Voxel Ray Tracing

Ethan Tucker
Advisor: Dr. Hsu

The ray-tracing algorithm sends a ray through each pixel on the screen

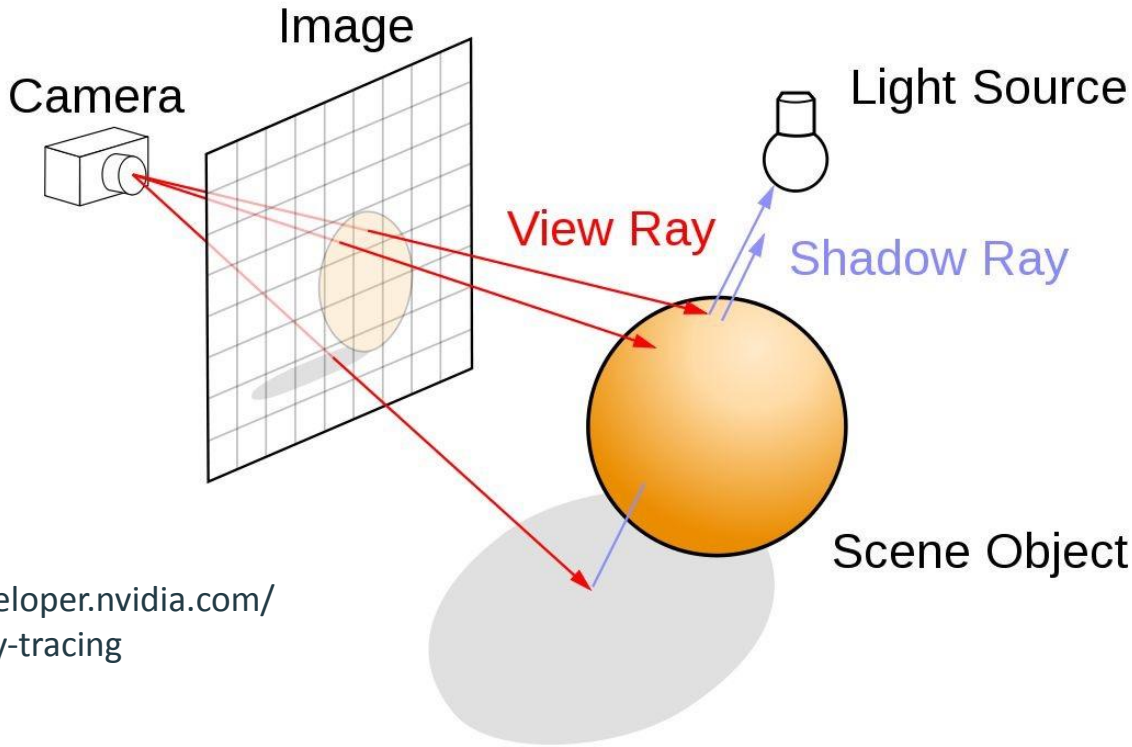


Image:
https://developer.nvidia.com/discover/ray-tracing

Voxels are commonly used in games or data visualizations

Teardown is a major inspiration for this project

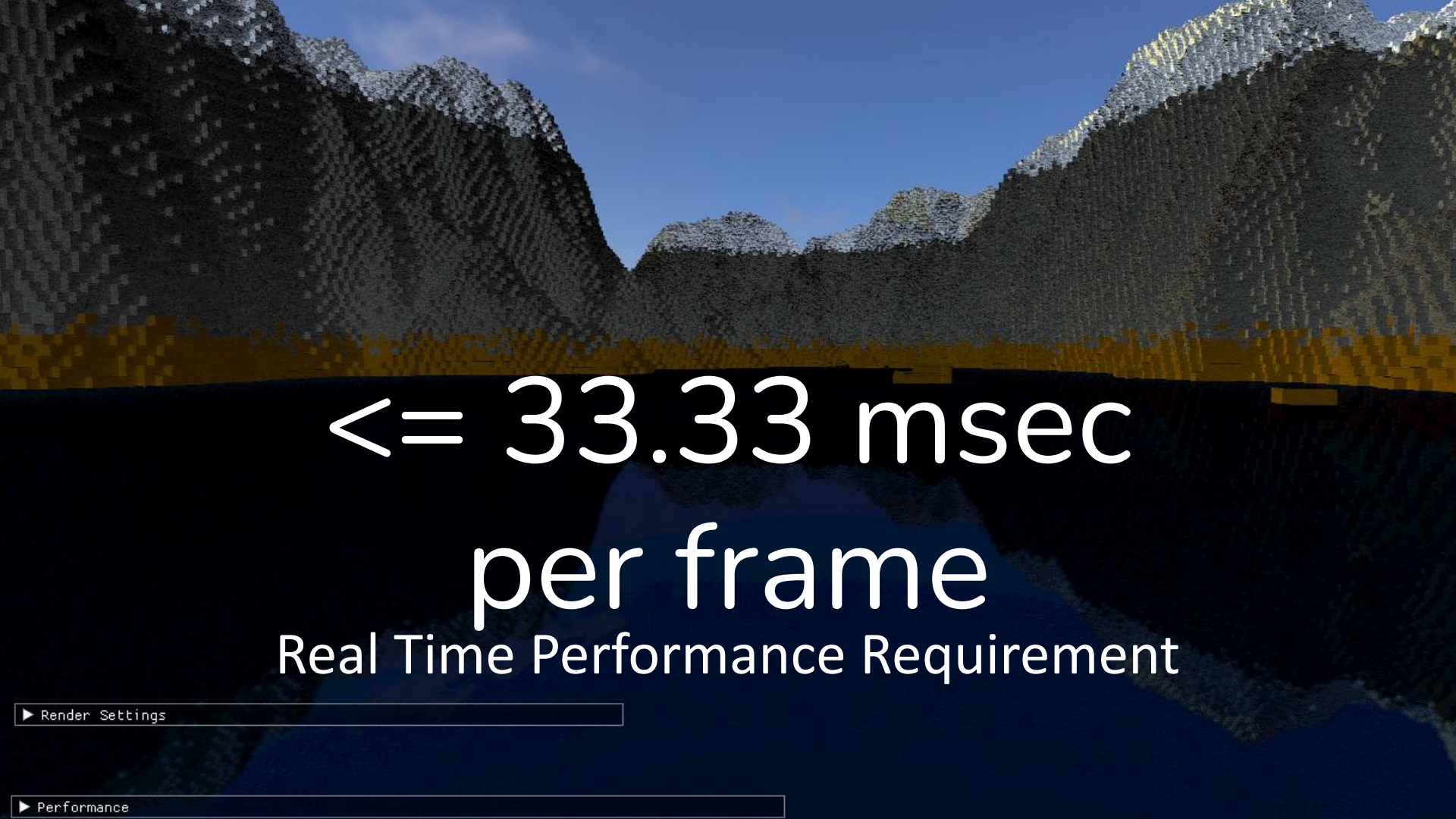
Feature Requirements

Minimum Viable Product

- Voxel scene storage in 3D texture ✔
- Screen-space voxel ray tracing for direct lighting ✔
- Ray-traced ambient occlusion and shadows ✔
- Image denoising filter ✔

