Voxel Ray Tracing Project Design

Ethan Tucker
Advisor: Dr. Hsu

The goal is to ray-trace each pixel

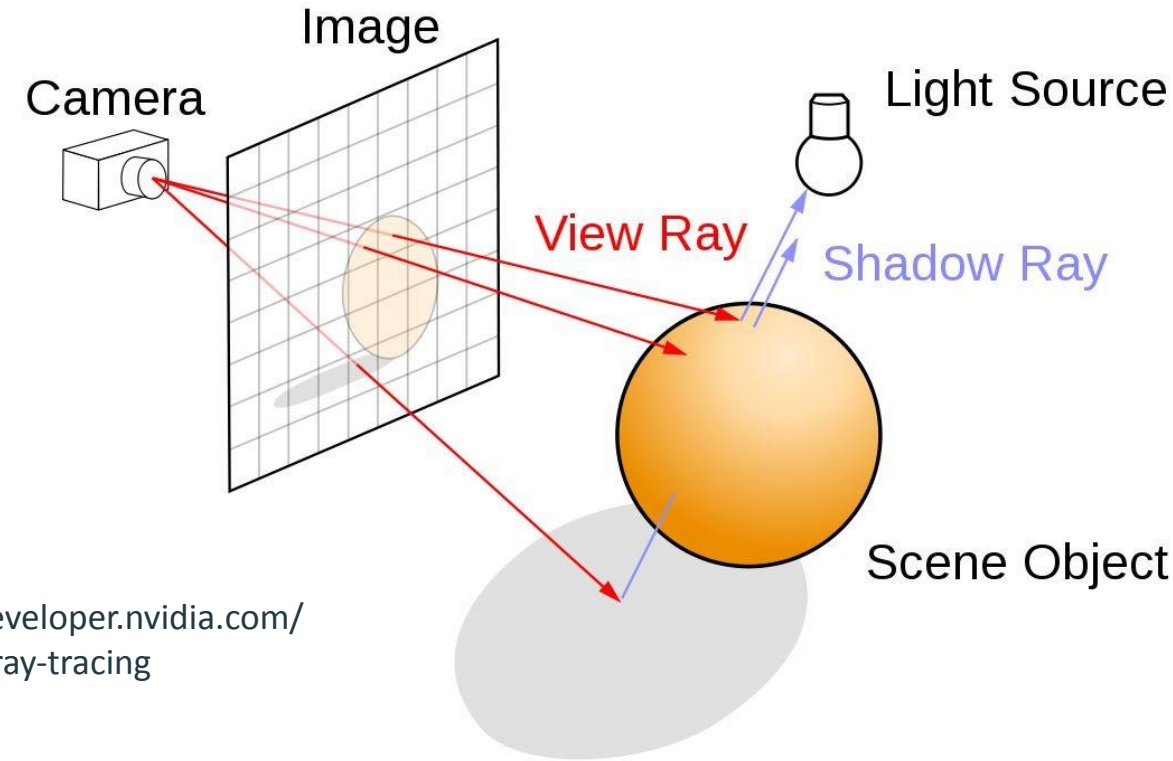


Image:
https://developer.nvidia.com/discover/ray-tracing

The DDA algorithm implements ray tracing for voxel grids

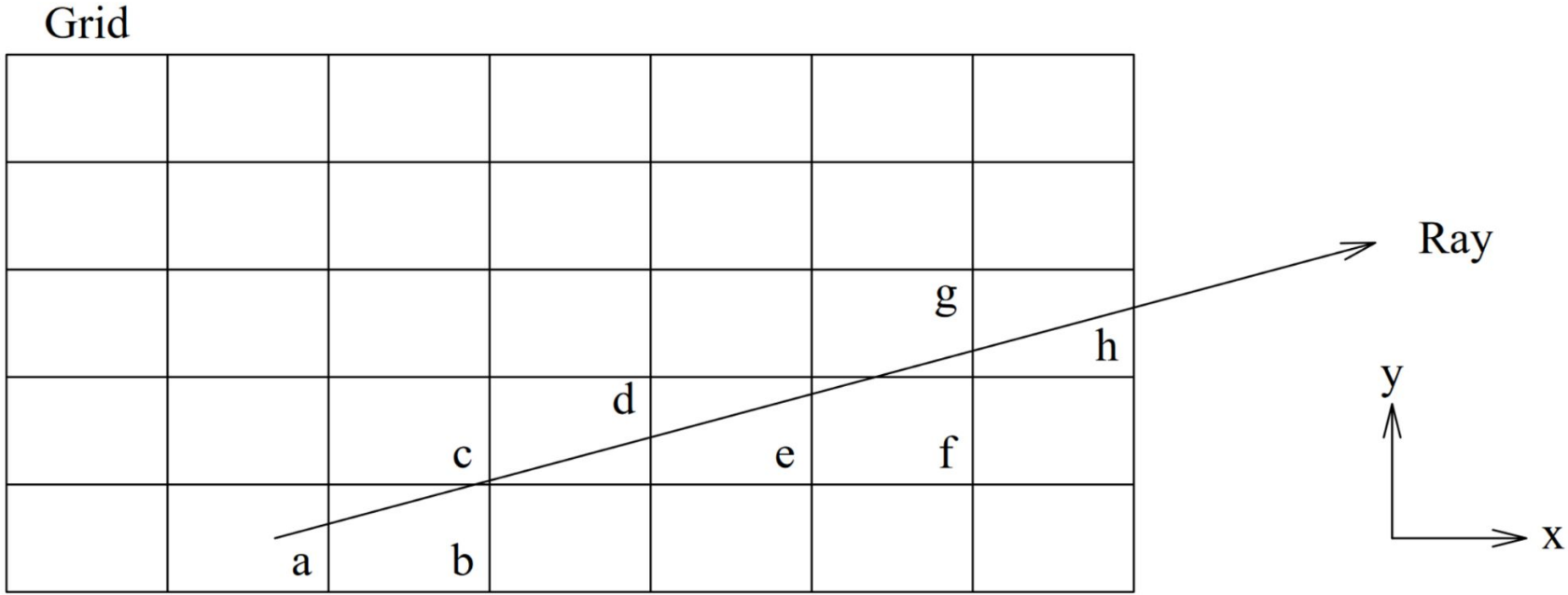


Image: Amantides & Woo “A Fast Voxel Traversal Algorithm for Ray Tracing”

