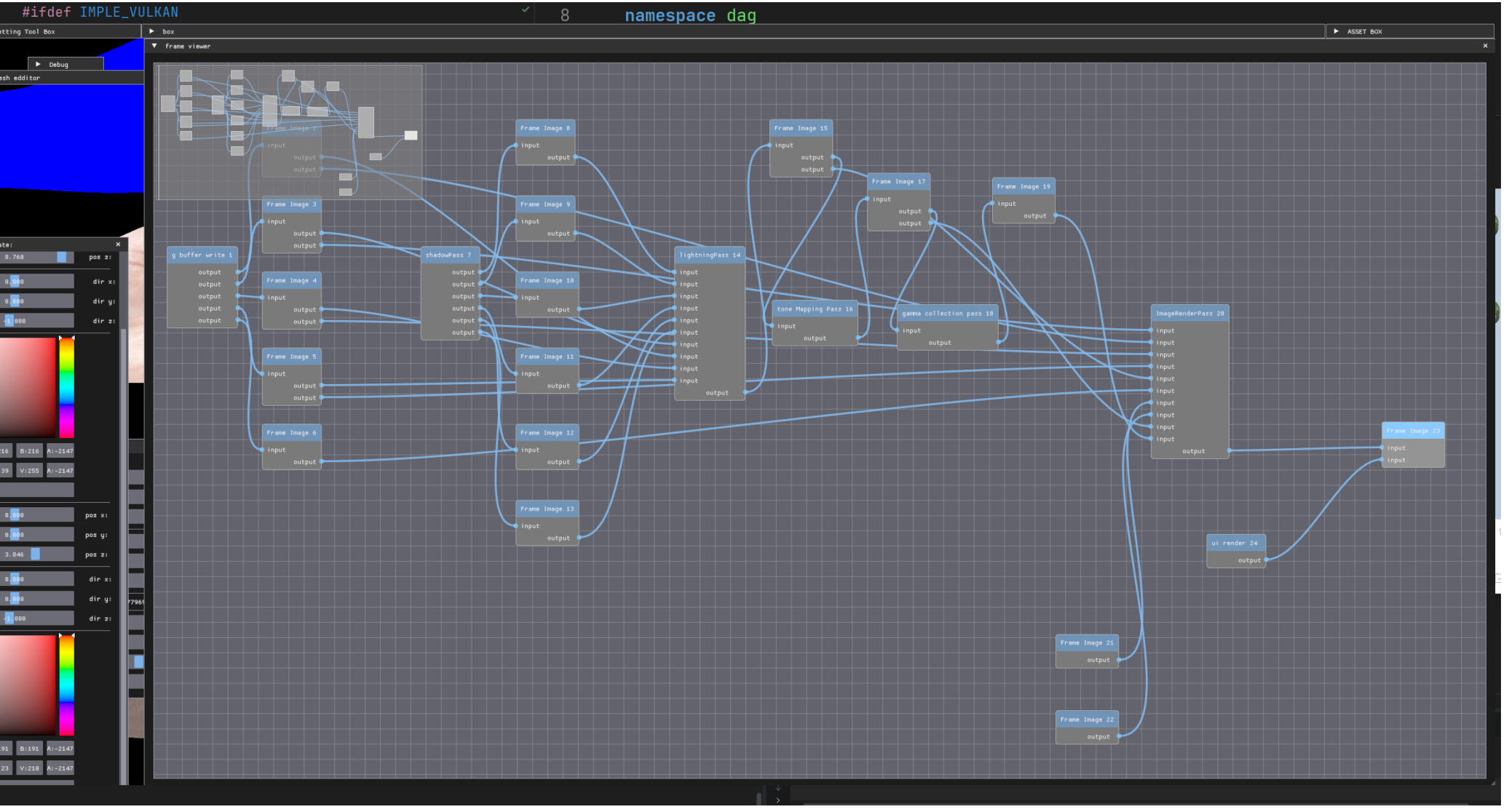