Version 1.0

- GUI interface to adjust rendering settings ✔
- Loading existing scenes from Magicavoxel .vox format ✔
- Skybox image-based lighting ✔
- Ray-traced reflections ✔


<= 33.33 msec
per frame
Real Time Performance Requirement
▶ Render Settings
▶ Performance

Project Tools

- C++ Desktop Application
- CMake Buildsystem
- CLion IDE
- Vulkan Rendering API
- GLSL Shaders
- Glfw Window
- ImGui GUI

Rendering happens in our fragment shader, which runs in parallel on GPU

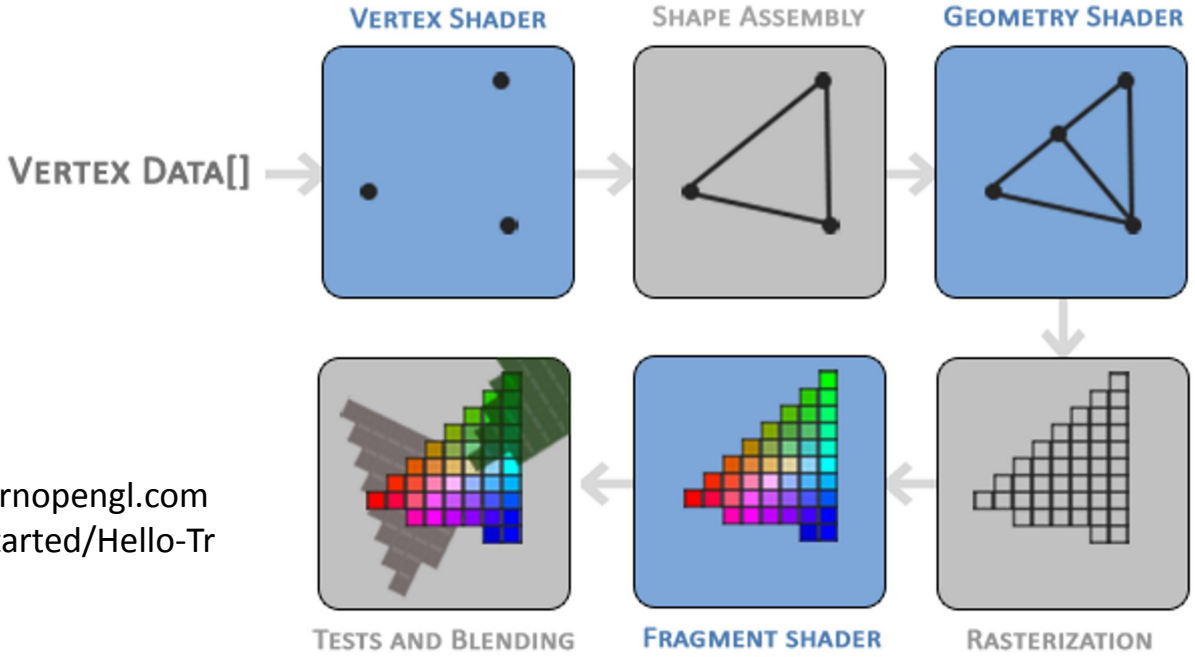
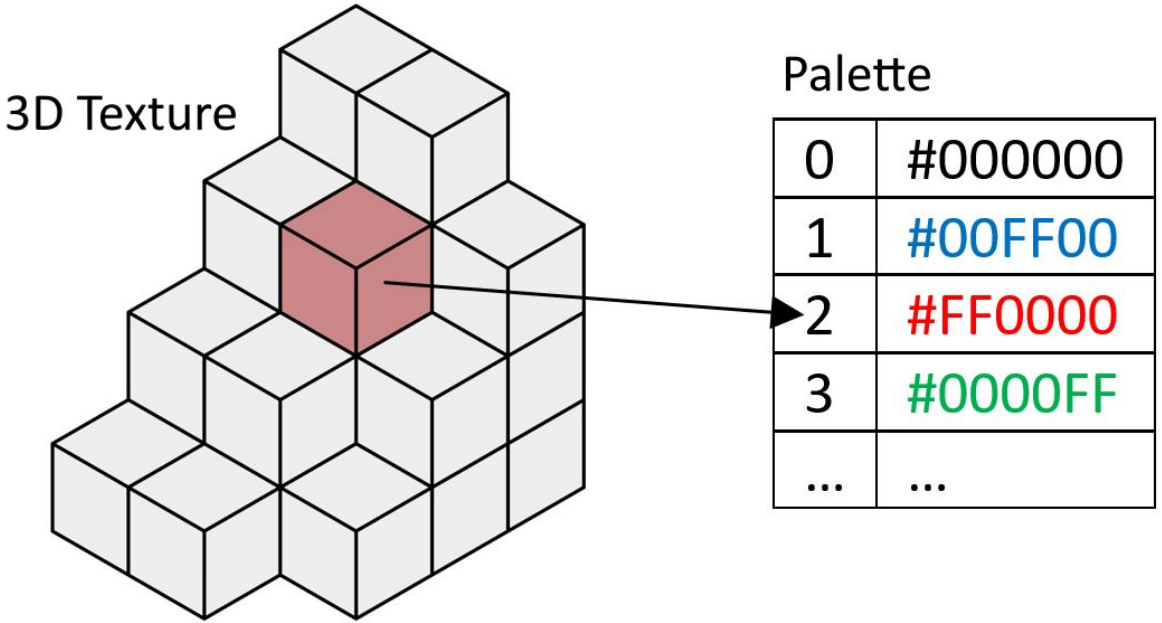