The GPU driver implements Vulkan to manage executing shaders

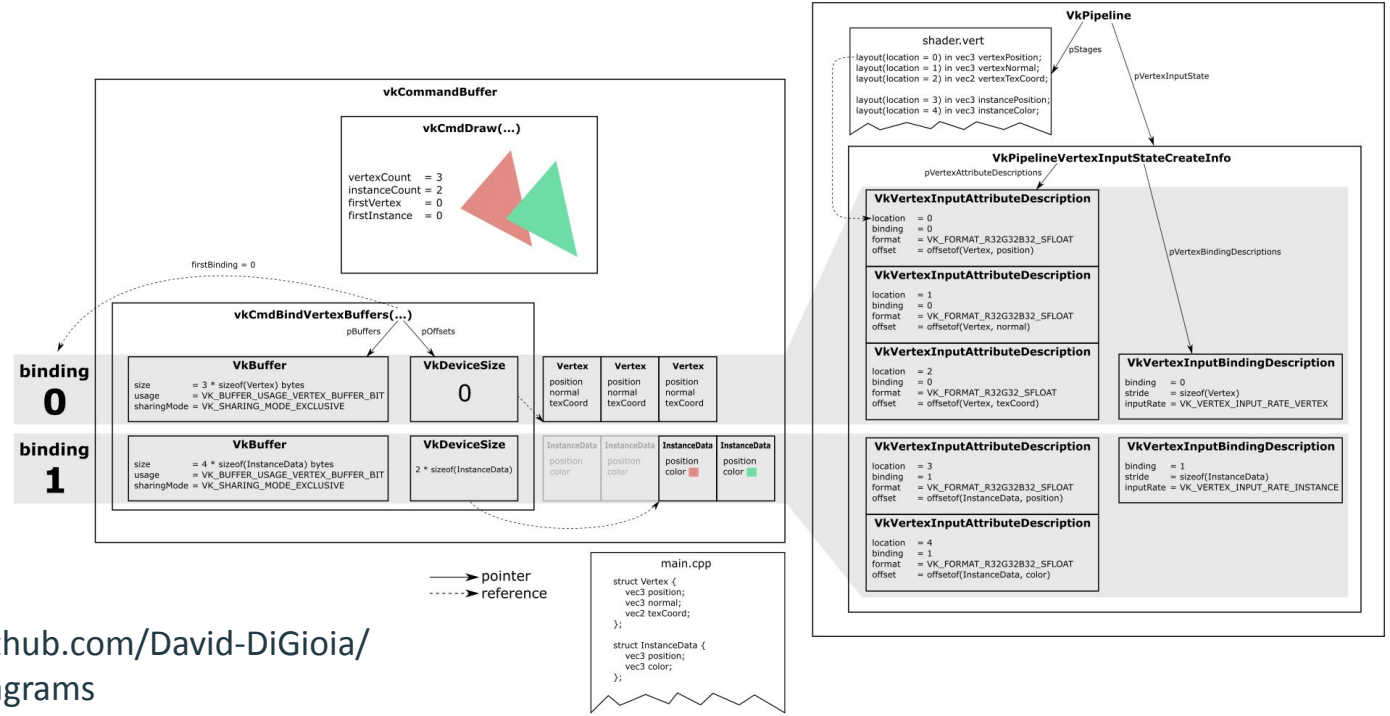
