphicsPipeline::Create (C++ function), 15
- Vortex2D::Renderer::GraphicsPipeline::GraphicsPipeline (C++ function), 15
- Vortex2D::Renderer::HasExtension (C++ function), 6
- Vortex2D::Renderer::HasLayer (C++ function), 6
- Vortex2D::Renderer::IndirectBuffer (C++ class), 16
- Vortex2D::Renderer::IndirectBuffer::IndirectBuffer (C++ function), 16
- Vortex2D::Renderer::Instance (C++ class), 16
- Vortex2D::Renderer::Instance::~~Instance (C++ function), 16
- Vortex2D::Renderer::Instance::GetInstance (C++ function), 16
- Vortex2D::Renderer::Instance::GetPhysicalDevice (C++ function), 16
- Vortex2D::Renderer::Instance::Instance (C++ function), 16
- Vortex2D::Renderer::IntRectangle (C++ class), 16
- Vortex2D::Renderer::IntRectangle::IntRectangle (C++ function), 16
- Vortex2D::Renderer::LayoutManager (C++ class), 16
- Vortex2D::Renderer::LayoutManager::CreateDescriptorPool (C++ function), 17
- Vortex2D::Renderer::LayoutManager::GetDescriptorSetLayout (C++ function), 17
- Vortex2D::Renderer::LayoutManager::GetPipelineLayout (C++ function), 17
- Vortex2D::Renderer::LayoutManager::LayoutManager (C++ function), 17
- Vortex2D::Renderer::LayoutManager::MakeDescriptorSet (C++ function), 17
- Vortex2D::Renderer::operator== (C++ function), 6
- Vortex2D::Renderer::Path (C++ type), 5
- Vortex2D::Renderer::PipelineLayout (C++ class), 17
- Vortex2D::Renderer::PipelineLayout::layouts (C++ member), 17
- Vortex2D::Renderer::property (C++ class), 17
- Vortex2D::Renderer::property::operator T const& (C++

- function), 17
- Vortex2D::Renderer::property::operator= (C++ function), 17
- Vortex2D::Renderer::property::property (C++ function), 17
- Vortex2D::Renderer::Rectangle (C++ class), 17
- Vortex2D::Renderer::Rectangle::Rectangle (C++ function), 18
- Vortex2D::Renderer::RenderCommand (C++ class), 18
- Vortex2D::Renderer::RenderCommand::~RenderCommand (C++ function), 18
- Vortex2D::Renderer::RenderCommand::operator= (C++ function), 18
- Vortex2D::Renderer::RenderCommand::RenderCommand (C++ function), 18
- Vortex2D::Renderer::RenderCommand::Submit (C++ function), 18
- Vortex2D::Renderer::RenderpassBuilder (C++ class), 18
- Vortex2D::Renderer::RenderpassBuilder::Attachement (C++ function), 18
- Vortex2D::Renderer::RenderpassBuilder::AttachementFinalLayout (C++ function), 19
- Vortex2D::Renderer::RenderpassBuilder::AttachementInitialLayout (C++ function), 19
- Vortex2D::Renderer::RenderpassBuilder::AttachementLoadOp (C++ function), 18
- Vortex2D::Renderer::RenderpassBuilder::AttachementStoreOp (C++ function), 18
- Vortex2D::Renderer::RenderpassBuilder::Create (C++ function), 19
- Vortex2D::Renderer::RenderpassBuilder::Dependency (C++ function), 19
- Vortex2D::Renderer::RenderpassBuilder::DependencyDstAccessMask (C++ function), 19
- Vortex2D::Renderer::RenderpassBuilder::DependencyDstStageMask (C++ function), 19
- Vortex2D::Renderer::RenderpassBuilder::DependencySrcAccessMask (C++ function), 19
- Vortex2D::Renderer::RenderpassBuilder::DependencySrcStageMask (C++ function), 19
- Vortex2D::Renderer::RenderpassBuilder::Subpass (C++ function), 19
- Vortex2D::Renderer::RenderpassBuilder::SubpassColorAttachment (C++ function), 19
- Vortex2D::Renderer::RenderState (C++ class), 19
- Vortex2D::Renderer::RenderState::ColorBlend (C++ member), 20
- Vortex2D::Renderer::RenderState::Height (C++ member), 20
- Vortex2D::Renderer::RenderState::RenderPass (C++ member), 20
- Vortex2D::Renderer::RenderState::RenderState (C++ function), 20
- Vortex2D::Renderer::RenderState::Width (C++ member), 20
- Vortex2D::Renderer::RenderTarget (C++ class), 20
- Vortex2D::Renderer::RenderTarget::~RenderTarget (C++ function), 20
- Vortex2D::Renderer::RenderTarget::DrawableList (C++ type), 20
- Vortex2D::Renderer::RenderTarget::Height (C++ member), 20
- Vortex2D::Renderer::RenderTarget::Orth (C++ member), 20
- Vortex2D::Renderer::RenderTarget::RenderPass (C++ member), 21
- Vortex2D::Renderer::RenderTarget::RenderTarget (C++ function), 20
- Vortex2D::Renderer::RenderTarget::View (C++ member), 20
- Vortex2D::Renderer::RenderTarget::Width (C++ member), 20
- Vortex2D::Renderer::RenderTexture (C++ class), 21
- Vortex2D::Renderer::RenderTexture::Record (C++ function), 21
