DEVELOPER DAY

RenderPass/Subpass

Bill Licea-Kane, Qualcomm

KHRONOS™ GROUP

MONTRÉAL
APRIL 2018

April 2018

Montréal

Qualcomm

Vulkan RenderPass/Subpass

Bill Licea-Kane

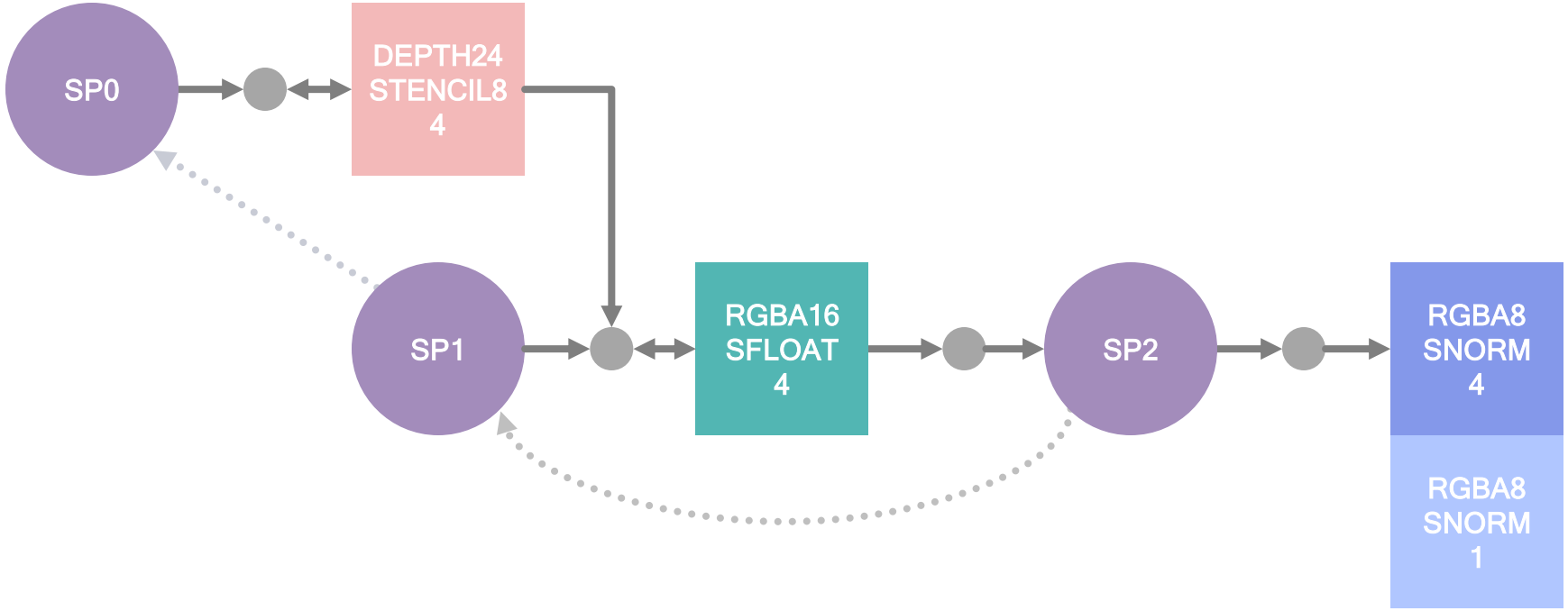
Engineer, Senior Staff
Qualcomm Innovation Center, Inc. (QuIC)

“A *render pass* represents a collection of attachments, subpasses, and dependencies between the subpasses, and describes how the attachments are used over the course of the subpasses. The use of a render pass in a command buffer is a *render pass instance*.”

“A *subpass* represents a phase of rendering that reads and writes a subset of the attachments in a render pass. Rendering commands are recorded into a particular subpass of a render pass instance.”

7.0 Render Pass

Vulkan® 1.1.74 - A Specification (with KHR extensions)


SP0
DEPTH24
STENCIL8
4
SP1
RGBA16
SFLOAT
4
SP2
RGBA8
SNORM
4
RGBA8
SNORM
1

Render Pass/Subpass

- API walkthrough/runthrough
- Example Render Pass/Subpass
- Known Limitations
- Dependencies, Self-dependices
- Recommedations

Overview

Create Render Pass Object

```
VKAPI_ATTR VkResult VKAPI_CALL vkCreateRenderPass(
    VkDevice                                    device,
    const VkRenderPassCreateInfo*               pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkRenderPass*                               pRenderPass);
```

Create Render Pass Object

Create Info

```
VKAPI_ATTR VkResult VKAPI_CALL vkCreateRenderPass(
    VkDevice                                    device,
    const VkRenderPassCreateInfo*               pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkRenderPass*                               pRenderPass);
```

Create Render Pass Object : Create Info

```
typedef struct VkRenderPassCreateInfo {
    VkStructureType                    sType;
    const void*                        pNext;
    VkRenderPassCreateFlags            flags;
    uint32_t                           attachmentCount;
    const VkAttachmentDescription*     pAttachments;
    uint32_t                           subpassCount;
    const VkSubpassDescription*        pSubpasses;
    uint32_t                           dependencyCount;
    const VkSubpassDependency*         pDependencies;
} VkRenderPassCreateInfo;
```

Create Render Pass Object : Create Info

Attachments

```
typedef struct VkRenderPassCreateInfo {
    VkStructureType                    sType;
    const void*                        pNext;
    VkRenderPassCreateFlags            flags;
    uint32_t                           attachmentCount;
    const VkAttachmentDescription*     pAttachments;
    uint32_t                           subpassCount;
    const VkSubpassDescription*        pSubpasses;
    uint32_t                           dependencyCount;
    const VkSubpassDependency*         pDependencies;
} VkRenderPassCreateInfo;
```

```
typedef struct VkAttachmentDescription {
    VkAttachmentDescriptionFlags    flags;
    VkFormat                        format;
    VkSampleCountFlagBits           samples;
    VkAttachmentLoadOp              loadOp;
    VkAttachmentStoreOp             storeOp;
    VkAttachmentLoadOp              stencilLoadOp;
    VkAttachmentStoreOp             stencilStoreOp;
    VkImageLayout                   initialLayout;
    VkImageLayout                   finalLayout;
} VkAttachmentDescription;
```

Create Render Pass Object : Attachments

```
typedef struct VkAttachmentDescription {
    VkAttachmentDescriptionFlags    flags;
    VkFormat                        format;
    VkSampleCountFlagBits           samples;
    VkAttachmentLoadOp              loadOp;
    VkAttachmentStoreOp             storeOp;
    VkAttachmentLoadOp              stencilLoadOp;
    VkAttachmentStoreOp             stencilStoreOp;
    VkImageLayout                   initialLayout;
    VkImageLayout                   finalLayout;
} VkAttachmentDescription;
```