Image: https://learnopengl.com/Getting-started/Hello-Triangle

Voxel data is stored in a combination of a 3D texture and palette



The DDA algorithm implements ray tracing/marching for voxel grids

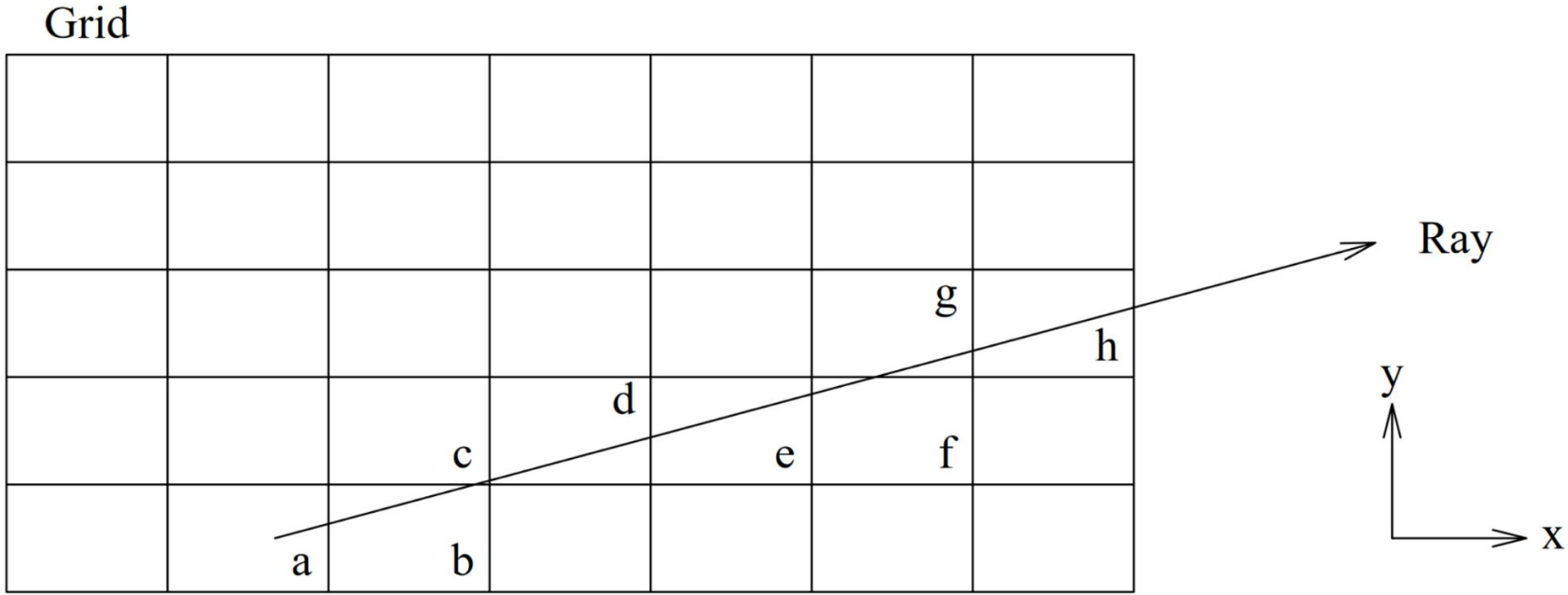


Image: Amantides & Woo “A Fast Voxel Traversal Algorithm for Ray Tracing”

The denoiser pass uses a A-Trous blur to smooth out the scene

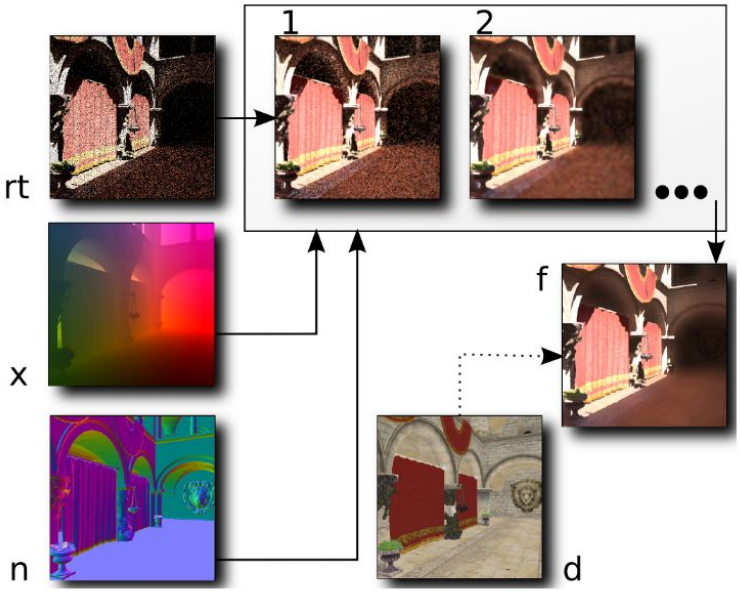
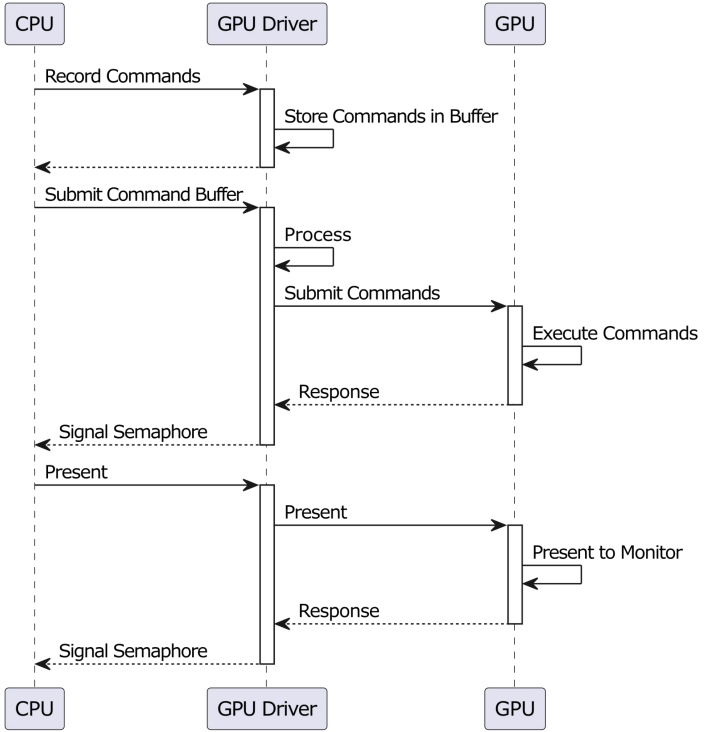
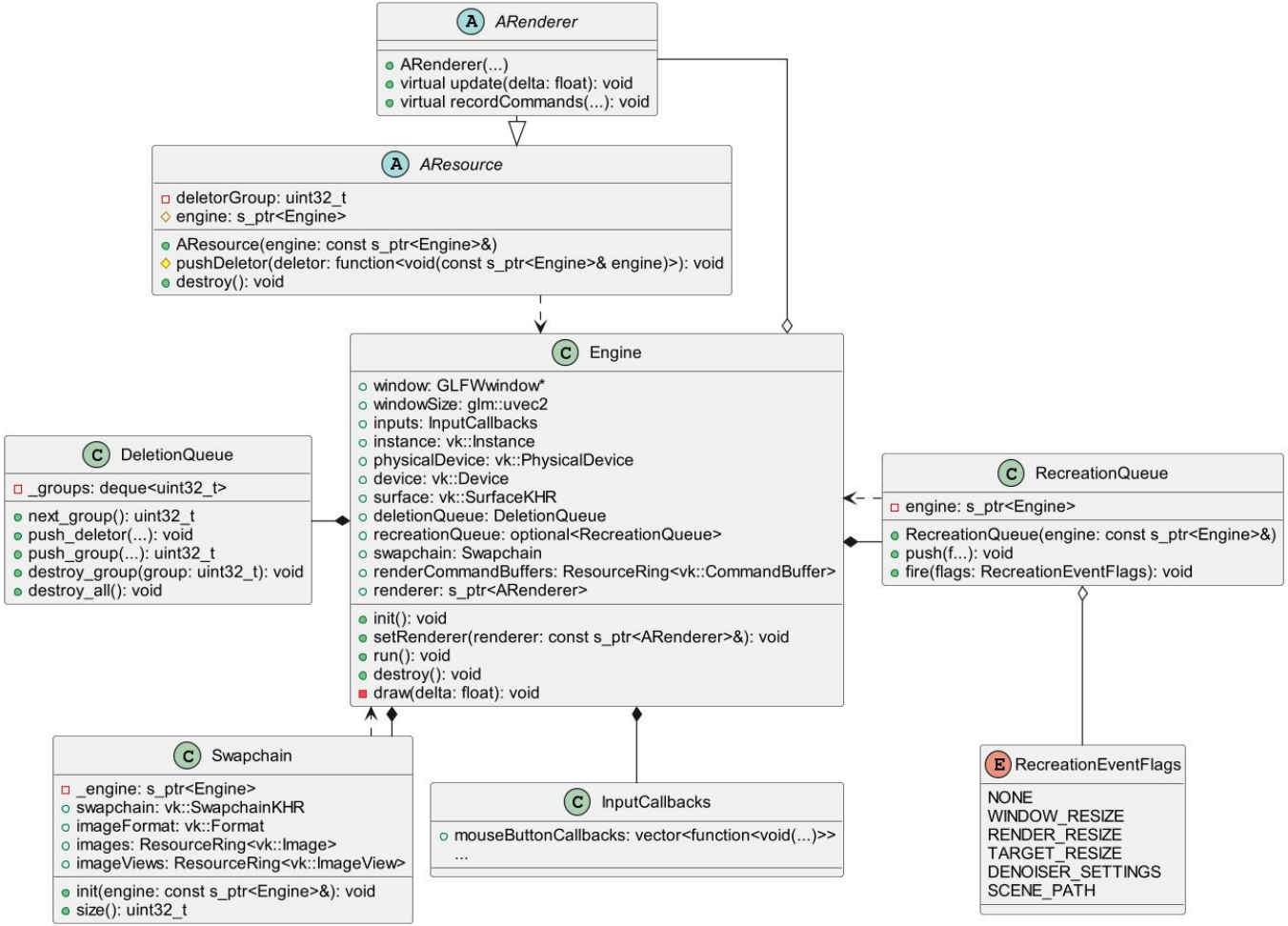