- Vortex2D::Renderer::RenderTexture::RenderTexture (C++ function), 21
- Vortex2D::Renderer::RenderTexture::Submit (C++ function), 21
- Vortex2D::Renderer::RenderWindow (C++ class), 21
- Vortex2D::Renderer::RenderWindow::Display (C++ function), 21
- Vortex2D::Renderer::RenderWindow::Record (C++ function), 21
- Vortex2D::Renderer::RenderWindow::RenderWindow (C++ function), 21
- Vortex2D::Renderer::RenderWindow::Submit (C++ function), 21
- Vortex2D::Renderer::SamplerBuilder (C++ class), 21
- Vortex2D::Renderer::SamplerBuilder::AddressMode (C++ function), 21
- Vortex2D::Renderer::SamplerBuilder::Create (C++ function), 22
- Vortex2D::Renderer::SamplerBuilder::Filter (C++ function), 21
- Vortex2D::Renderer::SamplerBuilder::SamplerBuilder (C++ function), 21
- Vortex2D::Renderer::ShaderLayout (C++ class), 22
- Vortex2D::Renderer::ShaderLayout::bindings (C++ member), 22
- Vortex2D::Renderer::ShaderLayout::pushConstantSize (C++ member), 22
- Vortex2D::Renderer::ShaderLayout::ShaderLayout (C++ function), 22
- Vortex2D::Renderer::ShaderLayout::shaderStage (C++ member), 22
- Vortex2D::Renderer::SpirvBinary (C++ class), 22
- Vortex2D::Renderer::SpirvBinary::data (C++ function), 22

- 22
- Vortex2D::Renderer::SpirvBinary::size (C++ function), 22
- Vortex2D::Renderer::SpirvBinary::SpirvBinary (C++ function), 22
- Vortex2D::Renderer::SpirvBinary::words (C++ function), 22
- Vortex2D::Renderer::Sprite (C++ class), 22
- Vortex2D::Renderer::Sprite::Sprite (C++ function), 22
- Vortex2D::Renderer::SwapChainSupportDetails (C++ class), 22
- Vortex2D::Renderer::SwapChainSupportDetails::capabilities (C++ member), 23
- Vortex2D::Renderer::SwapChainSupportDetails::formats (C++ member), 23
- Vortex2D::Renderer::SwapChainSupportDetails::IsValid (C++ function), 23
- Vortex2D::Renderer::SwapChainSupportDetails::presentModes (C++ member), 23
- Vortex2D::Renderer::SwapChainSupportDetails::SwapChainSupportDetails (C++ function), 23
- Vortex2D::Renderer::Texture (C++ class), 23
- Vortex2D::Renderer::Texture::~Texture (C++ function), 23
- Vortex2D::Renderer::Texture::Barrier (C++ function), 23
- Vortex2D::Renderer::Texture::Clear (C++ function), 23
- Vortex2D::Renderer::Texture::CopyFrom (C++ function), 23
- Vortex2D::Renderer::Texture::CopyTo (C++ function), 23
- Vortex2D::Renderer::Texture::GetFormat (C++ function), 23
- Vortex2D::Renderer::Texture::GetHeight (C++ function), 23
- Vortex2D::Renderer::Texture::GetView (C++ function), 23
- Vortex2D::Renderer::Texture::GetWidth (C++ function), 23
- Vortex2D::Renderer::Texture::Texture (C++ function), 23
- Vortex2D::Renderer::Timer (C++ class), 23
- Vortex2D::Renderer::Timer::GetElapsedNs (C++ function), 24
- Vortex2D::Renderer::Timer::Start (C++ function), 24
- Vortex2D::Renderer::Timer::Stop (C++ function), 24
- Vortex2D::Renderer::Timer::Timer (C++ function), 24
- Vortex2D::Renderer::Transformable (C++ class), 24
- Vortex2D::Renderer::Transformable::~Transformable (C++ function), 24
- Vortex2D::Renderer::Transformable::Anchor (C++ member), 24
- Vortex2D::Renderer::Transformable::GetTransform (C++ function), 24
- Vortex2D::Renderer::Transformable::Position (C++ member), 24
- Vortex2D::Renderer::Transformable::Rotation (C++ member), 24
- Vortex2D::Renderer::Transformable::Scale (C++ member), 24
- Vortex2D::Renderer::Transformable::Transformable (C++ function), 24
- Vortex2D::Renderer::Transformable::Update (C++ function), 24
- Vortex2D::Renderer::UniformBuffer (C++ class), 24
- Vortex2D::Renderer::UniformBuffer::UniformBuffer (C++ function), 25
- Vortex2D::Renderer::VertexBuffer (C++ class), 25
- Vortex2D::Renderer::VertexBuffer::VertexBuffer (C++ function), 25
- Vortex2D::Renderer::Work (C++ class), 25
- Vortex2D::Renderer::Work::Bind (C++ function), 25
- Vortex2D::Renderer::Work::Bound (C++ class), 25
- Vortex2D::Renderer::Work::Bound::Bound (C++ function), 25
- Vortex2D::Renderer::Work::Bound::Dispatch (C++ function), 26
- Vortex2D::Renderer::Work::Bound::PushConstant (C++ function), 25
- Vortex2D::Renderer::Work::Bound::Record (C++ function), 26
- Vortex2D::Renderer::Work::Bound::RecordIndirect (C++ function), 26
- Vortex2D::Renderer::Work::Work (C++ function), 25