Create Render Pass Object : Attachments

Flags, Format

```
typedef struct VkAttachmentDescription {
    VkAttachmentDescriptionFlags    flags;
    VkFormat                        format;
    VkSampleCountFlagBits           samples;
    VkAttachmentLoadOp              loadOp;
    VkAttachmentStoreOp             storeOp;
    VkAttachmentLoadOp              stencilLoadOp;
    VkAttachmentStoreOp             stencilStoreOp;
    VkImageLayout                   initialLayout;
    VkImageLayout                   finalLayout;
} VkAttachmentDescription;
```

```
typedef enum VkAttachmentDescriptionFlagBits {
    VK_ATTACHMENT_DESCRIPTION_MAY_ALIAS_BIT = 0x00000001
} VkAttachmentDescriptionFlagBits;

typedef enum VkFormat {
    VK_FORMAT_UNDEFINED = 0,
    VK_FORMAT_R4G4_UNORM_PACK8 = 1,
    VK_FORMAT_R4G4B4A4_UNORM_PACK16 = 2,
    VK_FORMAT_B4G4R4A4_UNORM_PACK16 = 3,
    VK_FORMAT_R5G6B5_UNORM_PACK16 = 4,
    VK_FORMAT_B5G6R5_UNORM_PACK16 = 5,
    VK_FORMAT_R5G5B5A1_UNORM_PACK16 = 6,
    VK_FORMAT_B5G5R5A1_UNORM_PACK16 = 7,
    VK_FORMAT_A1R5G5B5_UNORM_PACK16 = 8,
    VK_FORMAT_R8_UNORM = 9,
    VK_FORMAT_R8_SNORM = 10,
    VK_FORMAT_R8_USCALED = 11,
    VK_FORMAT_R8_SSCALED = 12,
    VK_FORMAT_R8_UINT = 13,
    VK_FORMAT_R8_SINT = 14,
    VK_FORMAT_R8_SRGB = 15,
    // ...
} VkFormat;
```

Create Render Pass Object : Attachments

Samples

```
typedef struct VkAttachmentDescription {
    VkAttachmentDescriptionFlags    flags;
    VkFormat                        format;
    VkSampleCountFlagBits           samples;
    VkAttachmentLoadOp              loadOp;
    VkAttachmentStoreOp             storeOp;
    VkAttachmentLoadOp              stencilLoadOp;
    VkAttachmentStoreOp             stencilStoreOp;
    VkImageLayout                   initialLayout;
    VkImageLayout                   finalLayout;
} VkAttachmentDescription;
```

```
typedef enum VkSampleCountFlagBits {
    VK_SAMPLE_COUNT_1_BIT = 0x00000001,
    VK_SAMPLE_COUNT_2_BIT = 0x00000002,
    VK_SAMPLE_COUNT_4_BIT = 0x00000004,
    VK_SAMPLE_COUNT_8_BIT = 0x00000008,
    VK_SAMPLE_COUNT_16_BIT = 0x00000010,
    VK_SAMPLE_COUNT_32_BIT = 0x00000020,
    VK_SAMPLE_COUNT_64_BIT = 0x00000040,
    VK_SAMPLE_COUNT_FLAG_BITS_MAX_ENUM = 0x7FFFFFFF
} VkSampleCountFlagBits;
```

Create Render Pass Object : Attachments

Load Op, Store Op, Stencil Load Op, Stencil Store Op, Initial Layout, Final Layout

```
typedef struct VkAttachmentDescription {
    VkAttachmentDescriptionFlags    flags;
    VkFormat                        format;
    VkSampleCountFlagBits           samples;
    VkAttachmentLoadOp              loadOp;
    VkAttachmentStoreOp             storeOp;
    VkAttachmentLoadOp              stencilLoadOp;
    VkAttachmentStoreOp             stencilStoreOp;
    VkImageLayout                   initialLayout;
    VkImageLayout                   finalLayout;
} VkAttachmentDescription;
```

```
typedef enum VkAttachmentLoadOp {
    VK_ATTACHMENT_LOAD_OP_LOAD = 0,
    VK_ATTACHMENT_LOAD_OP_CLEAR = 1,
    VK_ATTACHMENT_LOAD_OP_DONT_CARE = 2,
} VkAttachmentLoadOp;

typedef enum VkAttachmentStoreOp {
    VK_ATTACHMENT_STORE_OP_STORE = 0,
    VK_ATTACHMENT_STORE_OP_DONT_CARE = 1,
} VkAttachmentStoreOp;

typedef enum VkImageLayout {
    VK_IMAGE_LAYOUT_UNDEFINED = 0,
    VK_IMAGE_LAYOUT_GENERAL = 1,
    VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL = 2,
    VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL = 3,
    VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL = 4,
    VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL = 5,
    VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL = 6,
    VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL = 7,
    VK_IMAGE_LAYOUT_PREINITIALIZED = 8,
} VkImageLayout;
```

Create Render Pass Object : Create Info

Subpasses : Input Attachments, Resolve Attachments, Depth Stencil Attachments

```
typedef struct VkRenderPassCreateInfo {
    VkStructureType                    sType;
    const void*                        pNext;
    VkRenderPassCreateFlags            flags;
    uint32_t                           attachmentCount;
    const VkAttachmentDescription*     pAttachments;
    uint32_t                           subpassCount;
    const VkSubpassDescription*        pSubpasses;
    uint32_t                           dependencyCount;
    const VkSubpassDependency*         pDependencies;
} VkRenderPassCreateInfo;
```

```
typedef struct VkSubpassDescription {
    VkSubpassDescriptionFlags       flags;
    VkPipelineBindPoint             pipelineBindPoint;
    uint32_t                        inputAttachmentCount;
    const VkAttachmentReference*    pInputAttachments;
    uint32_t                        colorAttachmentCount;
    const VkAttachmentReference*    pColorAttachments;
    const VkAttachmentReference*    pResolveAttachments;
    const VkAttachmentReference*    pDepthStencilAttachment;
    uint32_t                        preserveAttachmentCount;
    const uint32_t*                 pPreserveAttachments;
} VkSubpassDescription;

typedef struct VkAttachmentReference {
    uint32_t         attachment;
    VkImageLayout    layout;
} VkAttachmentReference;
```