Image: Dammertz et. al. “Edge-Avoiding À-Trous Wavelet Transform for fast Global Illumination Filtering”

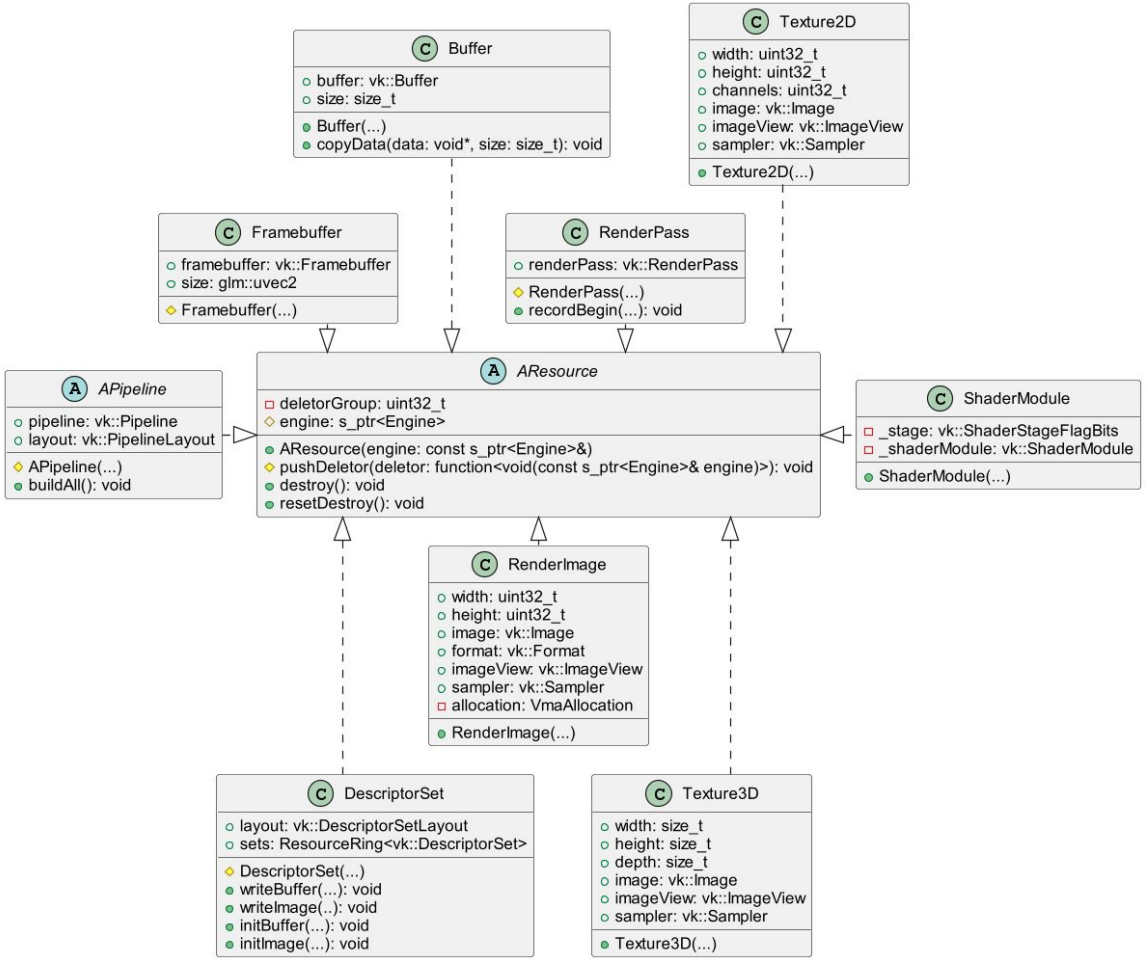
The Vulkan API requests the GPU to execute our commands