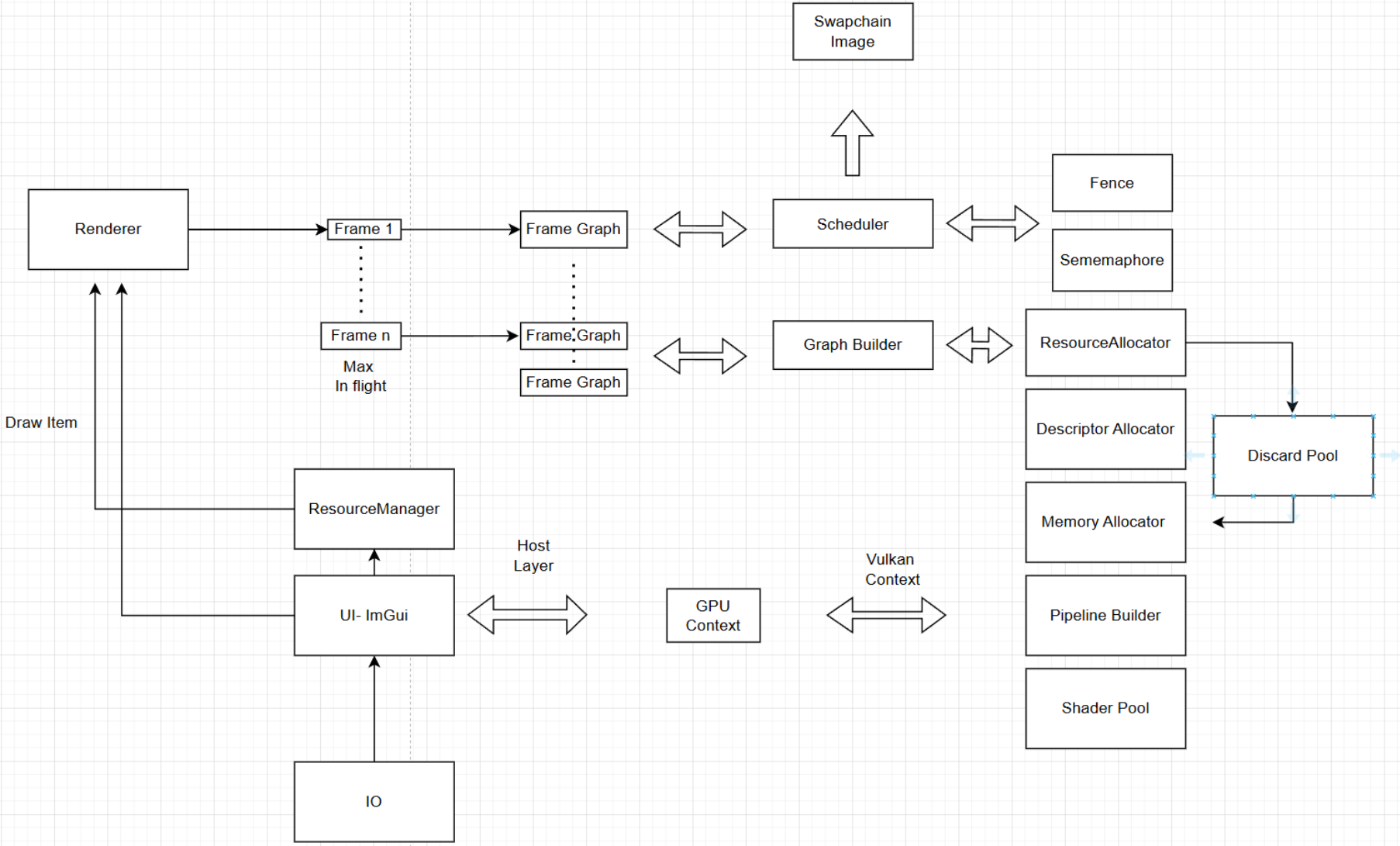