Create Render Pass Object : Create Info

Input Attachment Aspect Create Info : Aspect References

```
typedef struct VkRenderPassCreateInfo {
    VkStructureType                    sType;
    const void*                        pNext;
    VkRenderPassCreateFlags            flags;
    uint32_t                           attachmentCount;
    const VkAttachmentDescription*     pAttachments;
    uint32_t                           subpassCount;
    const VkSubpassDescription*        pSubpasses;
    uint32_t                           dependencyCount;
    const VkSubpassDependency*         pDependencies;
} VkRenderPassCreateInfo;
```

```
typedef struct VkRenderPassInputAttachmentAspectCreateInfo {
    VkStructureType                          sType;
    const void*                              pNext;
    uint32_t                                 aspectReferenceCount;
    const VkInputAttachmentAspectReference*  pAspectReferences;
} VkRenderPassInputAttachmentAspectCreateInfo;

typedef struct VkInputAttachmentAspectReference {
    uint32_t              subpass;
    uint32_t              inputAttachmentIndex;
    VkImageAspectFlags    aspectMask;
} VkInputAttachmentAspectReference;

typedef enum VkImageAspectFlagBits {
    VK_IMAGE_ASPECT_COLOR_BIT = 0x00000001,
    VK_IMAGE_ASPECT_DEPTH_BIT = 0x00000002,
    VK_IMAGE_ASPECT_STENCIL_BIT = 0x00000004,
    VK_IMAGE_ASPECT_METADATA_BIT = 0x00000008,
    VK_IMAGE_ASPECT_PLANE_0_BIT = 0x00000010,
    VK_IMAGE_ASPECT_PLANE_1_BIT = 0x00000020,
    VK_IMAGE_ASPECT_PLANE_2_BIT = 0x00000040,
} VkImageAspectFlagBits;
```

Create Render Pass Object : Create Info

Dependencies

```
typedef struct VkRenderPassCreateInfo {
    VkStructureType                    sType;
    const void*                        pNext;
    VkRenderPassCreateFlags            flags;
    uint32_t                           attachmentCount;
    const VkAttachmentDescription*     pAttachments;
    uint32_t                           subpassCount;
    const VkSubpassDescription*        pSubpasses;
    uint32_t                           dependencyCount;
    const VkSubpassDependency*         pDependencies;
} VkRenderPassCreateInfo;
```

```
typedef struct VkSubpassDependency {
    uint32_t                 srcSubpass;
    uint32_t                 dstSubpass;
    VkPipelineStageFlags     srcStageMask;
    VkPipelineStageFlags     dstStageMask;
    VkAccessFlags            srcAccessMask;
    VkAccessFlags            dstAccessMask;
    VkDependencyFlags        dependencyFlags;
} VkSubpassDependency;
```

Create Render Pass Object : Dependencies

```
typedef struct VkSubpassDependency {
    uint32_t                srcSubpass;
    uint32_t                dstSubpass;
    VkPipelineStageFlags    srcStageMask;
    VkPipelineStageFlags    dstStageMask;
    VkAccessFlags           srcAccessMask;
    VkAccessFlags           dstAccessMask;
    VkDependencyFlags       dependencyFlags;
} VkSubpassDependency;
```

Create Render Pass Object : Dependencies

Src Subpass, Dst Subpass

```
typedef struct VkSubpassDependency {
    uint32_t                    srcSubpass;
    uint32_t                    dstSubpass;
    VkPipelineStageFlags        srcStageMask;
    VkPipelineStageFlags        dstStageMask;
    VkAccessFlags               srcAccessMask;
    VkAccessFlags               dstAccessMask;
    VkDependencyFlags           dependencyFlags;
} VkSubpassDependency;
```

Create Render Pass Object : Dependencies

Src Stage Mask, Dst Stage Mask

```
typedef struct VkSubpassDependency {
    uint32_t                    srcSubpass;
    uint32_t                    dstSubpass;
    VkPipelineStageFlags        srcStageMask;
    VkPipelineStageFlags        dstStageMask;
    VkAccessFlags               srcAccessMask;
    VkAccessFlags               dstAccessMask;
    VkDependencyFlags           dependencyFlags;
} VkSubpassDependency;
```

```
typedef enum VkPipelineStageFlagBits {
    VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT = 0x00000001,
    VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT = 0x00000002,
    VK_PIPELINE_STAGE_VERTEX_INPUT_BIT = 0x00000004,
    VK_PIPELINE_STAGE_VERTEX_SHADER_BIT = 0x00000008,
    VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT = 0x00000010,
    VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT = 0x00000020,
    VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT = 0x00000040,
    VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT = 0x00000080,
    VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT = 0x00000100,
    VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT = 0x00000200,
    VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT = 0x00000400,
    VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT = 0x00000800,
    VK_PIPELINE_STAGE_TRANSFER_BIT = 0x00001000,
    VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT = 0x00002000,
    VK_PIPELINE_STAGE_HOST_BIT = 0x00004000,
    VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT = 0x00008000,
    VK_PIPELINE_STAGE_ALL_COMMANDS_BIT = 0x00010000,
} VkPipelineStageFlagBits;
```

Create Render Pass Object : Dependencies

Src Access Mask, Dst Access Mask, Dependency Flags

```
typedef struct VkSubpassDependency {
    uint32_t                    srcSubpass;
    uint32_t                    dstSubpass;
    VkPipelineStageFlags        srcStageMask;
    VkPipelineStageFlags        dstStageMask;
    VkAccessFlags               srcAccessMask;
    VkAccessFlags               dstAccessMask;
    VkDependencyFlags           dependencyFlags;
} VkSubpassDependency;
```

```
typedef enum VkAccessFlagBits {
    VK_ACCESS_INDIRECT_COMMAND_READ_BIT = 0x00000001,
    VK_ACCESS_INDEX_READ_BIT = 0x00000002,
    VK_ACCESS_VERTEX_ATTRIBUTE_READ_BIT = 0x00000004,
    VK_ACCESS_UNIFORM_READ_BIT = 0x00000008,
    VK_ACCESS_INPUT_ATTACHMENT_READ_BIT = 0x00000010,
    VK_ACCESS_SHADER_READ_BIT = 0x00000020,
    VK_ACCESS_SHADER_WRITE_BIT = 0x00000040,
    VK_ACCESS_COLOR_ATTACHMENT_READ_BIT = 0x00000080,
    VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT = 0x00000100,
    VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT = 0x00000200,
    VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT = 0x00000400,
    VK_ACCESS_TRANSFER_READ_BIT = 0x00000800,
    VK_ACCESS_TRANSFER_WRITE_BIT = 0x00001000,
    VK_ACCESS_HOST_READ_BIT = 0x00002000,
    VK_ACCESS_HOST_WRITE_BIT = 0x00004000,
    VK_ACCESS_MEMORY_READ_BIT = 0x00008000,
    VK_ACCESS_MEMORY_WRITE_BIT = 0x00010000,
} VkAccessFlagBits;

typedef enum VkDependencyFlagBits {
    VK_DEPENDENCY_BY_REGION_BIT = 0x00000001
} VkDependencyFlagBits;
```

“Framebuffers represent a collection of specific memory attachments that a render pass instance uses.”