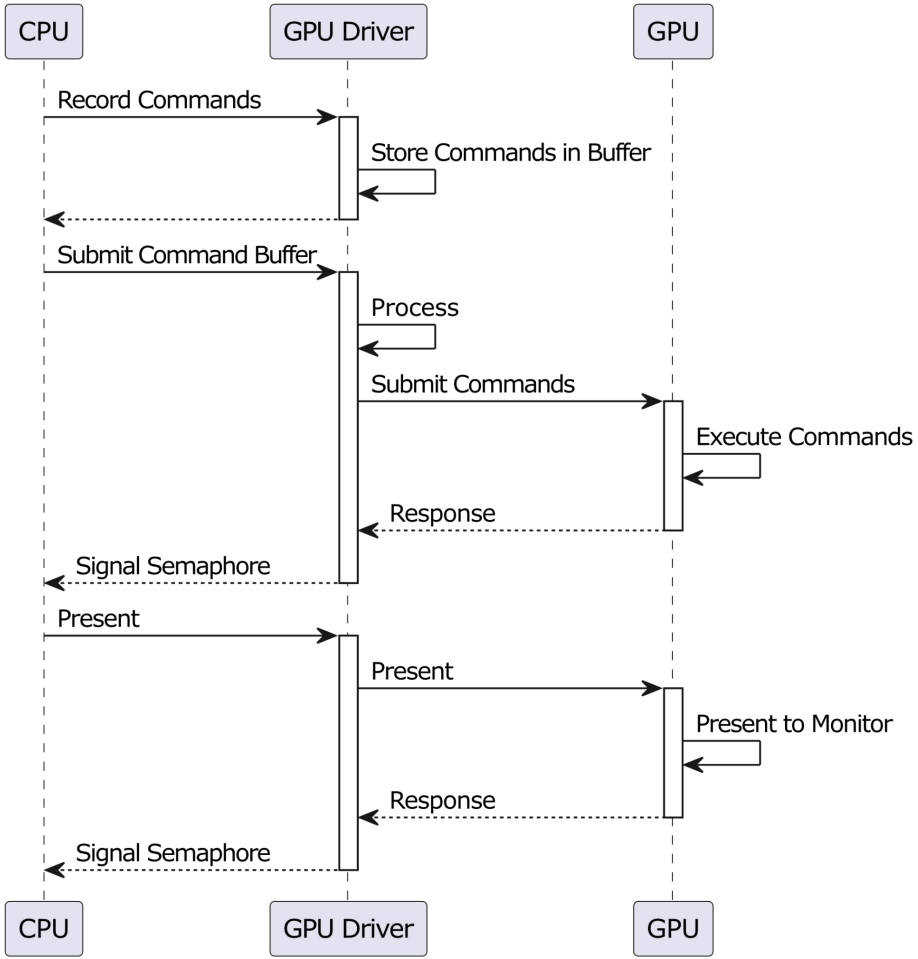
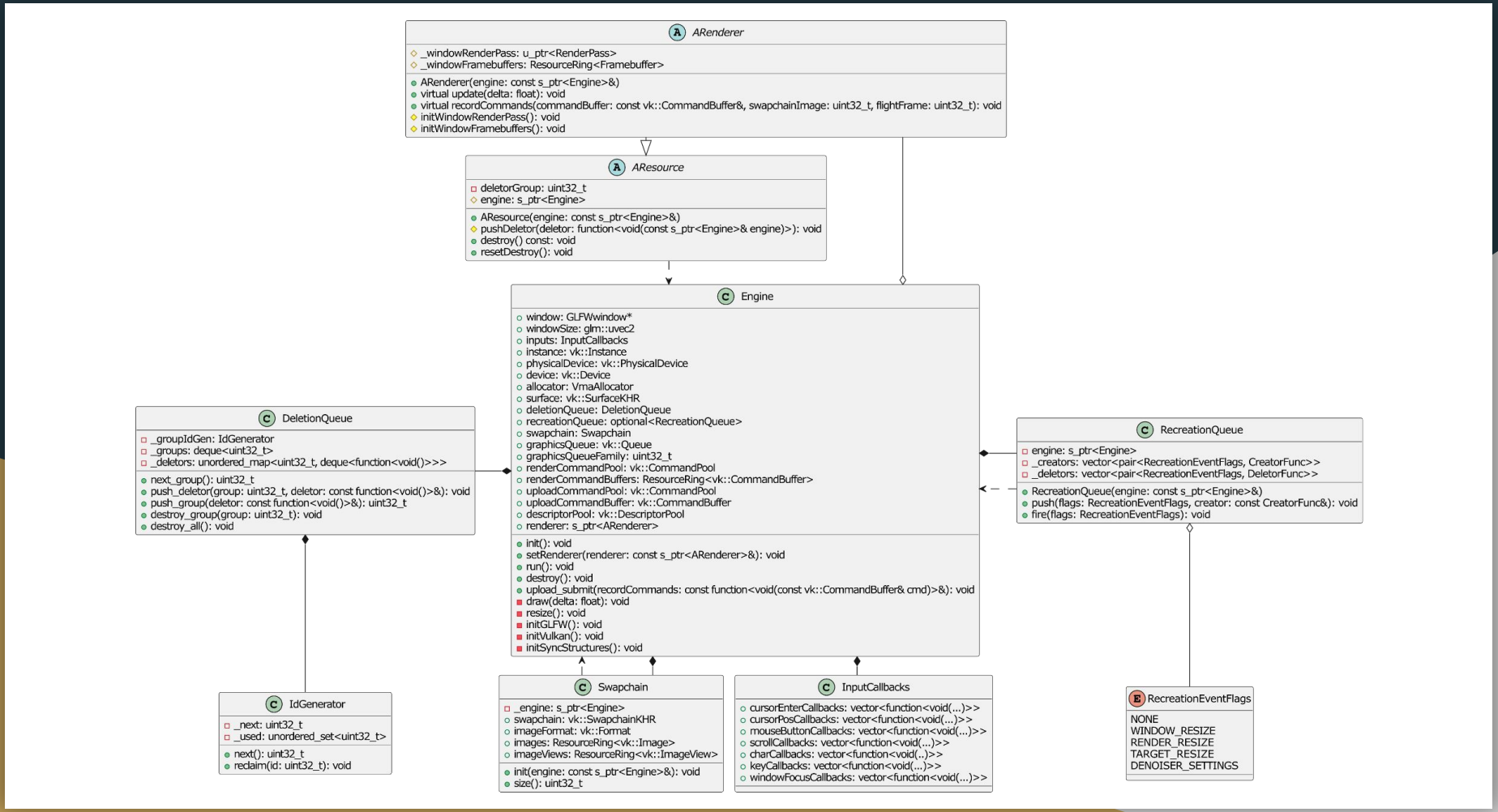


