Vulkan™ is a graphics and compute API consisting of procedures and functions to specify shader programs, compute kernels, objects, and operations involved in producing high-quality graphical images, specifically color images of three-dimensional objects. Vulkan is also a pipeline with programmable and state-driven fixed-function stages that are invoked by a set of specific drawing operations.

Specification and additional resources at **www.khronos.org/vulkan**

Color coded names as follows: **Function names** and **Structure names**
[n.n.n] Indicates sections and text in the Vulkan API 1.0 Specification.
P.# Indicates a page in this reference guide for more information.
= 0 Indicates reserved for future use.

Return Codes [2.5.2]

Return codes are reported via VkResult return values.

Success Codes [2.5.2.1]

Success codes are non-negative.

VK_SUCCESS
VK_NOT_READY
VK_TIMEOUT
VK_EVENT_{SET, RESET}
VK_INCOMPLETE
VK_SUBOPTIMAL_KHR

Error Codes [2.5.2.2]

Error codes are negative.

VK_ERROR_OUT_OF_{HOST, DEVICE}_MEMORY
VK_ERROR_{INITIALIZATION, MEMORY_MAP}_FAILED
VK_ERROR_DEVICE_LOST
VK_ERROR_{EXTENSION, FEATURE, LAYER}_NOT_PRESENT
VK_ERROR_INCOMPATIBLE_DRIVER
VK_ERROR_TOO_MANY_OBJECTS
VK_ERROR_FORMAT_NOT_SUPPORTED
VK_ERROR_SURFACE_LOST_KHR
VK_ERROR_OUT_OF_DATE_KHR
VK_ERROR_INCOMPATIBLE_DISPLAY_KHR
VK_ERROR_NATIVE_WINDOW_IN_USE_KHR
VK_ERROR_VALIDATION_FAILED_EXT

Physical Devices [4.1]

```
VkResult vkEnumeratePhysicalDevices(
    VkInstance instance,
    uint32_t* pPhysicalDeviceCount,
    VkPhysicalDevice* pPhysicalDevices);

void vkGetPhysicalDeviceProperties(
    VkPhysicalDevice physicalDevice,
    VkPhysicalDeviceProperties* pProperties);

typedef struct VkPhysicalDeviceProperties {
    uint32_t apiVersion;
    uint32_t driverVersion;
    uint32_t vendorID;
    uint32_t deviceID;
    VkPhysicalDeviceType deviceType;
    char deviceName[
        VK_MAX_PHYSICAL_DEVICE_NAME_SIZE];
    uint8_t pipelineCacheUUID[VK_UUID_SIZE];
    VkPhysicalDeviceLimits limits; P.12
    VkPhysicalDeviceSparseProperties sparseProperties;
} VkPhysicalDeviceProperties;
```

deviceType:
VK_PHYSICAL_DEVICE_TYPE_*X* where *X* is OTHER, INTEGRATED_GPU, DISCRETE_GPU, VIRTUAL_GPU, CPU

```
typedef struct VkPhysicalDeviceSparseProperties {
    VkBool32 residencyStandard2DBlockShape;
    VkBool32
        residencyStandard2DMultisampleBlockShape;
    VkBool32 residencyStandard3DBlockShape;
    VkBool32 residencyAlignedMipSize;
    VkBool32 residencyNonResidentStrict;
} VkPhysicalDeviceSparseProperties;

void vkGetPhysicalDeviceQueueFamilyProperties(
    VkPhysicalDevice physicalDevice,
    uint32_t* pQueueFamilyPropertyCount,
    VkQueueFamilyProperties*
        pQueueFamilyProperties);

typedef struct VkQueueFamilyProperties {
    VkQueueFlags queueFlags;
    uint32_t queueCount;
    uint32_t timestampValidBits;
    VkExtent3D minImageTransferGranularity; P.10
} VkQueueFamilyProperties;
```

queueFlags:
VK_QUEUE_*X*_BIT where *X* is GRAPHICS, COMPUTE, TRANSFER, SPARSE_BINDING

Command Function Pointers [3.1]

```
PFN_vkVoidFunction vkGetInstanceProcAddr(
    VkInstance instance,
    const char *pName);

PFN_vkVoidFunction vkGetDeviceProcAddr(
    VkDevice device,
    const char *pName);
```

Instances [3.2]

```
VkResult vkCreateInstance(
    const VkInstanceCreateInfo* pCreateInfo,
    const VkAllocationCallbacks *pAllocator, P.10
    VkInstance *pInstance);

typedef struct VkInstanceCreateInfo {
    VkStructureType sType;
    const void *pNext;
    VkInstanceCreateFlags flags; = 0
    const VkApplicationInfo* pApplicationInfo;
    uint32_t enabledLayerCount;
    const char* const* ppEnabledLayerNames;
    uint32_t enabledExtensionCount;
    const char* const* ppEnabledExtensionNames;
} VkInstanceCreateInfo;

typedef struct VkApplicationInfo {
    VkStructureType sType;
    const void *pNext;
    const char* pApplicationName;
    uint32_t applicationVersion;
    const char* pEngineName;
    uint32_t engineVersion;
    uint32_t apiVersion;
} VkApplicationInfo;

void vkDestroyInstance(
    VkInstance instance,
    const VkAllocationCallbacks *pAllocator); P.10
```