7.3 Framebuffers

Vulkan® 1.1.74 - A Specification (with KHR extensions)

Create Framebuffer Object

```
VKAPI_ATTR VkResult VKAPI_CALL vkCreateFramebuffer(
    VkDevice                                    device,
    const VkFramebufferCreateInfo*              pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkFramebuffer*                              pFramebuffer);
```

Create Framebuffer Object

Create Info

```
VKAPI_ATTR VkResult VKAPI_CALL vkCreateFramebuffer(
    VkDevice                                    device,
    const VkFramebufferCreateInfo*              pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkFramebuffer*                              pFramebuffer);
```

Create Framebuffer Object : Create Info

Render Pass, Attachments, Width, Height, Layers

```
typedef struct VkFramebufferCreateInfo {
    VkStructureType             sType;
    const void*                 pNext;
    VkFramebufferCreateFlags    flags;
    VkRenderPass                renderPass;
    uint32_t                    attachmentCount;
    const VkImageView*          pAttachments;
    uint32_t                    width;
    uint32_t                    height;
    uint32_t                    layers;
} VkFramebufferCreateInfo;
```

Create Image View Object

```
VKAPI_ATTR VkResult VKAPI_CALL vkCreateImageView(
    VkDevice                                    device,
    const VkImageViewCreateInfo*                pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkImageView*                                pView);
```

Create Image View Object

Create Info

```
VKAPI_ATTR VkResult VKAPI_CALL vkCreateImageView(
    VkDevice                                    device,
    const VkImageViewCreateInfo*                pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkImageView*                                pView);
```

Create Image View Object : Create Info

```
typedef struct VkImageViewCreateInfo {
    VkStructureType             sType;
    const void*                 pNext;
    VkImageViewCreateFlags      flags;
    VkImage                     image;
    VkImageViewType             viewType;
    VkFormat                    format;
    VkComponentMapping          components;
    VkImageSubresourceRange     subresourceRange;
} VkImageViewCreateInfo;
```

Create Image View Object : Create Info

Image, View Type

```
typedef struct VkImageViewCreateInfo {
    VkStructureType             sType;
    const void*                 pNext;
    VkImageViewCreateFlags      flags;
    VkImage                     image;
    VkImageViewType             viewType;
    VkFormat                    format;
    VkComponentMapping          components;
    VkImageSubresourceRange     subresourceRange;
} VkImageViewCreateInfo;
```

```
typedef enum VkImageViewType {
    VK_IMAGE_VIEW_TYPE_1D = 0,
    VK_IMAGE_VIEW_TYPE_2D = 1,
    VK_IMAGE_VIEW_TYPE_3D = 2,
    VK_IMAGE_VIEW_TYPE_CUBE = 3,
    VK_IMAGE_VIEW_TYPE_1D_ARRAY = 4,
    VK_IMAGE_VIEW_TYPE_2D_ARRAY = 5,
    VK_IMAGE_VIEW_TYPE_CUBE_ARRAY = 6,
} VkImageViewType;
```

Create Image View Object : Create Info

Components

```
typedef struct VkImageViewCreateInfo {
    VkStructureType             sType;
    const void*                 pNext;
    VkImageViewCreateFlags      flags;
    VkImage                     image;
    VkImageViewType             viewType;
    VkFormat                    format;
    VkComponentMapping          components;
    VkImageSubresourceRange     subresourceRange;
} VkImageViewCreateInfo;
```

```
typedef struct VkComponentMapping {
    VkComponentSwizzle    r;
    VkComponentSwizzle    g;
    VkComponentSwizzle    b;
    VkComponentSwizzle    a;
} VkComponentMapping;

typedef enum VkComponentSwizzle {
    VK_COMPONENT_SWIZZLE_IDENTITY = 0,
    VK_COMPONENT_SWIZZLE_ZERO = 1,
    VK_COMPONENT_SWIZZLE_ONE = 2,
    VK_COMPONENT_SWIZZLE_R = 3,
    VK_COMPONENT_SWIZZLE_G = 4,
    VK_COMPONENT_SWIZZLE_B = 5,
    VK_COMPONENT_SWIZZLE_A = 6,
} VkComponentSwizzle;
```

Create Image View Object : Create Info

Subresource Range

```
typedef struct VkImageViewCreateInfo {
    VkStructureType             sType;
    const void*                 pNext;
    VkImageViewCreateFlags      flags;
    VkImage                     image;
    VkImageViewType             viewType;
    VkFormat                    format;
    VkComponentMapping          components;
    VkImageSubresourceRange     subresourceRange;
} VkImageViewCreateInfo;
```

```
typedef struct VkImageSubresourceRange {
    VkImageAspectFlags    aspectMask;
    uint32_t              baseMipLevel;
    uint32_t              levelCount;
    uint32_t              baseArrayLayer;
    uint32_t              layerCount;
} VkImageSubresourceRange;
```

Create Image Object

```
VKAPI_ATTR VkResult VKAPI_CALL vkCreateImage(
    VkDevice                                    device,
    const VkImageCreateInfo*                    pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkImage*                                    pImage);
```

Create Image Object

Create Info

```
VKAPI_ATTR VkResult VKAPI_CALL vkCreateImage(
    VkDevice                                    device,
    const VkImageCreateInfo*                    pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkImage*                                    pImage);
```

Create Image Object : Create Info

```
typedef struct VkImageCreateInfo {
    VkStructureType          sType;
    const void*              pNext;
    VkImageCreateFlags       flags;
    VkImageType              imageType;
    VkFormat                 format;
    VkExtent3D               extent;
    uint32_t                 mipLevels;
    uint32_t                 arrayLayers;
    VkSampleCountFlagBits    samples;
    VkImageTiling            tiling;
    VkImageUsageFlags        usage;
    VkSharingMode            sharingMode;
    uint32_t                 queueFamilyIndexCount;
    const uint32_t*          pQueueFamilyIndices;
    VkImageLayout            initialLayout;
} VkImageCreateInfo;
```