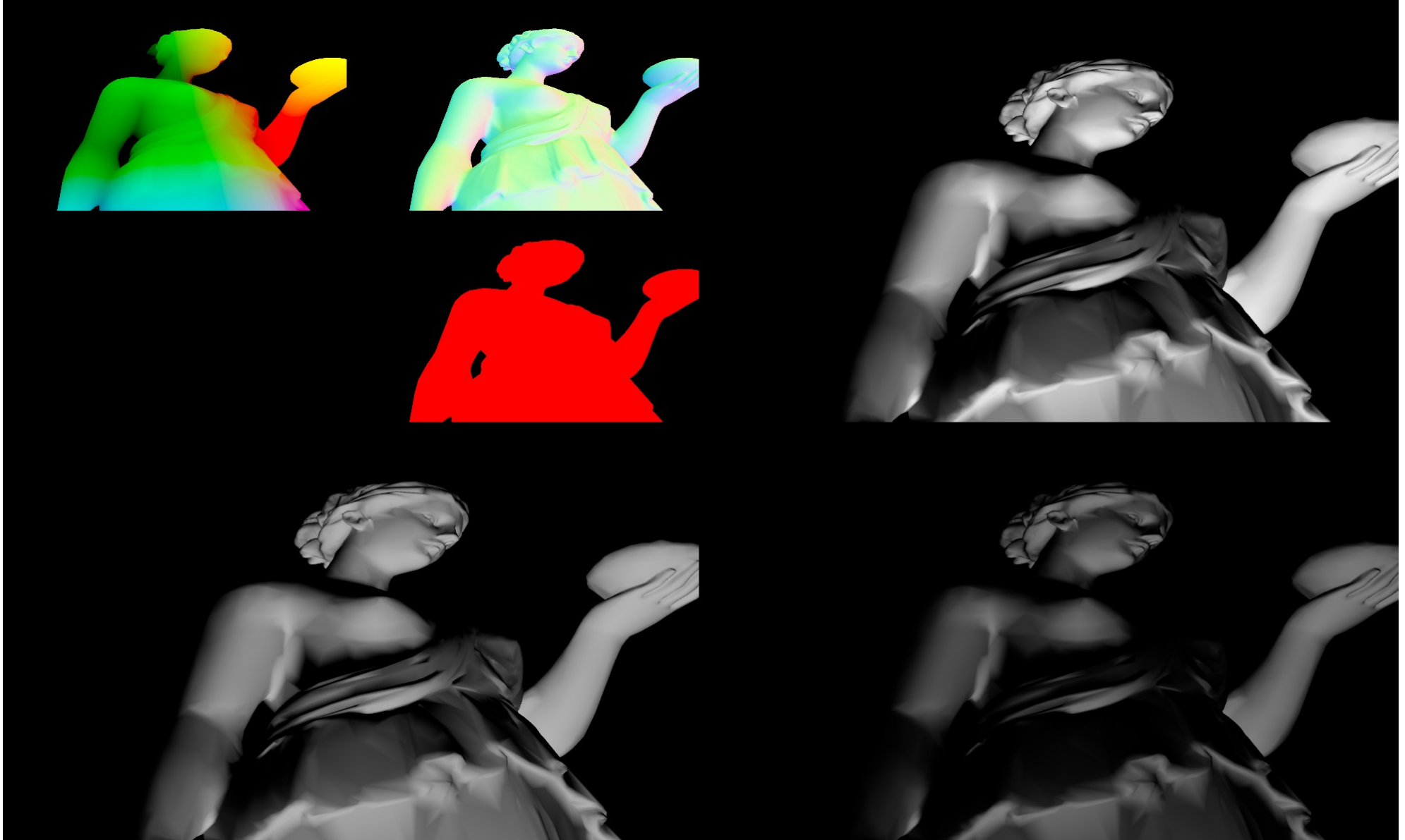
Rendering Engine