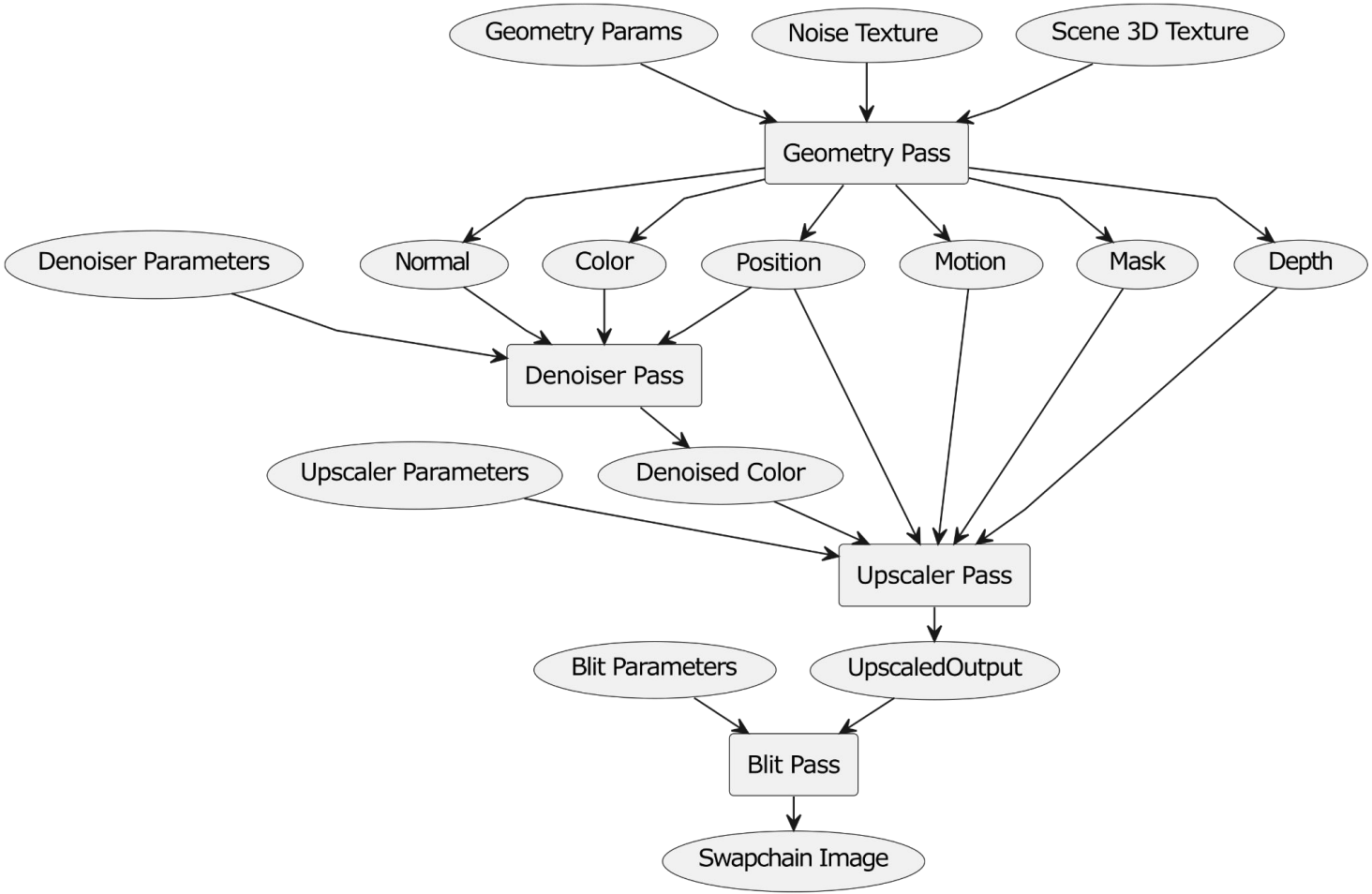



ARenderer
ARenderer(...)
virtual update(delta: float): void
virtual recordCommands(...): void
AResource
deletorGroup: uint32_t
engine: s_ptr<Engine>
AResource(engine: const s_ptr<Engine>&)
pushDeletor(deletor: function<void(const s_ptr<Engine>& engine)>): void
destroy(): void
Engine
window: GLFWwindow*
windowSize: glm::uvec2
inputs: InputCallbacks
instance: vk::Instance
physicalDevice: vk::PhysicalDevice
device: vk::Device
surface: vk::SurfaceKHR
deletionQueue: DeletionQueue
recreationQueue: optional<RecreationQueue>
swapchain: Swapchain
renderCommandBuffers: ResourceRing<vk::CommandBuffer>
renderer: s_ptr<ARenderer>
init(): void
setRenderer(renderer: const s_ptr<ARenderer>&): void
run(): void
destroy(): void
draw(delta: float): void
DeletionQueue
_groups: deque<uint32_t>
next_group(): uint32_t
push_deletor(...): void
push_group(...): uint32_t
destroy_group(group: uint32_t): void
destroy_all(): void
RecreationQueue
engine: s_ptr<Engine>
RecreationQueue(engine: const s_ptr<Engine>&)
push(f...): void
fire(flags: RecreationEventFlags): 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
mouseButtonCallbacks: vector<function<void(...)>>
...
RecreationEventFlags
NONE
WINDOW_RESIZE
RENDER_RESIZE
TARGET_RESIZE
DENOISER_SETTINGS
SCENE_PATH

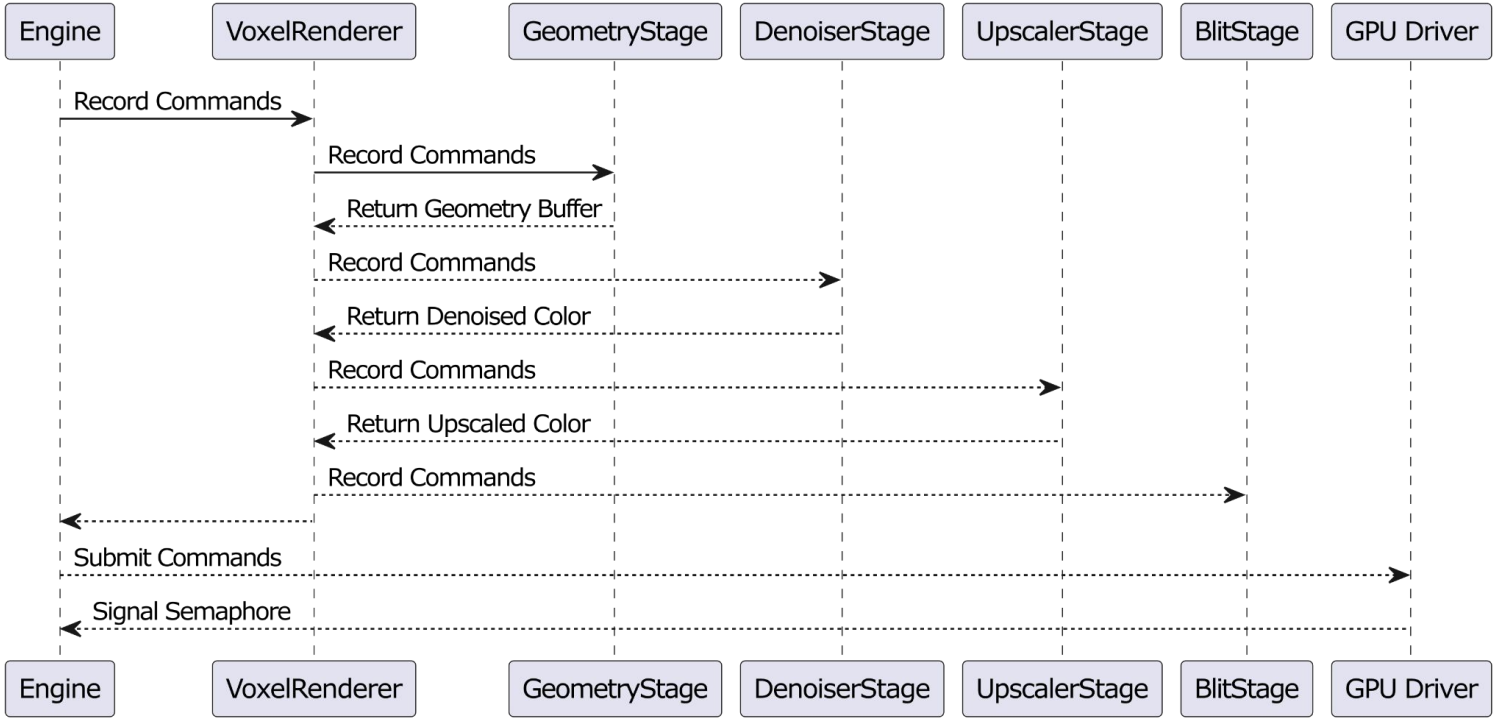
Initializing a Vulkan resource is tedious