Create Image Object : Create Info

Flags, Format

```vulkan
typedef struct VkImageCreateInfo {
    VkStructureType          sType;
    const void*              pNext;
    VkImageCreateFlags       flags;
    VkImageType              imageType;
    VkFormat                 format;
    VkExtent3D               extent;
    uint32_t                 mipLevels;
    uint32_t                 arrayLayers;
    VkSampleCountFlagBits    samples;
    VkImageTiling            tiling;
    VkImageUsageFlags        usage;
    VkSharingMode            sharingMode;
    uint32_t                 queueFamilyIndexCount;
    const uint32_t*          pQueueFamilyIndices;
    VkImageLayout            initialLayout;
} VkImageCreateInfo;
```

```vulkan
typedef enum VkImageCreateFlagBits {
    VK_IMAGE_CREATE_SPARSE_BINDING_BIT = 0x00000001,
    VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT = 0x00000002,
    VK_IMAGE_CREATE_SPARSE_ALIASED_BIT = 0x00000004,
    VK_IMAGE_CREATE_MUTABLE_FORMAT_BIT = 0x00000008,
    VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT = 0x00000010,
    VK_IMAGE_CREATE_2D_ARRAY_COMPATIBLE_BIT = 0x00000020,
    VK_IMAGE_CREATE_SPLIT_INSTANCE_BIND_REGIONS_BIT = 0x00000040,
    VK_IMAGE_CREATE_BLOCK_TEXEL_VIEW_COMPATIBLE_BIT = 0x00000080,
    VK_IMAGE_CREATE_EXTENDED_USAGE_BIT = 0x00000100,
    VK_IMAGE_CREATE_DISJOINT_BIT = 0x00000200,
    VK_IMAGE_CREATE_ALIAS_BIT = 0x00000400,
    VK_IMAGE_CREATE_PROTECTED_BIT = 0x00000800,
} VkImageCreateFlagBits;

typedef enum VkImageType {
    VK_IMAGE_TYPE_1D = 0,
    VK_IMAGE_TYPE_2D = 1,
    VK_IMAGE_TYPE_3D = 2,
} VkImageType;
```

Create Image Object : Create Info

Extent, MipLevels, Array Layers, Samples, Tiling

```vulkan
typedef struct VkImageCreateInfo {
    VkStructureType          sType;
    const void*              pNext;
    VkImageCreateFlags       flags;
    VkImageType              imageType;
    VkFormat                 format;
    VkExtent3D               extent;
    uint32_t                 mipLevels;
    uint32_t                 arrayLayers;
    VkSampleCountFlagBits    samples;
    VkImageTiling            tiling;
    VkImageUsageFlags        usage;
    VkSharingMode            sharingMode;
    uint32_t                 queueFamilyIndexCount;
    const uint32_t*          pQueueFamilyIndices;
    VkImageLayout            initialLayout;
} VkImageCreateInfo;
```

```vulkan
typedef struct VkExtent3D {
    uint32_t    width;
    uint32_t    height;
    uint32_t    depth;
} VkExtent3D;

typedef enum VkImageTiling {
    VK_IMAGE_TILING_OPTIMAL = 0,
    VK_IMAGE_TILING_LINEAR = 1,
} VkImageTiling;
```

Create Image Object : Create Info

Usage, Sharing Mode, Queue Family Indices, Initial Layout

```
typedef struct VkImageCreateInfo {
    VkStructureType          sType;
    const void*              pNext;
    VkImageCreateFlags       flags;
    VkImageType              imageType;
    VkFormat                 format;
    VkExtent3D               extent;
    uint32_t                 mipLevels;
    uint32_t                 arrayLayers;
    VkSampleCountFlagBits    samples;
    VkImageTiling            tiling;
    VkImageUsageFlags        usage;
    VkSharingMode            sharingMode;
    uint32_t                 queueFamilyIndexCount;
    const uint32_t*          pQueueFamilyIndices;
    VkImageLayout            initialLayout;
} VkImageCreateInfo;
```

```
typedef enum VkImageUsageFlagBits {
    VK_IMAGE_USAGE_TRANSFER_SRC_BIT = 0x00000001,
    VK_IMAGE_USAGE_TRANSFER_DST_BIT = 0x00000002,
    VK_IMAGE_USAGE_SAMPLED_BIT = 0x00000004,
    VK_IMAGE_USAGE_STORAGE_BIT = 0x00000008,
    VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT = 0x00000010,
    VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT = 0x00000020,
    VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT = 0x00000040,
    VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT = 0x00000080,
} VkImageUsageFlagBits;

typedef enum VkSharingMode {
    VK_SHARING_MODE_EXCLUSIVE = 0,
    VK_SHARING_MODE_CONCURRENT = 1,
} VkSharingMode;
```

Begin Render Pass

```
VKAPI_ATTR void VKAPI_CALL vkCmdBeginRenderPass(
    VkCommandBuffer                     commandBuffer,
    const VkRenderPassBeginInfo*        pRenderPassBegin,
    VkSubpassContents                   contents);
```

Begin Render Pass

```
VKAPI_ATTR void VKAPI_CALL vkCmdBeginRenderPass(
    VkCommandBuffer                    commandBuffer,
    const VkRenderPassBeginInfo*       pRenderPassBegin,
    VkSubpassContents                  contents);
```

Begin Render Pass

Render Pass Begin

```
VKAPI_ATTR void VKAPI_CALL vkCmdBeginRenderPass(
    VkCommandBuffer                 commandBuffer,
    const VkRenderPassBeginInfo*    pRenderPassBegin,
    VkSubpassContents               contents);
```

```
typedef struct VkRenderPassBeginInfo {
    VkStructureType         sType;
    const void*             pNext;
    VkRenderPass            renderPass;
    VkFramebuffer           framebuffer;
    VkRect2D                renderArea;
    uint32_t                clearValueCount;
    const VkClearValue*     pClearValues;
} VkRenderPassBeginInfo;
```

Begin Render Pass : Render Pass Begin

```
typedef struct VkRenderPassBeginInfo {
    VkStructureType         sType;
    const void*             pNext;
    VkRenderPass            renderPass;
    VkFramebuffer           framebuffer;
    VkRect2D                renderArea;
    uint32_t                clearValueCount;
    const VkClearValue*     pClearValues;
} VkRenderPassBeginInfo;
```

Begin Render Pass : Render Pass Begin

Render Pass, Framebuffer, Render Area

```
typedef struct VkRenderPassBeginInfo {
    VkStructureType         sType;
    const void*             pNext;
    VkRenderPass            renderPass;
    VkFramebuffer           framebuffer;
    VkRect2D                renderArea;
    uint32_t                clearValueCount;
    const VkClearValue*     pClearValues;
} VkRenderPassBeginInfo;
```

```
typedef struct VkRect2D {
    VkOffset2D    offset;
    VkExtent2D    extent;
} VkRect2D;

typedef struct VkOffset2D {
    int32_t    x;
    int32_t    y;
} VkOffset2D;

typedef struct VkExtent2D {
    uint32_t    width;
    uint32_t    height;
} VkExtent2D;
```