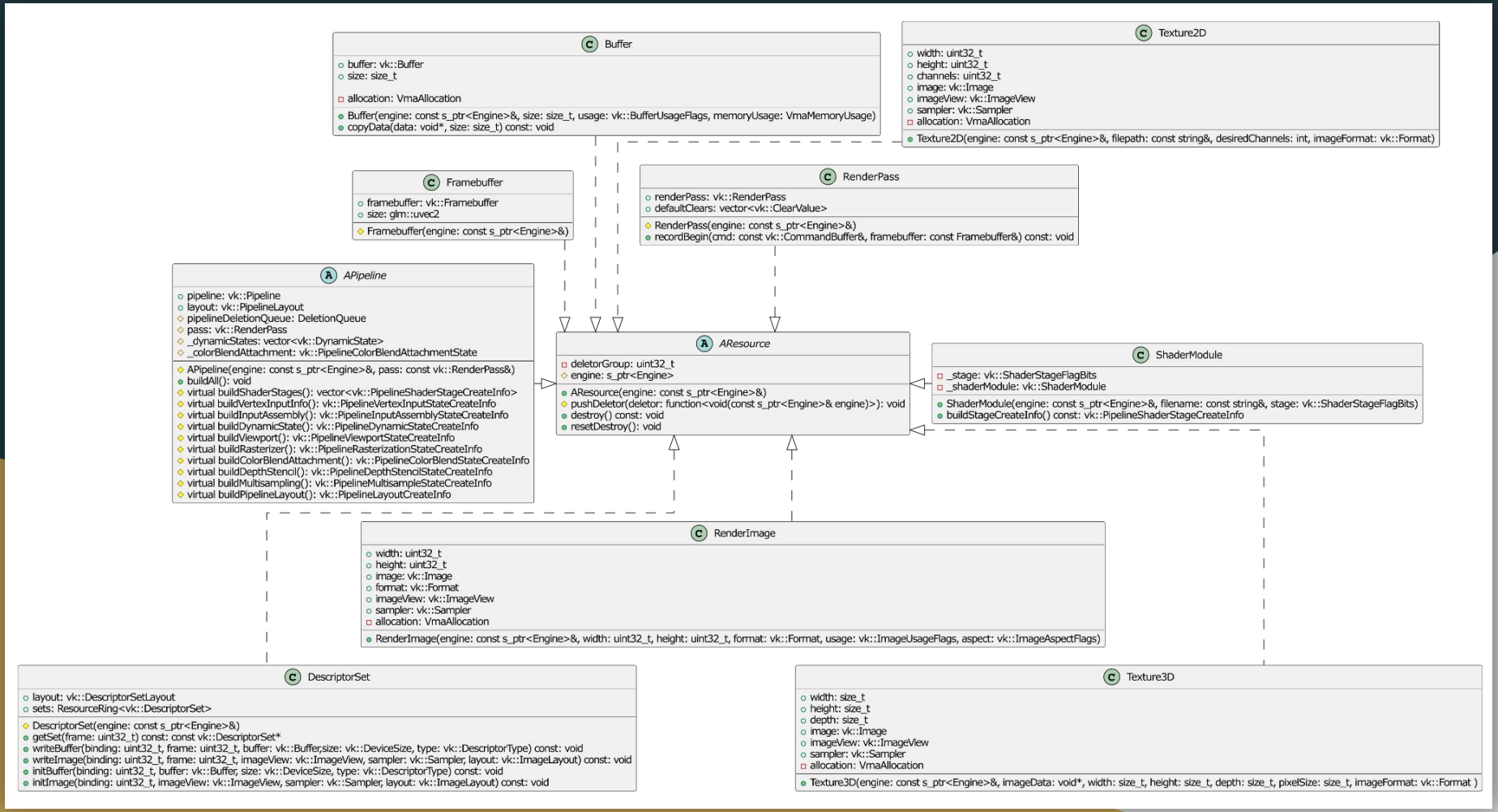
Image: https://github.com/David-DiGioia/vulkan-diagrams