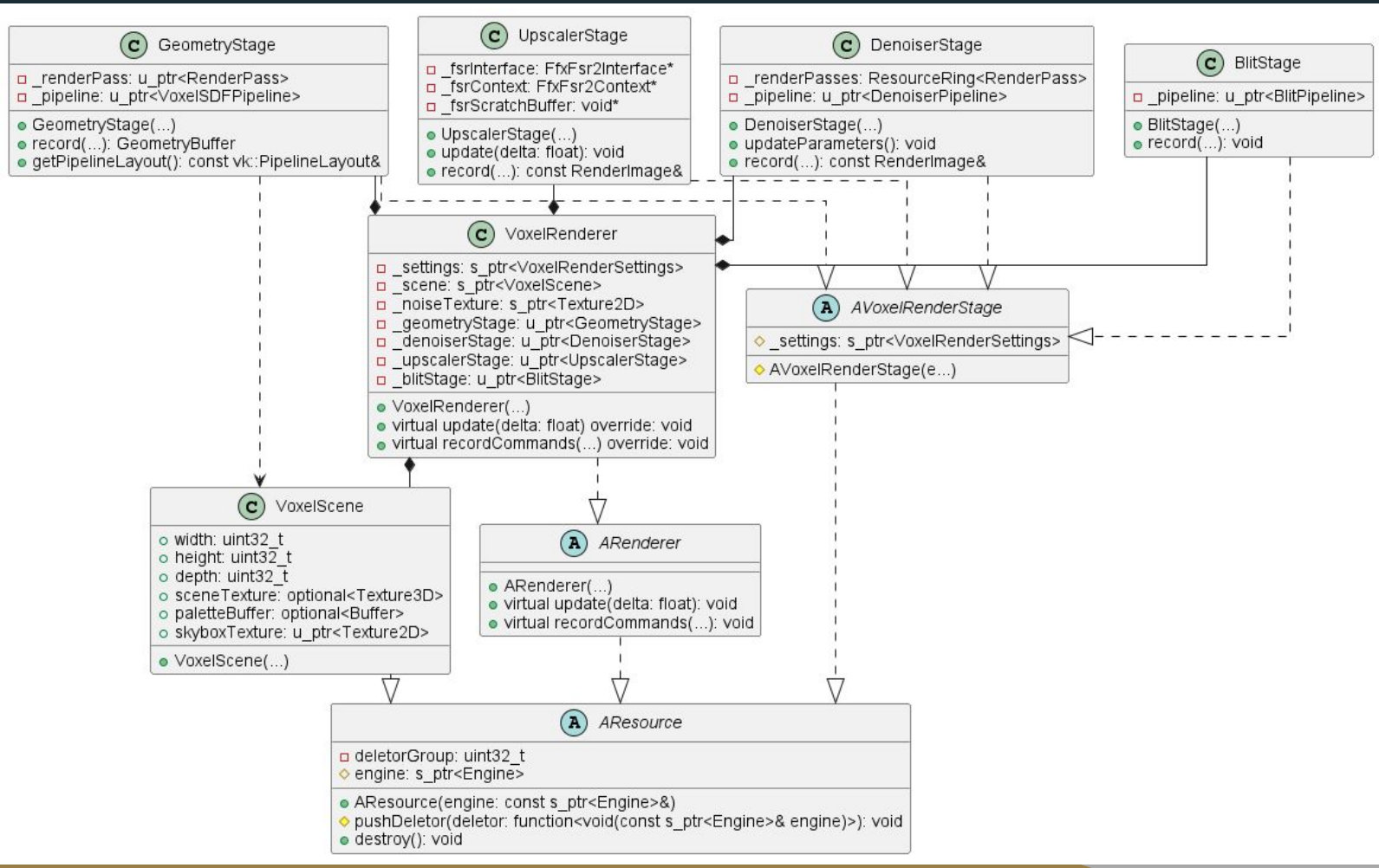
```
vk::DeviceSize imageSize = width * height * depth * pixelSize;

// 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
Buffer(...)
copyData(data: void*, size: size_t): void
Texture2D
width: uint32_t
height: uint32_t
channels: uint32_t
image: vk::Image
imageView: vk::ImageView
sampler: vk::Sampler
Texture2D(...)
Framebuffer
framebuffer: vk::Framebuffer
size: glm::uvec2
Framebuffer(...)
RenderPass
renderPass: vk::RenderPass
RenderPass(...)
recordBegin(...): void
APipeline
pipeline: vk::Pipeline
layout: vk::PipelineLayout
APipeline(...)
buildAll(): 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(): void
resetDestroy(): void
ShaderModule
_stage: vk::ShaderStageFlagBits
_shaderModule: vk::ShaderModule
ShaderModule(...)
RenderImage
width: uint32_t
height: uint32_t
image: vk::Image
format: vk::Format
imageView: vk::ImageView
sampler: vk::Sampler
allocation: VmaAllocation
RenderImage(...)
DescriptorSet
layout: vk::DescriptorSetLayout
sets: ResourceRing<vk::DescriptorSet>
DescriptorSet(...)
writeBuffer(...): void
writeImage(..): void
initBuffer(...): void
initImage(...): void
Texture3D
width: size_t
height: size_t
depth: size_t
image: vk::Image
imageView: vk::ImageView
sampler: vk::Sampler
Texture3D(...)

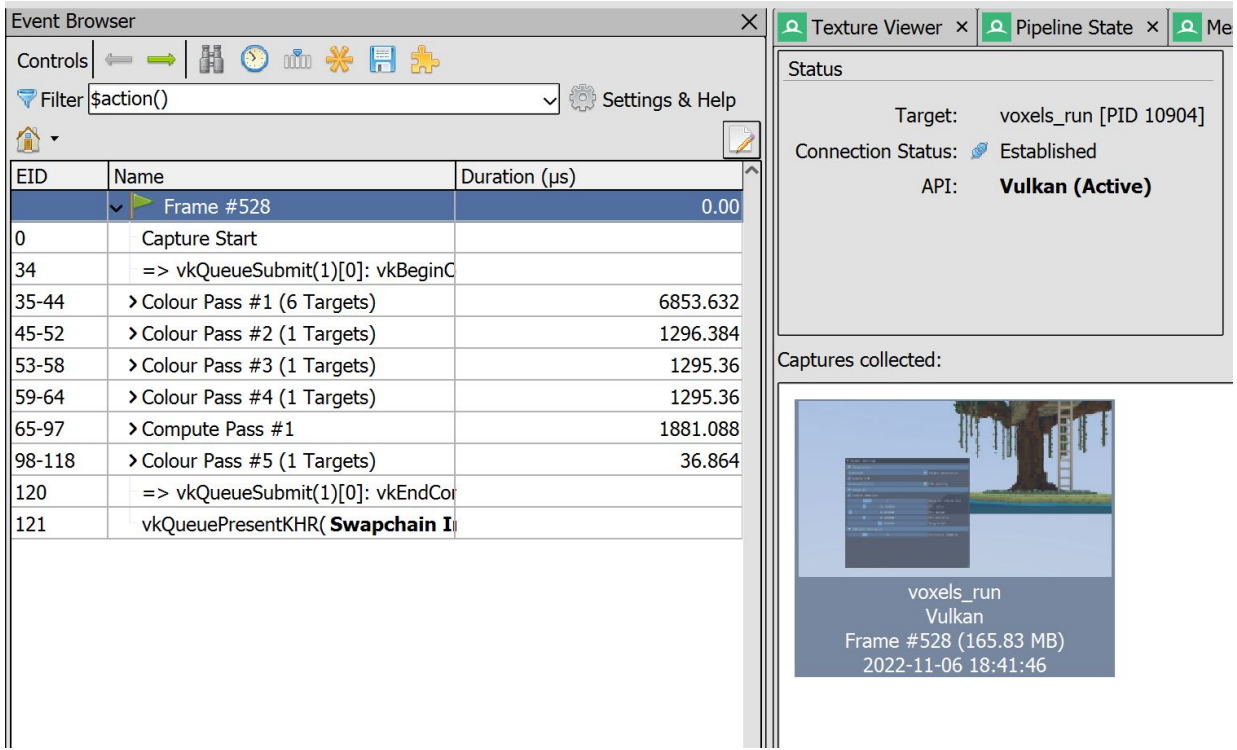