Begin Render Pass : Render Pass Begin

Clear Values

```
typedef struct VkRenderPassBeginInfo {
    VkStructureType         sType;
    const void*             pNext;
    VkRenderPass            renderPass;
    VkFramebuffer           framebuffer;
    VkRect2D                renderArea;
    uint32_t                clearValueCount;
    const VkClearValue*     pClearValues;
} VkRenderPassBeginInfo;
```

```
VkRect2D; typedef union VkClearValue {
    VkClearColorValue           color;
    VkClearDepthStencilValue    depthStencil;
} VkClearValue;

typedef union VkClearColorValue {
    float       float32[4];
    int32_t     int32[4];
    uint32_t    uint32[4];
} VkClearColorValue;

typedef struct VkClearDepthStencilValue {
    float       depth;
    uint32_t    stencil;
} VkClearDepthStencilValue;
```

Begin Render Pass

```
VKAPI_ATTR void VKAPI_CALL vkCmdBeginRenderPass(
    VkCommandBuffer                     commandBuffer,
    const VkRenderPassBeginInfo*        pRenderPassBegin,
    VkSubpassContents                   contents);
```

Begin Render Pass

Contents

```
VKAPI_ATTR void VKAPI_CALL vkCmdBeginRenderPass(
    VkCommandBuffer                     commandBuffer,
    const VkRenderPassBeginInfo*        pRenderPassBegin,
    VkSubpassContents                   contents);
```

```
typedef enum VkSubpassContents {
    VK_SUBPASS_CONTENTS_INLINE = 0,
    VK_SUBPASS_CONTENTS_SECONDARY_COMMAND_BUFFERS = 1,
} VkSubpassContents;
```

Next Subpass

```
VKAPI_ATTR void VKAPI_CALL vkCmdNextSubpass(
    VkCommandBuffer                 commandBuffer,
    VkSubpassContents               contents);
```

“The subpasses in a render pass all render to the same dimensions, and fragments for pixel (x,y,layer) in one subpass **can** only read attachment contents written by previous subpasses at that same (x,y,layer) location.”

7.0 Render Pass

Vulkan® 1.1.74 - A Specification (with KHR extensions)

“So tonight I’m gonna render like it’s nineteen ninety-nine”

With humble apologies to Prince 1999

“So tonight I’m gonna render like it’s nineteen ninety-nine”

Z Only subpass

FP16 subpass

“Tonemap” (manual exposure control) subpass

With humble apologies to Prince 1999

Render Pass/Subpass

Attachments

DEPTH24 STENCIL8 4	RGBA16 SFLOAT 4	RGBA8 SNORM 4	RGBA8 SNORM 1

	0	1	2	3
Load Op	CLEAR	CLEAR	DONT_CARE	DONT_CARE
Stencil Load Op	DONT_CARE	–	–	–
Initial Layout	DEPTH_STENCIL_ ATTACHMENT_OPTIMAL	COLOR_ ATTACHMENT_OPTIMAL	COLOR_ ATTACHMENT_OPTIMAL	COLOR_ ATTACHMENT_OPTIMAL
Format	D24_UNORM_S8_UINT	R16G16B16A16_SFLOAT	R8G9B8A8_SNORM	R8G9B8A8_SNORM
Samples	4	4	4	1
Store Op	DONT_CARE	DONT_CARE	DONT_CARE	STORE
Final Layout	DEPTH_STENCIL _ATTACHMENT_OPTIMAL	COLOR_ ATTACHMENT_OPTIMAL	COLOR_ ATTACHMENT_OPTIMAL	COLOR_ ATTACHMENT_OPTIMAL

Render Pass/Subpass

Subpasses

DEPTH24 STENCIL8 4	RGBA16 SFLOAT 4	RGBA8 SNORM 4	RGBA8 SNORM 1

	0 - Z Only Subpass	1 - FP16 Subpass	2 - “Tonemap” Subpass
Input Attachment	–	–	1 R16G16B16A16_SLFLOAT x 4
Input Layout	–	–	COLOR_ ATTACHMENT_OPTIMAL
Color Attachment	–	1 R16G16B16A16_SFLOAT x 4	2 R8G8B8A8_SNORM x 4
Color Layout	–	COLOR_ ATTACHMENT_OPTIMAL	COLOR_ ATTACHMENT_OPTIMAL
Resolve Attachment	–	–	3 R8G8B8A8_SNORM x 1
Resolve Layout	–	–	COLOR_ ATTACHMENT_OPTIMAL
Depth Stencil Attachment	0 D24_UNORM_S8_UINT x 4	0 D24_UNORM_S8_UINT x 4	–
Depth Stencil Layout	DEPTH_STENCIL_ ATTACHMENT_OPTIMAL	DEPTH_STENCIL_ READ_ONLY_OPTIMAL	–
Preserve Attachment	–	–	–

Render Pass/Subpass

Dependencies

DEPTH24 STENCIL8 4

RGBA16 SFLOAT 4

RGBA8 SNORM 4

RGBA8 SNORM 1

	0	1
Src Subpass	0 - Z Only Subpass	1 - FP16 Subpass
Dst Subpass	1 - FP16 Subpass	2 - “Tonemap” Subpass
Src Stage Mask	LATE_FRAGMENT_TESTS	COLOR_ATTACHMENT_OUTPUT
Dst Stage mask	EARLY_FRAGMENT_TESTS	FRAGMENT_SHADER
Src Access Mask	DEPTH_STENCIL_ATTACHMENT_WRITE	COLOR_ATTACHMENT_WRITE
Dst Access Mask	DEPTH_STENCIL_ATTACHMENT_READ	INPUT_ATTACHMENT_READ
Dependency Flags	BY_REGION	BY_REGION

Framebuffer

Render Pass Compatibility : Attachments

DEPTH24 STENCIL8 4	RGBA16 SFLOAT 4	RGBA8 SNORM 4	RGBA8 SNORM 1

	0	1	2	3
Load Op	CLEAR	CLEAR	DONT_CARE	DONT_CARE
Stencil Load Op	DONT_CARE	–	–	–
Initial Layout	DEPTH_STENCIL_ ATTACHMENT_OPTIMAL	COLOR_ ATTACHMENT_OPTIMAL	COLOR_ ATTACHMENT_OPTIMAL	COLOR_ ATTACHMENT_OPTIMAL
Format	D24_UNORM_S8_UINT	R16G16B16A16_SFLOAT	R8G9B8A8_SNORM	R8G9B8A8_SNORM
Samples	4	4	4	1
Store Op	DONT_CARE	DONT_CARE	DONT_CARE	STORE
Final Layout	DEPTH_STENCIL _ATTACHMENT_OPTIMAL	COLOR_ ATTACHMENT_OPTIMAL	COLOR_ ATTACHMENT_OPTIMAL	COLOR_ ATTACHMENT_OPTIMAL