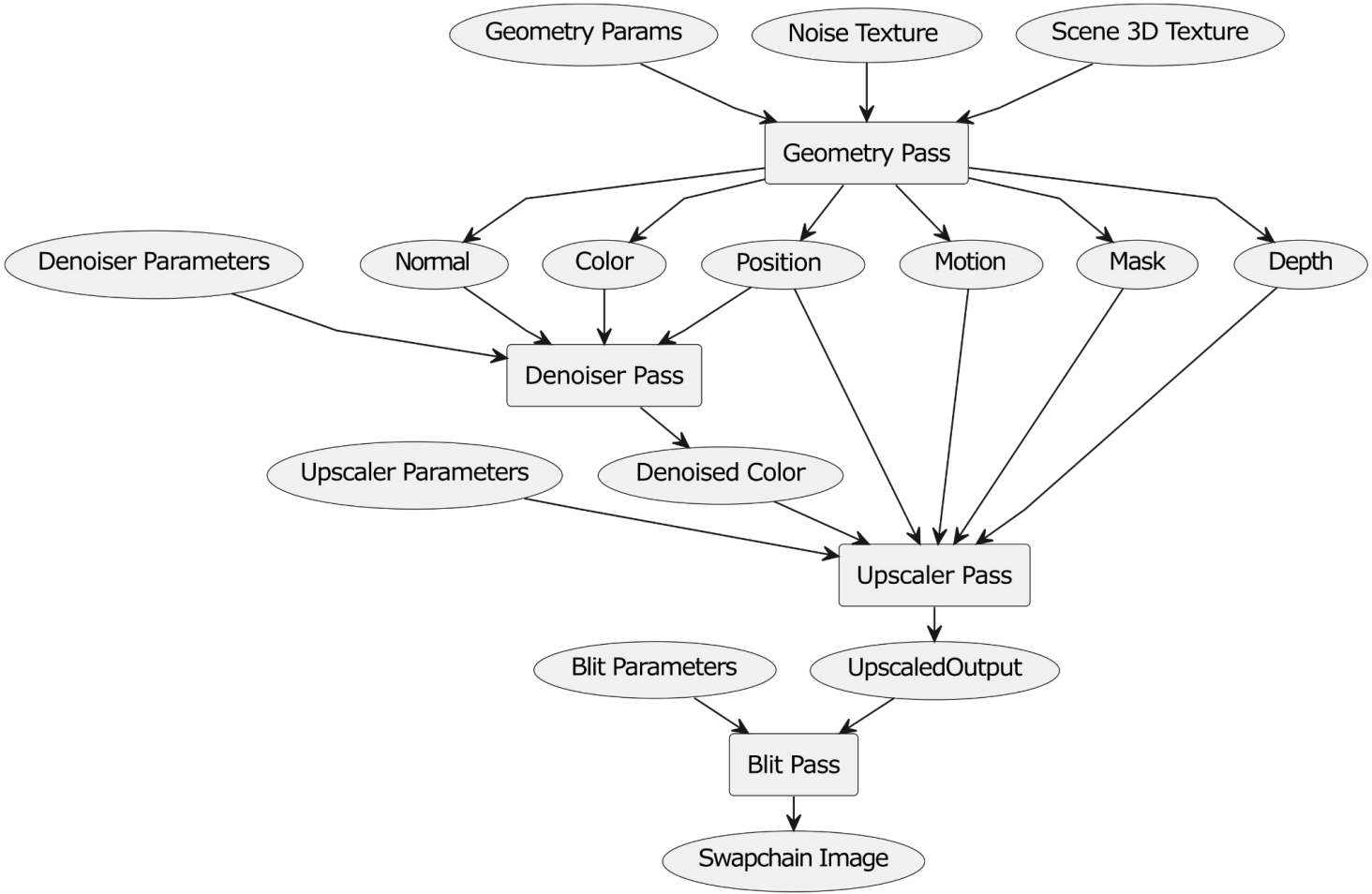

CPU
GPU Driver
GPU
Record Commands
Store Commands in Buffer
Submit Command Buffer
Process
Submit Commands
Execute Commands
Response
Signal Semaphore
Present
Present
Present to Monitor
Response
Signal Semaphore
CPU
GPU Driver
GPU


ARenderer
_windowRenderPass: u_ptr<RenderPass>
_windowFramebuffers: ResourceRing<Framebuffer>
ARenderer(engine: const s_ptr<Engine>&)
virtual update(delta: float): void
virtual recordCommands(commandBuffer: const vk::CommandBuffer&, swapchainImage: uint32_t, flightFrame: uint32_t): void
initWindowRenderPass(): void
initWindowFramebuffers(): void
AResource
deletorGroup: uint32_t
engine: s_ptr<Engine>
AResource(engine: const s_ptr<Engine>&)
pushDeletor(deletor: function<void(const s_ptr<Engine>& engine)>): void
destroy() const: void
resetDestroy(): void
Engine
window: GLFWwindow*
windowSize: glm::uvec2
inputs: InputCallbacks
instance: vk::Instance
physicalDevice: vk::PhysicalDevice
device: vk::Device
allocator: VmaAllocator
surface: vk::SurfaceKHR
deletionQueue: DeletionQueue
recreationQueue: optional<RecreationQueue>
swapchain: Swapchain
graphicsQueue: vk::Queue
graphicsQueueFamily: uint32_t
renderCommandPool: vk::CommandPool
renderCommandBuffers: ResourceRing<vk::CommandBuffer>
uploadCommandPool: vk::CommandPool
uploadCommandBuffer: vk::CommandBuffer
descriptorPool: vk::DescriptorPool
renderer: s_ptr<ARenderer>
init(): void
setRenderer(renderer: const s_ptr<ARenderer>&): void
run(): void
destroy(): void
upload_submit(recordCommands: const function<void(const vk::CommandBuffer& cmd)>&): void
draw(delta: float): void
resize(): void
initGLFW(): void
initVulkan(): void
initSyncStructures(): void
DeletionQueue
_groupIdGen: IdGenerator
_groups: deque<uint32_t>
_deletors: unordered_map<uint32_t, deque<function<void()>>>
next_group(): uint32_t
push_deletor(group: uint32_t, deletor: const function<void()>&): void
push_group(deletor: const function<void()>&): uint32_t
destroy_group(group: uint32_t): void
destroy_all(): void
RecreationQueue
engine: s_ptr<Engine>
_creators: vector<pair<RecreationEventFlags, CreatorFunc>>
_deletors: vector<pair<RecreationEventFlags, DeletorFunc>>
RecreationQueue(engine: const s_ptr<Engine>&)
push(flags: RecreationEventFlags, creator: const CreatorFunc&): void
fire(flags: RecreationEventFlags): void
IdGenerator
_next: uint32_t
_used: unordered_set<uint32_t>
next(): uint32_t
reclaim(id: uint32_t): void
Swapchain
_engine: s_ptr<Engine>
swapchain: vk::SwapchainKHR
imageFormat: vk::Format
images: ResourceRing<vk::Image>
imageViews: ResourceRing<vk::ImageView>
init(engine: const s_ptr<Engine>&): void
size(): uint32_t
InputCallbacks
cursorEnterCallbacks: vector<function<void(...)>>
cursorPosCallbacks: vector<function<void(...)>>
mouseButtonCallbacks: vector<function<void(...)>>
scrollCallbacks: vector<function<void(...)>>
charCallbacks: vector<function<void(..)>>
keyCallbacks: vector<function<void(...)>>
windowFocusCallbacks: vector<function<void(...)>>
RecreationEventFlags
NONE
WINDOW_RESIZE
RENDER_RESIZE
TARGET_RESIZE
DENOISER_SETTINGS