Geometry Params
Noise Texture
Scene 3D Texture
Geometry Pass
Denoiser Parameters
Normal
Color
Position
Motion
Mask
Depth
Denoiser Pass
Upscaler Parameters
Denoised Color
Upscaler Pass
Blit Parameters
UpscaledOutput
Blit Pass
Swapchain Image

Our VoxelRenderer organizes recording Vulkan commands

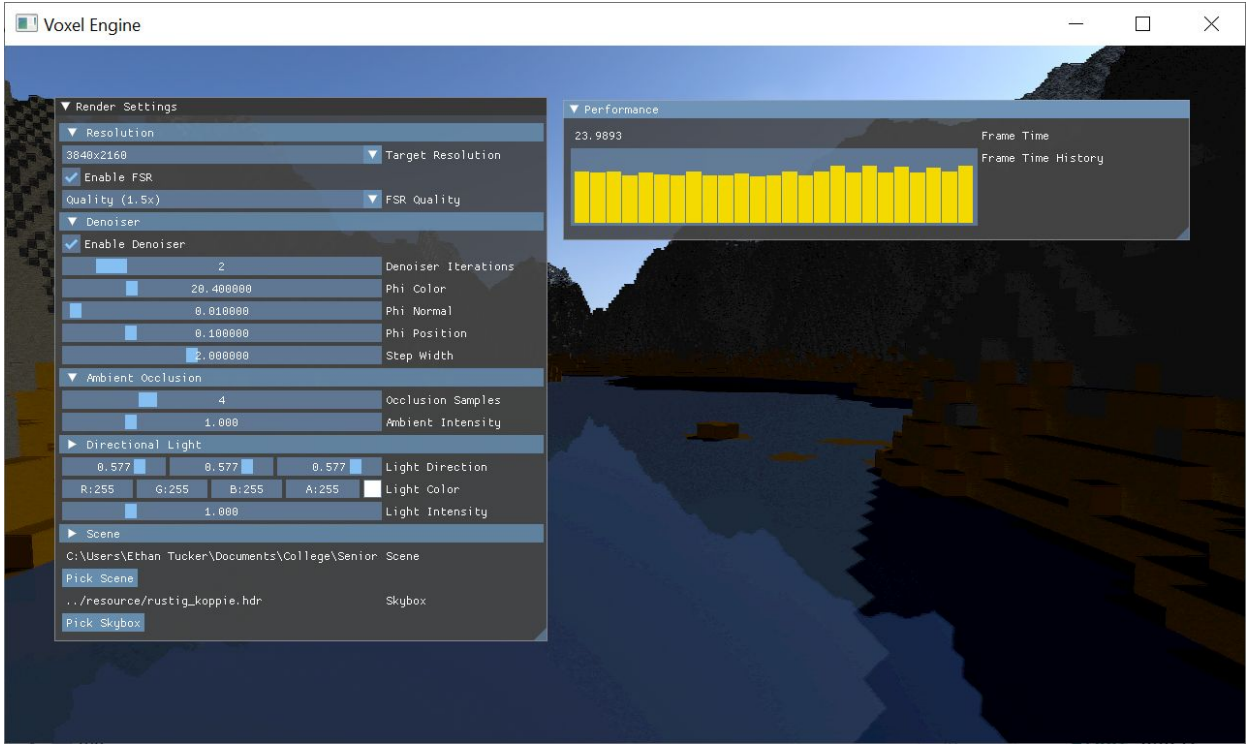


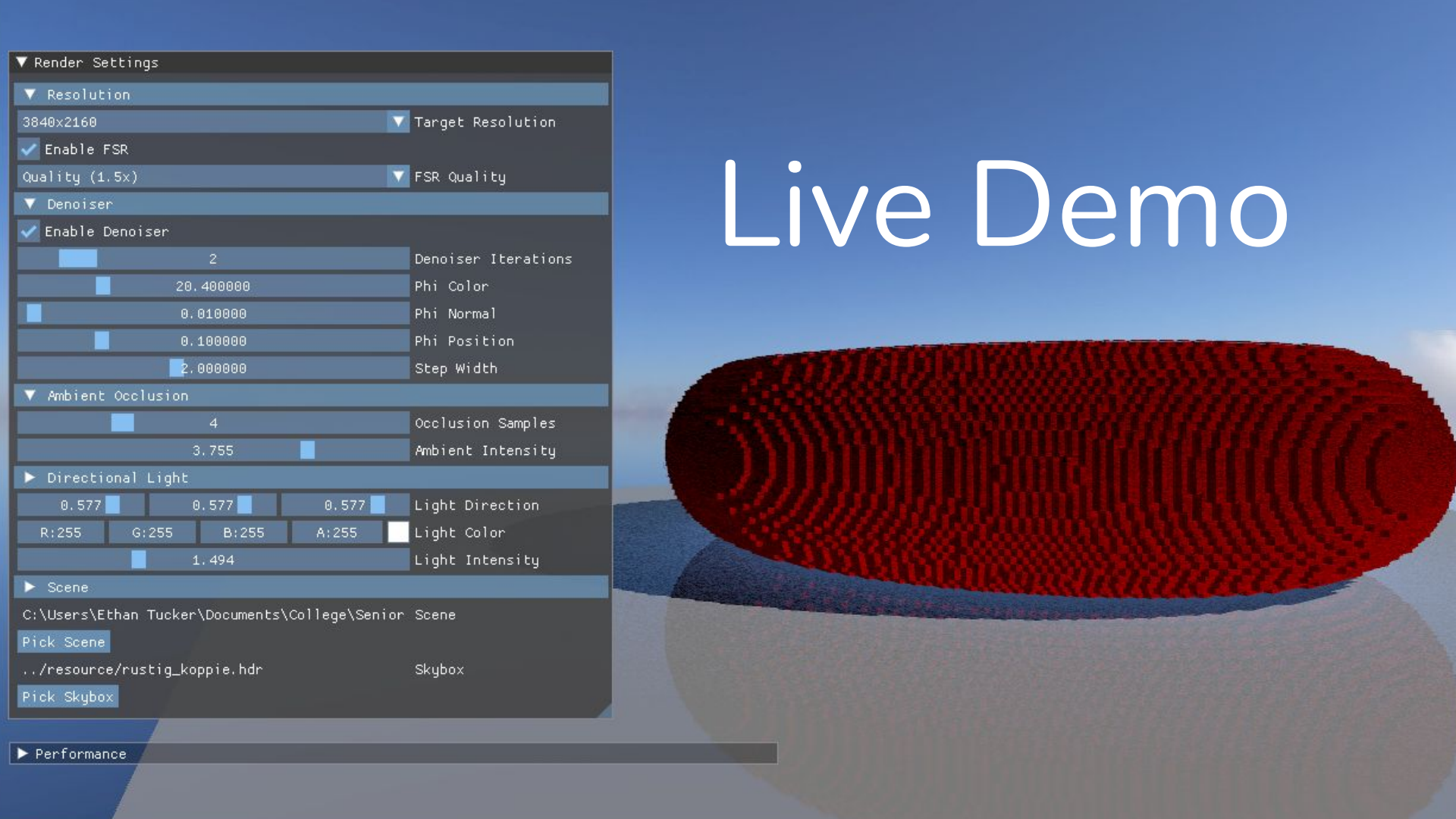

GeometryStage
- _renderPass: u_ptr<RenderPass>
- _pipeline: u_ptr<VoxelSDFPipeline>
+ GeometryStage(...)
+ record(...): GeometryBuffer
+ getPipelineLayout(): const vk::PipelineLayout&
UpscalerStage
- _fsrInterface: FfxFsr2Interface*
- _fsrContext: FfxFsr2Context*
- _fsrScratchBuffer: void*
+ UpscalerStage(...)
+ update(delta: float): void
+ record(...): const RenderImage&
DenoiserStage
- _renderPasses: ResourceRing<RenderPass>
- _pipeline: u_ptr<DenoiserPipeline>
+ DenoiserStage(...)
+ updateParameters(): void
+ record(...): const RenderImage&
BlitStage
- _pipeline: u_ptr<BlitPipeline>
+ BlitStage(...)
+ record(...): void
VoxelRenderer
- _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>
+ VoxelRenderer(...)
+ virtual update(delta: float) override: void
+ virtual recordCommands(...) override: void
AVoxelRenderStage
_settings: s_ptr<VoxelRenderSettings>
AVoxelRenderStage(e...)
VoxelScene
+ width: uint32_t
+ height: uint32_t