Framebuffer

Render Pass Compatibility : Subpasses

DEPTH24 STENCIL8 4	RGBA16 SFLOAT 4	RGBA8 SNORM 4	RGBA8 SNORM 1

	0 - Z Only Subpass	1 - FP16 Subpass	2 - “Tonemap” Subpass
Input Attachment	–	–	1 R16G16B16A16_SLFLOAT x 4
Input Layout	–		COLOR_ ATTACHMENT_OPTIMAL
Color Attachment	–	1 R16G16B16A16_SFLOAT x 4	2 R8G8B8A8_SNORM x 4
Color Layout	–	COLOR_ ATTACHMENT_OPTIMAL	COLOR_ ATTACHMENT_OPTIMAL
Resolve Attachment	–	–	3 R8G8B8A8_SNORM x 1
Resolve Layout	–	–	COLOR_ ATTACHMENT_OPTIMAL
Depth Stencil Attachment	0 D24_UNORM_S8_UINT x 4	0 D24_UNORM_S8_UINT x 4	–
Depth Stencil Layout	DEPTH_STENCIL_ ATTACHMENT_OPTIMAL	DEPTH_STENCIL_ READ_ONLY_OPTIMAL	–
Preserve Attachment	–	–	–

Framebuffer

Render Pass Compatibility : Dependencies

DEPTH24 STENCIL8 4

RGBA16 SFLOAT 4

RGBA8 SNORM 4

RGBA8 SNORM 1

	0	1
Src Subpass	1	2
Dst Subpass	2	3
Src Stage Mask	LATE_FRAGMENT_TESTS	COLOR_ATTACHMENT_OUTPUT
Dst Stage mask	EARLY_FRAGMENT_TESTS	FRAGMENT_SHADER
Src Access Mask	DEPTH_STENCIL_ATTACHMENT_WRITE	COLOR_ATTACHMENT_WRITE
Dst Access Mask	DEPTH_STENCIL_ATTACHMENT_READ	INPUT_ATTACHMENT_READ
Dependency Flags	BY_REGION	BY_REGION

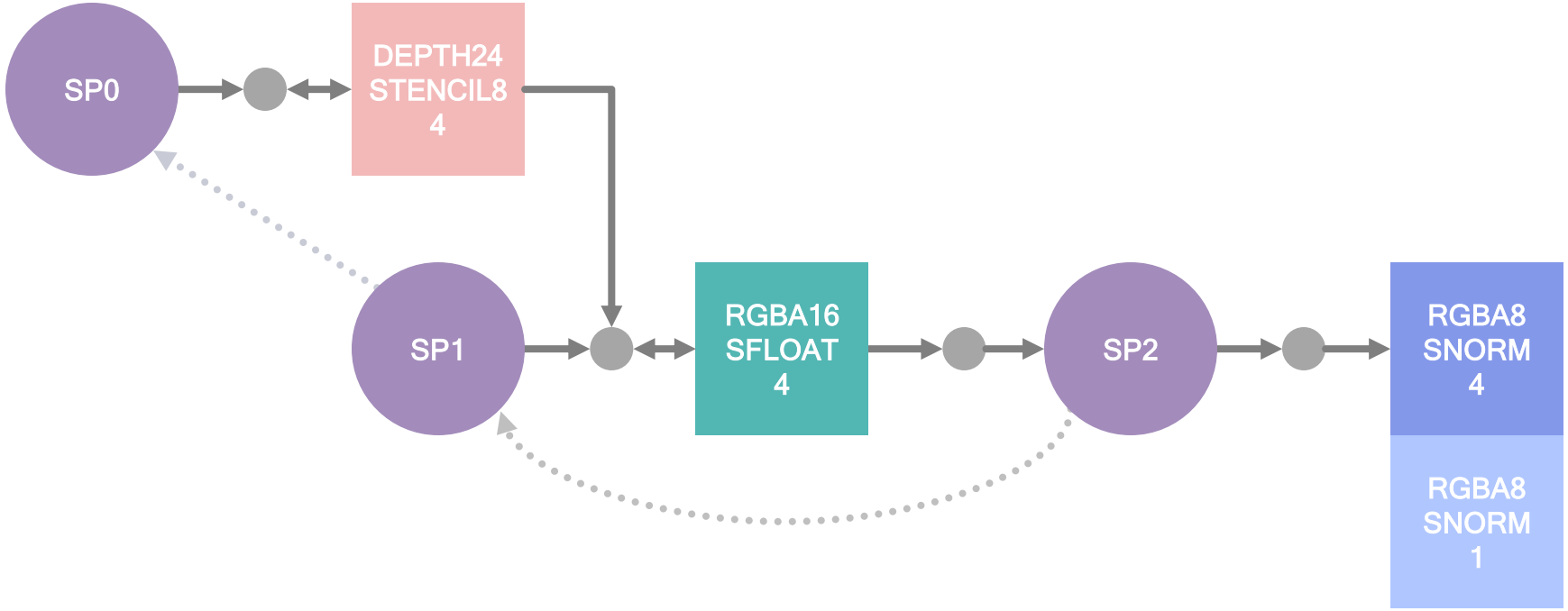
Framebuffer

Image View and Image

	DEPTH24 STENCIL8 4	RGBA16 SFLOAT 4	RGBA8 SNORM 4	RGBA8 SNORM 1
	0	1	2	3
Image View Subresource Range	–	–	–	–
Image Create Flags	MUTABLE_FORMAT?	MUTABLE_FORMAT?	MUTABLE_FORMAT?	MUTABLE_FORMAT?
Image Usage Flags	TRANSIENT_ ATTACHMENT	TRANSIENT_ ATTACHMENT	TRANSIENT_ ATTACHMENT	–
View Format List	–	–	–	–