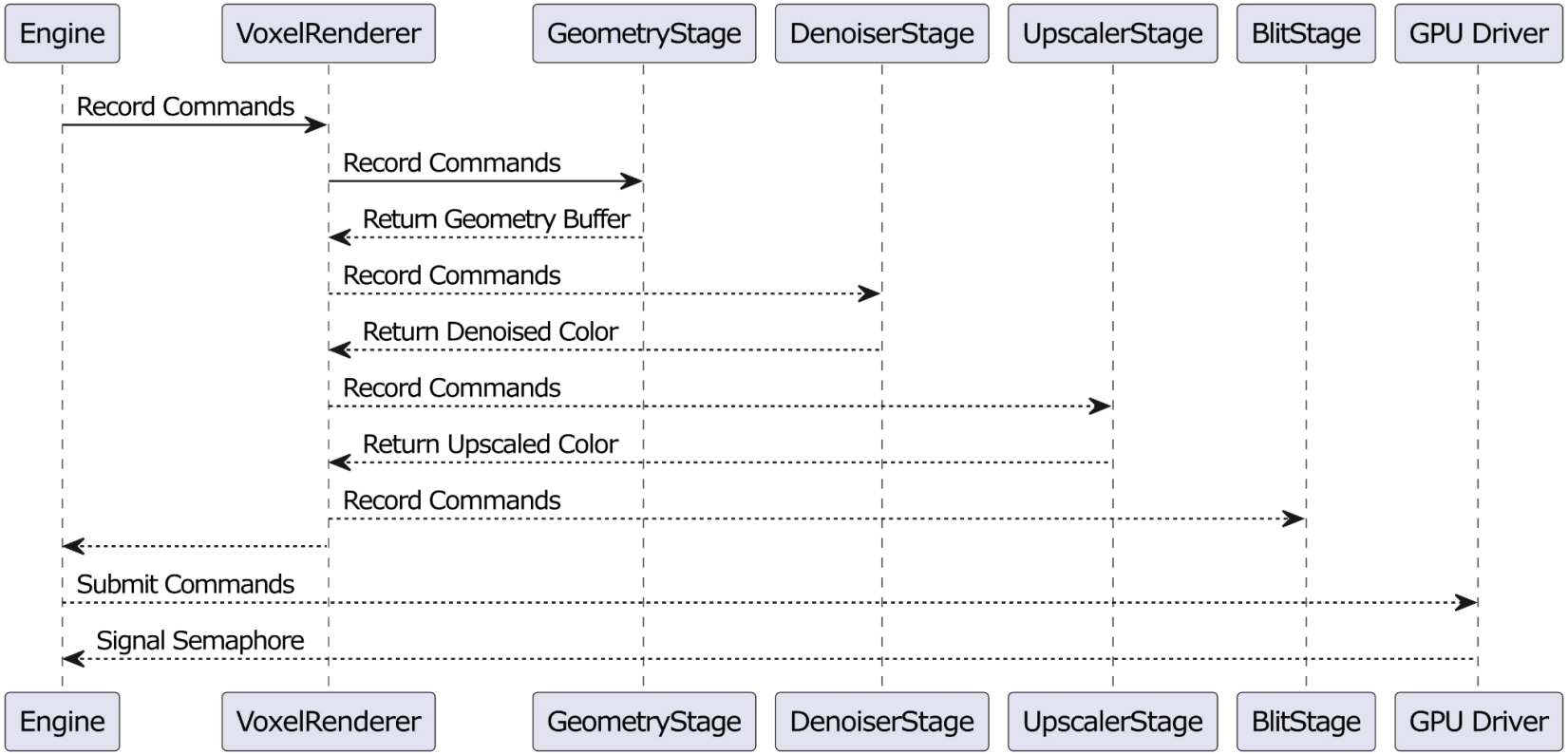
Initializing a Vulkan resource is tedious

```
vk::DeviceSize imageSize = width * height * depth * pixelSize;

// Create CPU-side buffer to hold data
Buffer stagingBuffer(engine, imageSize, vk::BufferUsageFlagBits::eTransferSrc, VMA_MEMORY_USAGE_CPU_ONLY);

// Copy data to buffer
stagingBuffer.copyData(imageData, static_cast<size_t>(imageSize));

// Extents
vk::Extent3D imageExtent;
imageExtent.width = static_cast<uint32_t>(width);
imageExtent.height = static_cast<uint32_t>(height);
imageExtent.depth = static_cast<uint32_t>(depth);

// Image create info
vk::ImageCreateInfo imageInfo = {};
imageInfo.imageType = vk::ImageType::e3D;
imageInfo.extent = imageExtent;
imageInfo.format = imageFormat;
imageInfo.usage = vk::ImageUsageFlagBits::eSampled | vk::ImageUsageFlagBits::eTransferDst;
imageInfo.mipLevels = 1;
imageInfo.arrayLayers = 1;
imageInfo.samples = vk::SampleCountFlagBits::e1;
imageInfo.tiling = vk::ImageTiling::eOptimal;

// Allocation info
VmaAllocationCreateInfo imageAllocInfo = {};
imageAllocInfo.usage = VMA_MEMORY_USAGE_GPU_ONLY;

// Actually create the image
VkImageCreateInfo imageInfoC = VkImageCreateInfo(imageInfo);
VkImage imageC;
auto res = vmaCreateImage(engine->allocator, &imageInfoC, &imageAllocInfo, &imageC, &allocation, nullptr);
vk::resultCheck(vk::Result(res), "Error creating image");
image = vk::Image(imageC);
```