+ depth: uint32_t
+ sceneTexture: optional<Texture3D>
+ paletteBuffer: optional<Buffer>
+ skyboxTexture: u_ptr<Texture2D>
+ VoxelScene(...)
ARenderer
+ ARenderer(...)
+ virtual update(delta: float): void
+ virtual recordCommands(...): 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(): void

RenderDoc can measure performance



On my RTX 3070, performance is consistently within our 30 FPS target




Render Settings
Resolution
3840x2160
Target Resolution
Enable FSR
Quality (1.5x)
FSR Quality
Denoiser
Enable Denoiser
2
Denoiser Iterations
20.400000
Phi Color
0.010000
Phi Normal
0.100000
Phi Position
2.000000
Step Width
Ambient Occlusion
4
Occlusion Samples
3.755
Ambient Intensity
Directional Light
0.577
0.577
0.577
Light Direction
R:255
G:255
B:255
A:255
Light Color
1.494
Light Intensity
Scene
C:\Users\Ethan Tucker\Documents\College\Senior
Scene
Pick Scene
../resource/rustig_koppie.hdr
Skybox
Pick Skybox
Performance
Live Demo

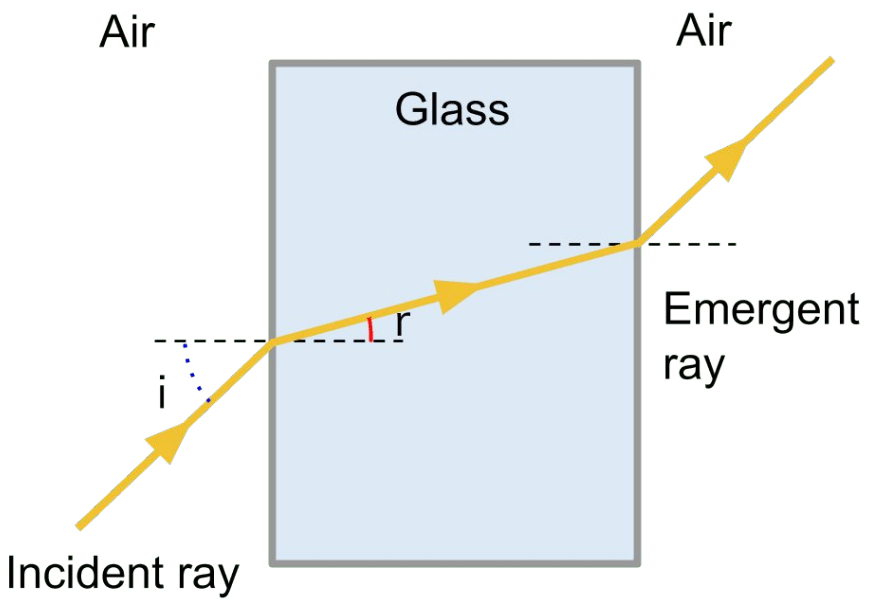
Future Work

Version 2.0 Ideas

- Transparent voxels and refraction
- Emissive materials
- Multiple voxel volumes
- Voxel animations

Other Ideas

- Integration into full game engine
- Octree Implementation



Questions?

Topics:

- Abstracting Vulkan
- Ray-tracing algorithm
- Shading algorithm
- Demo requests
- Overall architecture