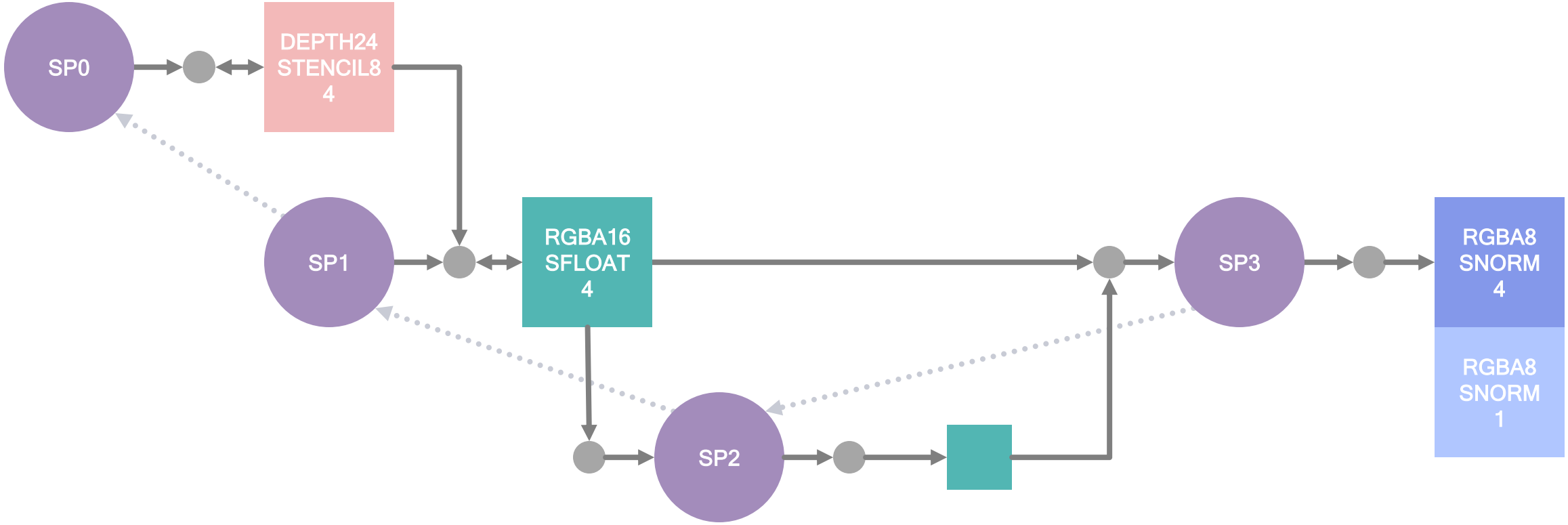
Render like it’s nineteen ninety-nine

(Grossly Simplified Example)

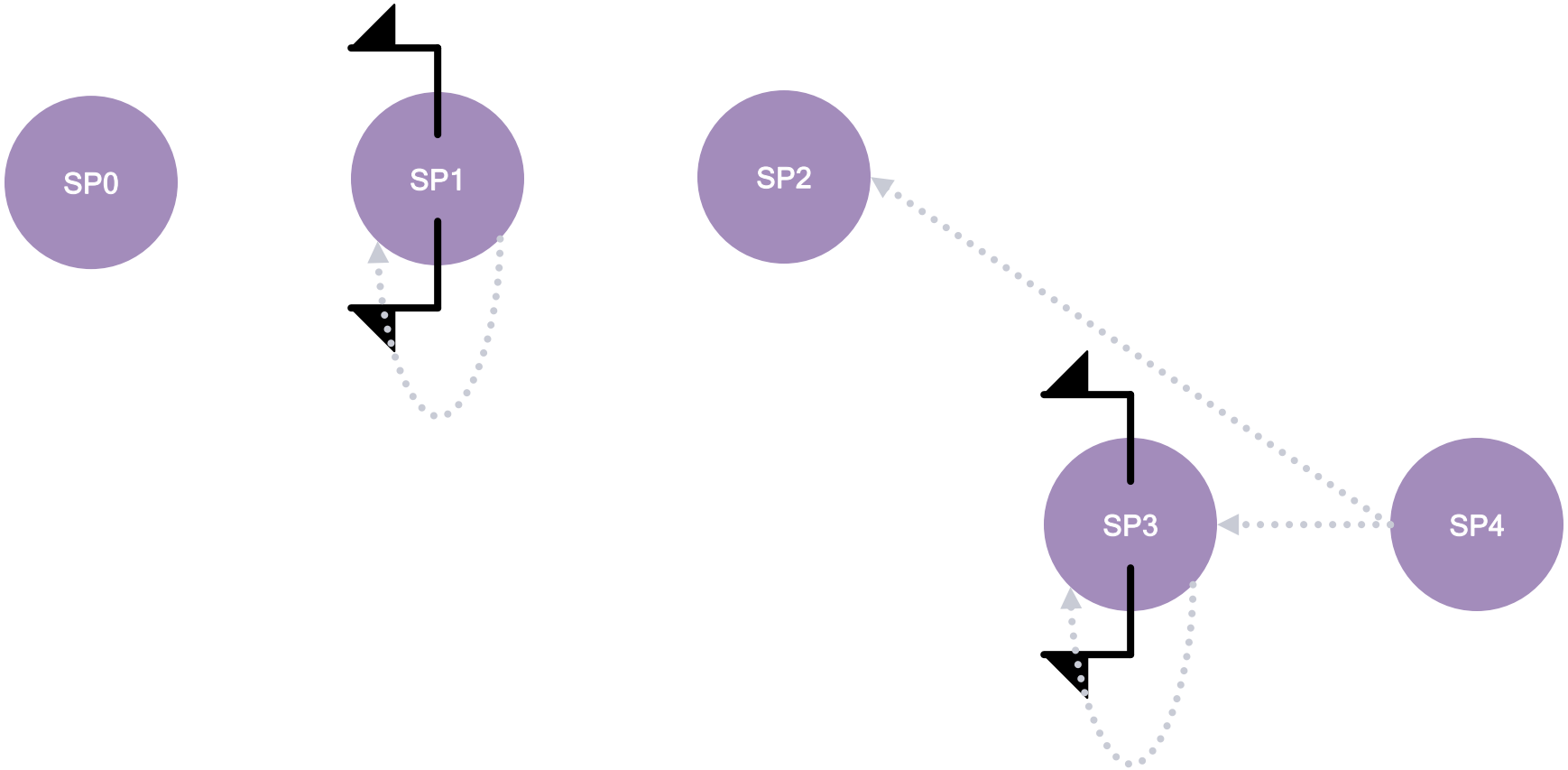


Render like it’s this millennium (ish)

(Somewhat Simplified Example)



Dependencies - Scope



Render Pass/Subpass

Recommendations

- Please, no Über Framebuffers.
- Load Ops and Store Ops are low hanging fruit.
- Gather independent work items same resolution images into a single render pass.
- When you are able to use “trivial” BY_REGION dependencies between two (or more) subpasses, do so.
- Keep expectations realistic.
  - If loads/stores are NOT bandwidth limited, power limited, or performance limited, Render Pass/Subpass will *NOT* be a huge win, but may buy you margin.

Keep it simple, and pocket the change.

Qualcomm

Thank you

Follow us on:

For more information, visit us at:

www.qualcomm.com & www.qualcomm.com/blog