: 24101515 이재훈


light state:
pos z:
dir x:
dir y:
dir z:
G:229
B:229
A:-2147
S: 26
V:255
A:-2147
FFE5E500
pos x:
pos y:
3.609
pos z:
dir x:
dir y:
dir z:
G:249
B:249
A:-2147
S: 6
V:255
A:-2147
FFF9F900
Overlay
FPS: 898.1
Frame Time: 1.11 ms
Debug
main camera
light window
test Model

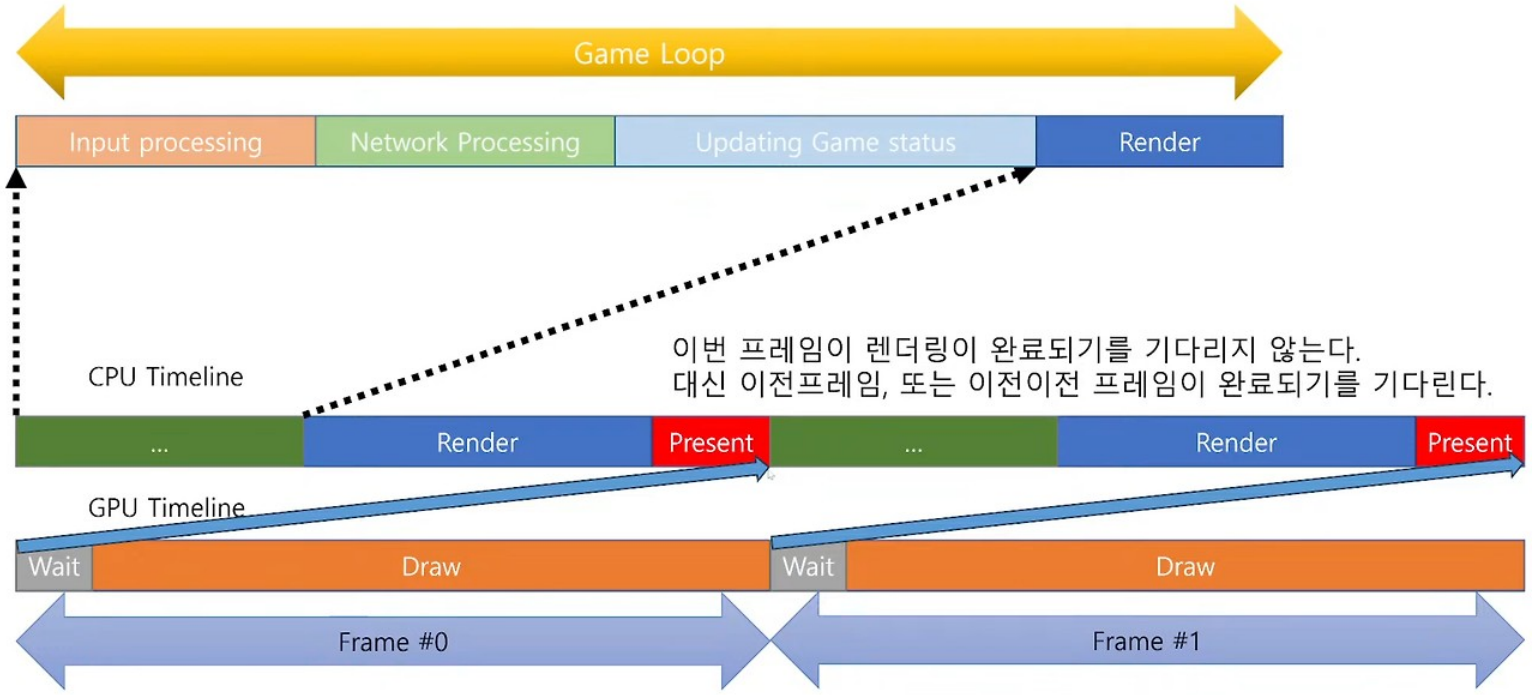

#ifdef IMPLE_VULKAN
8
namespace dag
box
ASSET BOX
frame viewer
Debug
Frame Image 2
Frame Image 3
Frame Image 4
Frame Image 5
Frame Image 6
Frame Image 8
Frame Image 9
Frame Image 10
Frame Image 11
Frame Image 12
Frame Image 13
Frame Image 15
Frame Image 17
Frame Image 19
Frame Image 21
Frame Image 22
Frame Image 23
g buffer write 1
shadowPass 7
lightningPass 14
tone Mapping Pass 16
gamma collection pass 18
ImageRenderPass 20
ui render 24
input
output


Swapchain Image
Renderer
Frame 1
Frame n
Max In flight
Frame Graph
Frame Graph
Frame Graph
Scheduler
Fence
Sememaphore
Graph Builder
ResourceAllocator
Descriptor Allocator
Discard Pool
Memory Allocator
Pipeline Builder
Shader Pool
Draw Item
ResourceManager
UI- ImGui
IO
Host Layer
GPU Context
Vulkan Context