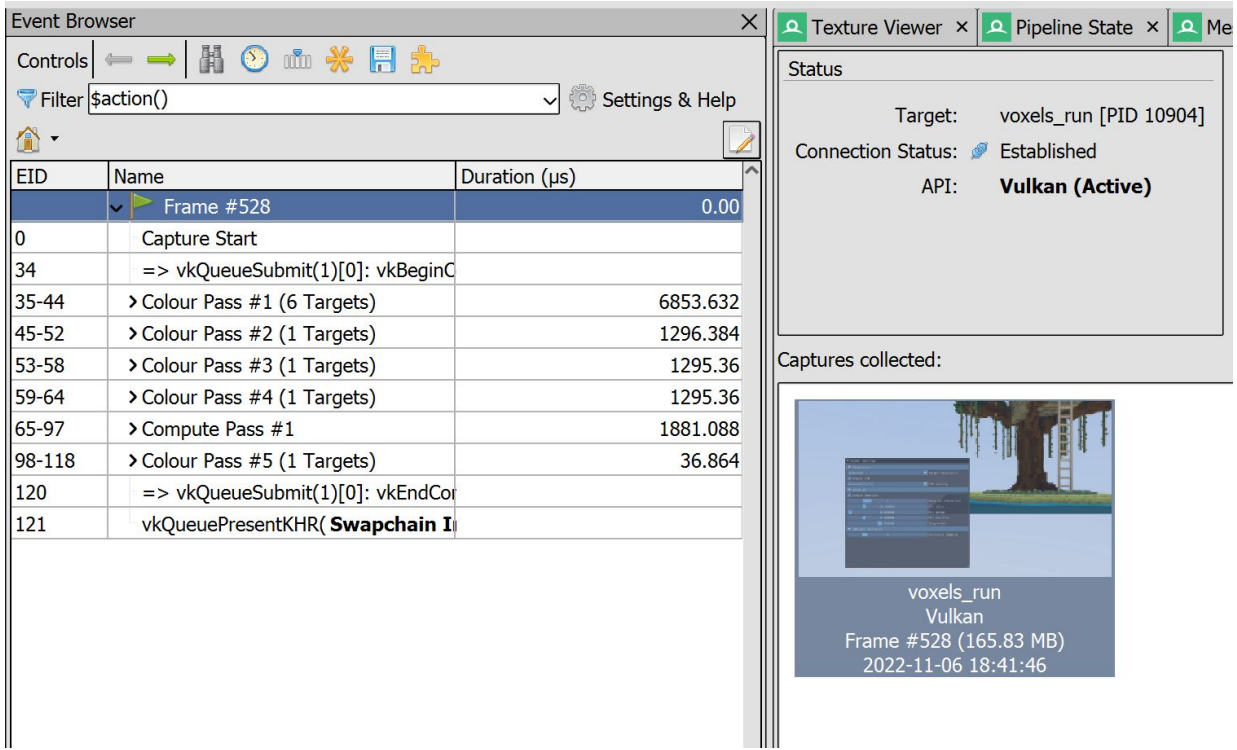

Buffer
buffer: vk::Buffer
size: size_t
allocation: VmaAllocation
Buffer(engine: const s_ptr<Engine>&, size: size_t, usage: vk::BufferUsageFlags, memoryUsage: VmaMemoryUsage)
copyData(data: void*, size: size_t) const: void
Texture2D
width: uint32_t
height: uint32_t
channels: uint32_t
image: vk::Image
imageView: vk::ImageView
sampler: vk::Sampler
allocation: VmaAllocation
Texture2D(engine: const s_ptr<Engine>&, filepath: const string&, desiredChannels: int, imageFormat: vk::Format)
Framebuffer
framebuffer: vk::Framebuffer
size: glm::uvec2
Framebuffer(engine: const s_ptr<Engine>&)
RenderPass
renderPass: vk::RenderPass
defaultClears: vector<vk::ClearValue>
RenderPass(engine: const s_ptr<Engine>&)
recordBegin(cmd: const vk::CommandBuffer&, framebuffer: const Framebuffer&) const: void
APipeline
pipeline: vk::Pipeline
layout: vk::PipelineLayout
pipelineDeletionQueue: DeletionQueue
pass: vk::RenderPass
_dynamicStates: vector<vk::DynamicState>
_colorBlendAttachment: vk::PipelineColorBlendAttachmentState
APipeline(engine: const s_ptr<Engine>&, pass: const vk::RenderPass&)
buildAll(): void
virtual buildShaderStages(): vector<vk::PipelineShaderStageCreateInfo>
virtual buildVertexInputInfo(): vk::PipelineVertexInputStateCreateInfo
virtual buildInputAssembly(): vk::PipelineInputAssemblyStateCreateInfo
virtual buildDynamicState(): vk::PipelineDynamicStateCreateInfo
virtual buildViewport(): vk::PipelineViewportStateCreateInfo
virtual buildRasterizer(): vk::PipelineRasterizationStateCreateInfo
virtual buildColorBlendAttachment(): vk::PipelineColorBlendStateCreateInfo
virtual buildDepthStencil(): vk::PipelineDepthStencilStateCreateInfo
virtual buildMultisampling(): vk::PipelineMultisampleStateCreateInfo
virtual buildPipelineLayout(): vk::PipelineLayoutCreateInfo
AResource
deletorGroup: uint32_t
engine: s_ptr<Engine>
AResource(engine: const s_ptr<Engine>&)
pushDeletor(deletor: function<void(const s_ptr<Engine>& engine)>): void
destroy() const: void
resetDestroy(): void
ShaderModule
_stage: vk::ShaderStageFlagBits
_shaderModule: vk::ShaderModule
ShaderModule(engine: const s_ptr<Engine>&, filename: const string&, stage: vk::ShaderStageFlagBits)
buildStageCreateInfo() const: vk::PipelineShaderStageCreateInfo
RenderImage
width: uint32_t
height: uint32_t
image: vk::Image
format: vk::Format
imageView: vk::ImageView
sampler: vk::Sampler
allocation: VmaAllocation
RenderImage(engine: const s_ptr<Engine>&, width: uint32_t, height: uint32_t, format: vk::Format, usage: vk::ImageUsageFlags, aspect: vk::ImageAspectFlags)
DescriptorSet
layout: vk::DescriptorSetLayout
sets: ResourceRing<vk::DescriptorSet>
DescriptorSet(engine: const s_ptr<Engine>&)
getSet(frame: uint32_t) const: const vk::DescriptorSet*
writeBuffer(binding: uint32_t, frame: uint32_t, buffer: vk::Buffer,size: vk::DeviceSize, type: vk::DescriptorType) const: void
writeImage(binding: uint32_t, frame: uint32_t, imageView: vk::ImageView, sampler: vk::Sampler, layout: vk::ImageLayout) const: void
initBuffer(binding: uint32_t, buffer: vk::Buffer, size: vk::DeviceSize, type: vk::DescriptorType) const: void
initImage(binding: uint32_t, imageView: vk::ImageView, sampler: vk::Sampler, layout: vk::ImageLayout) const: void
Texture3D
width: size_t
height: size_t
depth: size_t
image: vk::Image
imageView: vk::ImageView
sampler: vk::Sampler
allocation: VmaAllocation
Texture3D(engine: const s_ptr<Engine>&, imageData: void*, width: size_t, height: size_t, depth: size_t, pixelSize: size_t, imageFormat: vk::Format)