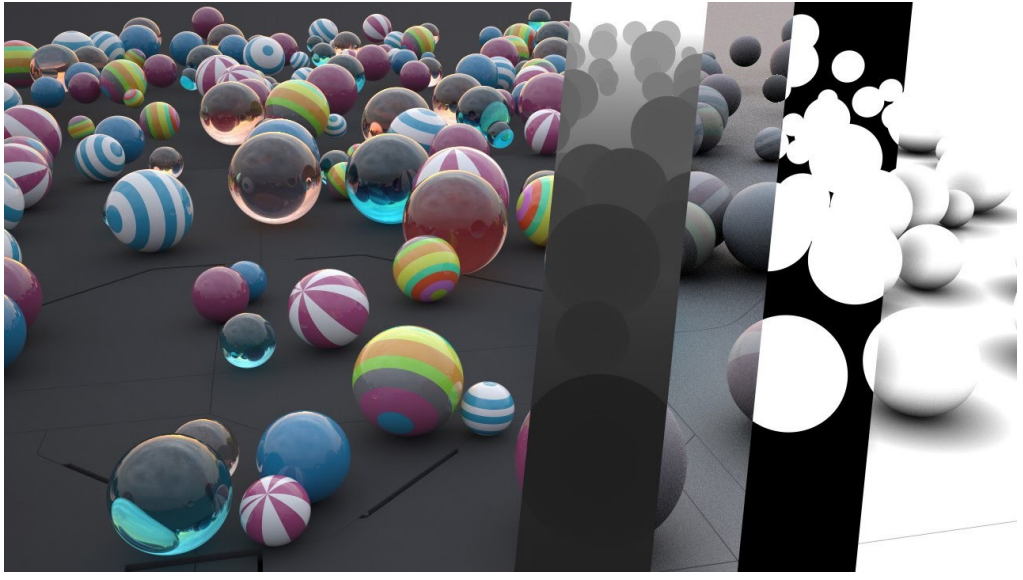


Modern Asyn Render

게임 루프 내에서의 timeline – 비동기식 중첩 렌더링

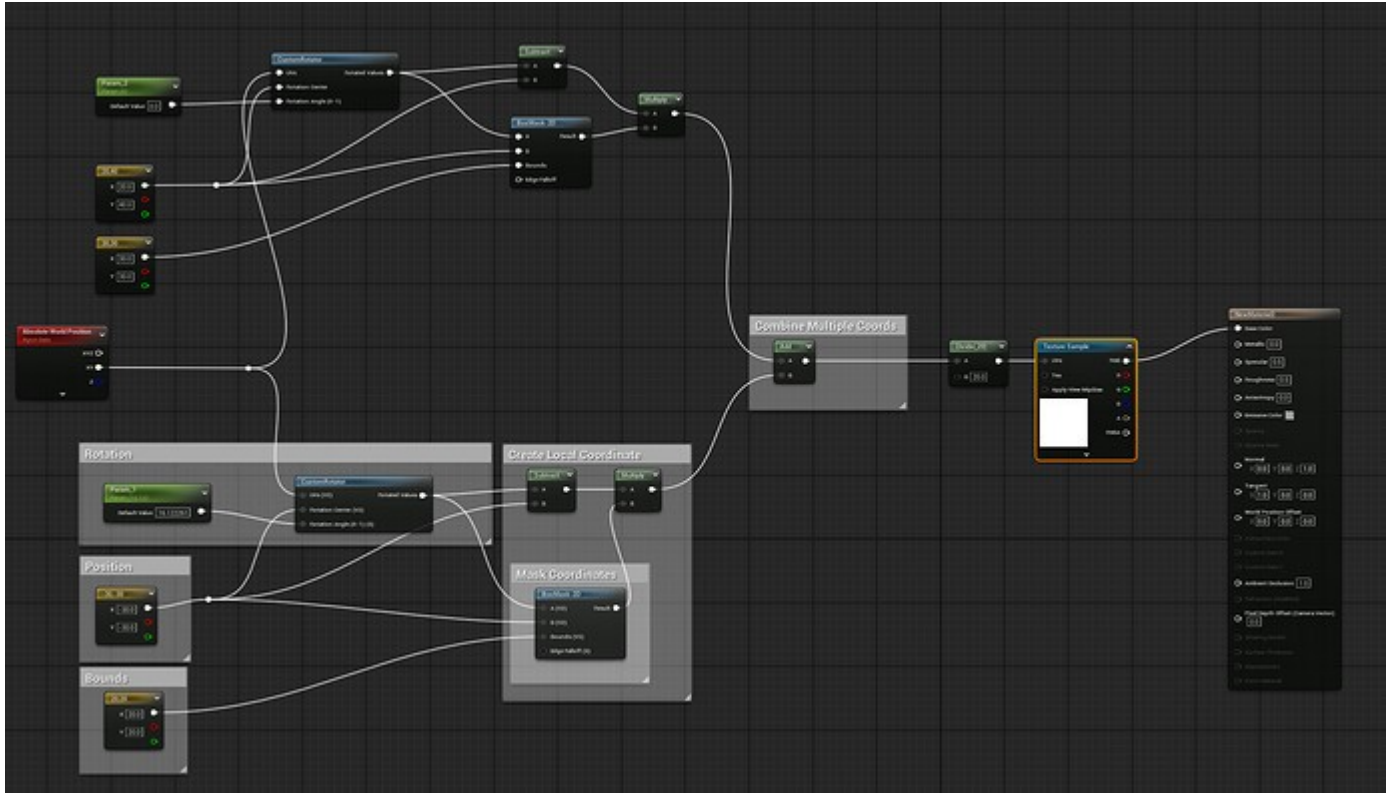


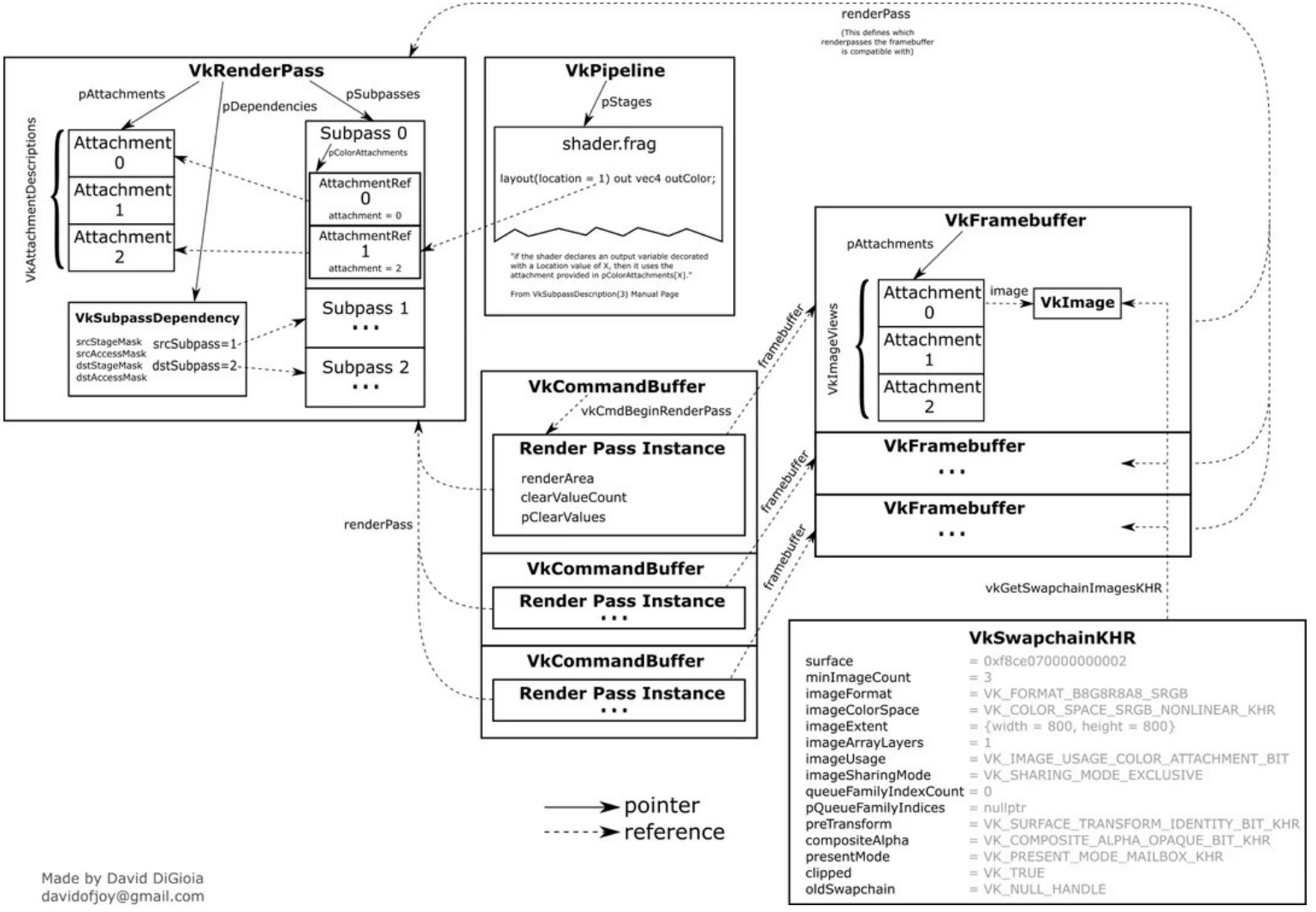
Multi Pass Rendering



Deffered Rendering

Rendering Graph





Made by David DiGioia
davidofjoy@gmail.com

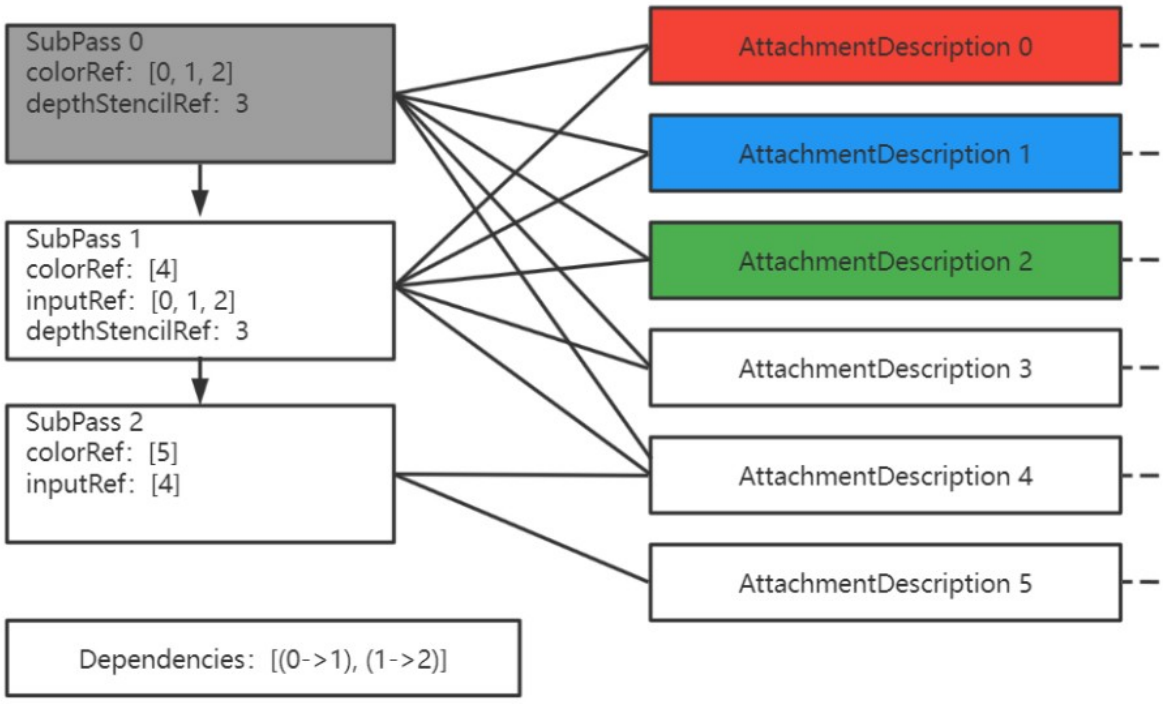
Dynamic Rendering :

- Dynamic Render:

  Very Simple:

  just Render!

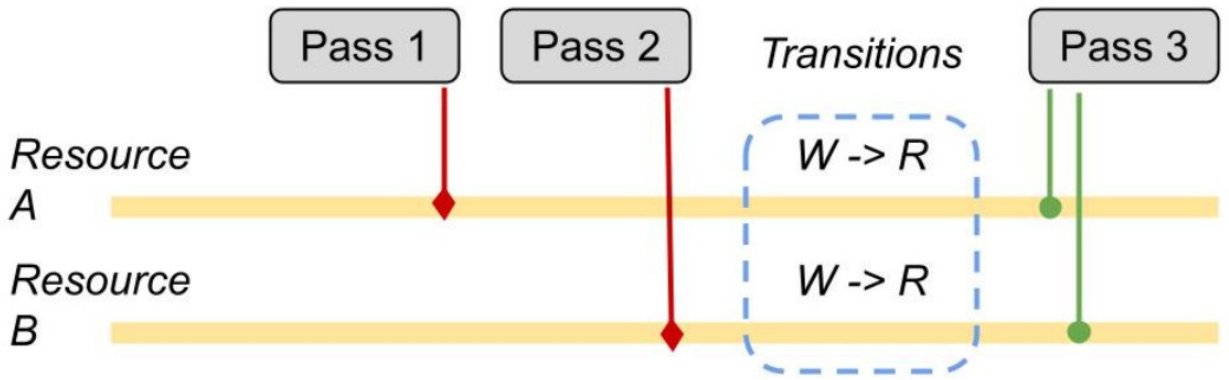
- Sync Need!!



Last writer Tracking:

- For resource Sync, insert barrer

  - W -> W



Resource Tracking

- For W -> R
- cashing resource State :

  insert Barrier

```
uint32_t currentPipeline__        = VK_PIPELINE_STAGE_TRANSFER_BIT;
uint32_t writePipeline__          = VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT;
VkAccessFlags writeAccessMask__   = VK_ACCESS_MEMORY_READ_BIT;
VkAccessFlags currentAccessMask__ = VK_ACCESS_NONE;
```

Lazy execute system

- seperate : execute and declare

```
class VkPass
{
  public:
  std::string name;
  VkPass();
  void clear();
  std::unordered_set<VkPass*> dependency__ = {};
  std::unordered_set<VkPass*> dependent__ = {};
  RenderPassType passType;
  std::vector<VkResource*> read__;
  std::vector<VkResource*> write__;
  std::function<void(VkCommandBuffer cmd)> execute = nullptr;
```