Geometry Params
Noise Texture
Scene 3D Texture
Geometry Pass
Denoiser Parameters
Normal
Color
Position
Motion
Mask
Depth
Denoiser Pass
Upscaler Parameters
Denoised Color
Upscaler Pass
Blit Parameters
UpscaledOutput
Blit Pass
Swapchain Image


UpscalerStage
jitterX: float
jitterY: float
frameCount: int
_target: u_ptr<RenderImage>
_fsrInterface: FfxFsr2Interface*
_fsrContext: FfxFsr2Context*
_fsrScratchBuffer: void*
_deltaMsec: float
UpscalerStage(engine: const s_ptr<Engine>&, settings: const s_ptr<VoxelRenderSettings>&)
update(delta: float): void
record(cmd: const vk::CommandBuffer&, ...): const RenderImage&
wrapRenderImage(image: const RenderImage&): FfxResource
DenoiserStage
_kernelBuffer: u_ptr<Buffer>
_offsetBuffer: u_ptr<Buffer>
_iterationParamsBuffers: vector<Buffer>
_iterationDescriptors: vector<DescriptorSet>
_colorTargets: ResourceRing<RenderImage>
_renderPasses: ResourceRing<RenderPass>
_framebuffers: ResourceRing<Framebuffer>
_pipeline: u_ptr<DenoiserPipeline>
DenoiserStage(engine: const s_ptr<Engine>&, settings: const s_ptr<VoxelRenderSettings>&)
updateParameters() const: void
record(cmd: const vk::CommandBuffer&, flightFrame: uint32_t, ...) const: const RenderImage&
BlitStage
_offsets: BlitOffsets
_offsetsBuffer: u_ptr<Buffer>
_pipeline: u_ptr<BlitPipeline>
BlitStage(engine: const s_ptr<Engine>&, settings: const s_ptr<VoxelRenderSettings>&, renderPass: const RenderPass&)
record(cmd: const vk::CommandBuffer&, flightFrame: uint32_t, ...): void
VoxelRenderer
_camera: u_ptr<CameraController>
_settings: s_ptr<VoxelRenderSettings>
_scene: s_ptr<VoxelScene>
_noiseTexture: s_ptr<Texture2D>
_geometryStage: u_ptr<GeometryStage>
_denoiserStage: u_ptr<DenoiserStage>
_upscalerStage: u_ptr<UpscalerStage>
_blitStage: u_ptr<BlitStage>
_imguiRenderer: u_ptr<ImguiRenderer>
_time: float
VoxelRenderer(engine: const s_ptr<Engine>&)
virtual update(delta: float) override: void
virtual recordCommands(commandBuffer: const vk::CommandBuffer&, swapchainImage: uint32_t, flightFrame: uint32_t) override: void
VoxelScene
width: uint32_t
height: uint32_t
depth: uint32_t
sceneTexture: optional<Texture3D>
paletteBuffer: optional<Buffer>
VoxelScene(engine: const s_ptr<Engine>&, filename: const string&)
GeometryStage
_parameters: VolumeParameters
_parametersBuffer: u_ptr<Buffer>
_colorTarget: u_ptr<RenderImage>
_depthTarget: u_ptr<RenderImage>
_motionTarget: u_ptr<RenderImage>
_maskTarget: u_ptr<RenderImage>
_normalTarget: u_ptr<RenderImage>
_positionTargets: ResourceRing<RenderImage>
_renderPass: u_ptr<RenderPass>
_framebuffers: ResourceRing<Framebuffer>
_pipeline: u_ptr<VoxelSDFPipeline>
GeometryStage(engine: const s_ptr<Engine>&, settings: const s_ptr<VoxelRenderSettings>&, ...)
record(cmd: const vk::CommandBuffer&, flightFrame: uint32_t): GeometryBuffer
getPipelineLayout() const: const vk::PipelineLayout&
ARenderer
_windowRenderPass: u_ptr<RenderPass>
_windowFramebuffers: ResourceRing<Framebuffer>
ARenderer(engine: const s_ptr<Engine>&)
virtual update(delta: float): void
virtual recordCommands(commandBuffer: const vk::CommandBuffer&, swapchainImage: uint32_t, flightFrame: uint32_t): void
initWindowRenderPass(): void
initWindowFramebuffers(): void
AVoxelRenderStage
_settings: s_ptr<VoxelRenderSettings>
AVoxelRenderStage(engine: const s_ptr<Engine>&, settings: const s_ptr<VoxelRenderSettings>&)
AResource
deletorGroup: uint32_t
engine: s_ptr<Engine>
AResource(engine: const s_ptr<Engine>&)
pushDeletor(deletor: function<void(const s_ptr<Engine>& engine)>): void
destroy() const: void
resetDestroy(): void


Engine
VoxelRenderer
GeometryStage
DenoiserStage
UpscalerStage
BlitStage
GPU Driver
Record Commands
Record Commands
Return Geometry Buffer
Record Commands
Return Denoised Color
Record Commands
Return Upscaled Color
Record Commands
Submit Commands
Signal Semaphore
Engine
VoxelRenderer
GeometryStage
DenoiserStage
UpscalerStage
BlitStage
GPU Driver

RenderDoc can measure performance



Progress is at ~75%

MVP ✔

- Scene storage ✔
- Ray-traced direct lighting ✔
- Ray-traced ambient occlusion ✔
- Denoiser ✔

Version 1.0

- Rendering settings GUI ✔
- Magicavoxel .vox scenes ✔
- Skybox image-based lighting
- Ray-traced reflections

Questions?

Topics:

- Abstracting Vulkan
- Voxel Ray-tracing Algorithm
- Render Passes
- Overall Architecture