Lazy execute system

- seperate :

  execute

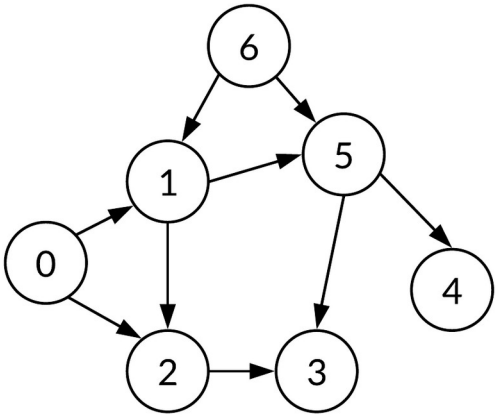
  and

  declare

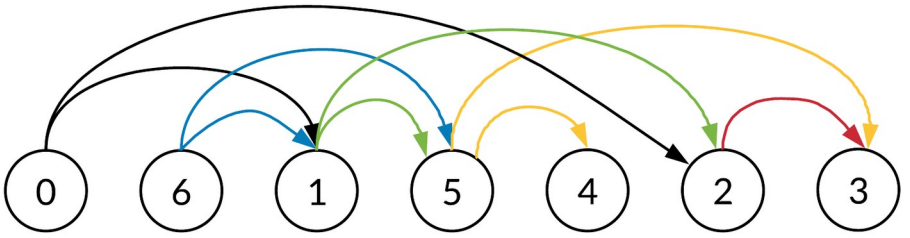
```
pass->read__.push_back(renderTargetFilm_->bloomingExtractAttachment_.get());
pass->write__.push_back(renderTargetFilm_->bloomingBlurAttachment_.get());
pass->execute = [this, pass](gpu::CommandBuffer cmd)
{
  gpu::cmdBindDescriptorSets(cmd,
                             VK_PIPELINE_BIND_POINT_GRAPHICS,
                             pipeline_->pipelineLayout_h,
                             0,
                             1,
                             &gpu::ctx__->pDescriptorAllocator->descriptorSets
                             [frameIndex_],
                             0,
                             nullptr);
  gpu::cmdBeginRendering(cmd, pass);
  renderTargetFilm_->updateFrameConstant();
  pushFrameConstant(cmd);
  gpu::cmdBindPipeline(cmd, VK_PIPELINE_BIND_POINT_GRAPHICS, pipeline_->bloomingBlurWritePipeline__);
  pipeline_->cmdSetPolygonMode(cmd, pipeline_->polygonMode);
  vkCmdSetDepthTestEnable(cmd, pipeline_->depthTest);
  gpu::cmdSetViewports(cmd,
                       0.0,
                       0.0,
                       (float)gpu::ctx__->pSwapChainContext->extent__.width,
                       (float)gpu::ctx__->pSwapChainContext->extent__.height
                      );
```

Multi Pass Rendering

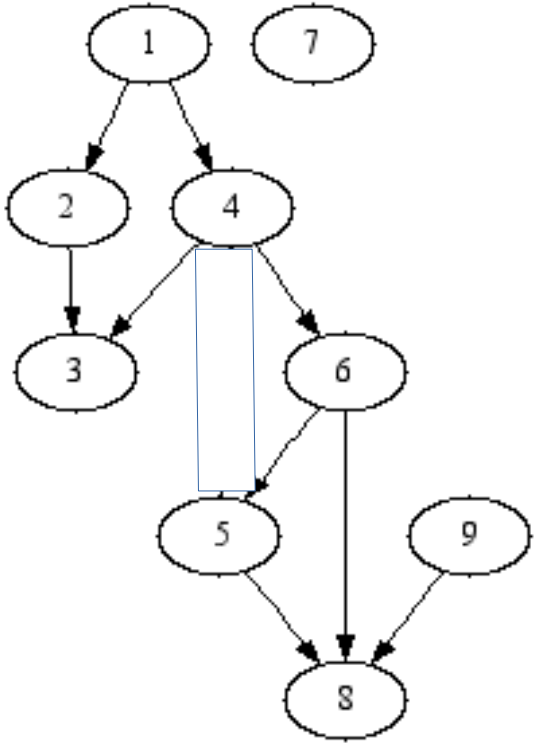
Unsorted graph



Topologically
sorted graph



Pass Level:



PASS level :
ordered
by
dependency

NV DLSS

- Tensor CORE !
- DLSS use

  AI Inference
- FRAME LOOP DEV UP!

- Using DLSS up scaling
- insert inferenced frame