Devices

Device Creation [4.2.1]

```
VkResult vkCreateDevice(
    VkPhysicalDevice physicalDevice,
    const VkDeviceCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.10
    VkDevice* pDevice);

typedef struct VkDeviceCreateInfo {
    VkStructureType sType;
    const void* pNext;
    VkDeviceCreateFlags flags; = 0
    uint32_t queueCreateInfoCount;
    const VkDeviceQueueCreateInfo* pQueueCreateInfos;
    uint32_t enabledLayerCount;
    const char* const* ppEnabledLayerNames;
    uint32_t enabledExtensionCount;
    const char* const* ppEnabledExtensionNames;
    const VkPhysicalDeviceFeatures* pEnabledFeatures; P.11
} VkDeviceCreateInfo;

typedef struct VkDeviceQueueCreateInfo {
    VkStructureType sType;
    const void* pNext;
    VkDeviceQueueCreateFlags flags; = 0
    uint32_t queueFamilyIndex;
    uint32_t queueCount;
    const float* pQueuePriorities;
} VkDeviceQueueCreateInfo;
```

Device Idle [4.2.3]

```
VkResult vkDeviceWaitIdle(
    VkDevice device);
```

Device Destruction [4.2.5]

```
void vkDestroyDevice(
    VkDevice device,
    const VkAllocationCallbacks* pAllocator); P.10
```

Queues [4.3]

Queue Creation [4.3.2]

```
void vkGetDeviceQueue(
    VkDevice device,
    uint32_t queueFamilyIndex,
    uint32_t queueIndex,
    VkQueue* pQueue);
```

Queue Synchronization [4.3.5]

```
VkResult vkQueueWaitIdle(
    VkQueue queue);
```

Command Buffers [5]

Command Pools [5.1]

```
VkResult vkCreateCommandPool(
    VkDevice device,
    const VkCommandPoolCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.10
    VkCommandPool* pCommandPool);

typedef struct VkCommandPoolCreateInfo {
    VkStructureType sType;
    const void* pNext;
    VkCommandPoolCreateFlags flags;
    uint32_t queueFamilyIndex;
} VkCommandPoolCreateInfo;
```

flags: VK_COMMAND_POOL_CREATE_*X*_BIT where *X* is RESET_COMMAND_BUFFER, TRANSIENT

```
VkResult vkResetCommandPool(
    VkDevice device,
    VkCommandPool commandPool,
    VkCommandPoolResetFlags flags);
```

flags:
VK_COMMAND_POOL_RESET_RELEASE_RESOURCES_BIT

```
void vkDestroyCommandPool(
    VkDevice device,
    VkCommandPool commandPool,
    const VkAllocationCallbacks* pAllocator); P.10
```

Command Buffer Lifetime [5.2]

```
VkResult vkAllocateCommandBuffers(
    VkDevice device,
    const VkCommandBufferAllocateInfo* pAllocateInfo,
    VkCommandBuffer* pCommandBuffers);

typedef struct VkCommandBufferAllocateInfo{
    VkStructureType sType;
    const void* pNext;
    VkCommandPool commandPool;
    VkCommandBufferLevel level;
    uint32_t commandBufferCount;
} VkCommandBufferAllocateInfo;
```

level:
VK_COMMAND_BUFFER_LEVEL_{PRIMARY, SECONDARY}

```
VkResult vkResetCommandBuffer(
    VkCommandBuffer commandBuffer,
    VkCommandBufferResetFlags flags);
```

flags:
VK_COMMAND_BUFFER_RESET_RELEASE_RESOURCES_BIT

```
void vkFreeCommandBuffers(
    VkDevice device,
    VkCommandPool commandPool,
    uint32_t commandBufferCount,
    const VkCommandBuffer* pCommandBuffers);
```

Continued on next page >

Command Buffers (continued)

Command Buffer Recording [5.3]

VkResult **vkBeginCommandBuffer**(
  VkCommandBuffer *commandBuffer*,
  const VkCommandBufferBeginInfo* *pBeginInfo*);

typedef struct **VkCommandBufferBeginInfo**{
  VkStructureType *sType*;
  const void* *pNext*;
  VkCommandBufferUsageFlags *flags*;
  const VkCommandBufferInheritanceInfo* *pInheritanceInfo*;
} VkCommandBufferBeginInfo;

flags: VK_COMMAND_BUFFER_USAGE_*X*_BIT where *X* is ONE_TIME_SUBMIT, RENDER_PASS_CONTINUE, SIMULTANEOUS_USE

typedef struct **VkCommandBufferInheritanceInfo** {
  VkStructureType *sType*;
  const void* *pNext*;
  VkRenderPass *renderPass*;
  uint32_t *subpass*;
  VkFramebuffer *framebuffer*;
  VkBool32 *occlusionQueryEnable*;
  VkQueryControlFlags *queryFlags*;
  VkQueryPipelineStatisticFlags *pipelineStatistics*; P.12
} VkCommandBufferInheritanceInfo;

queryFlags: VK_QUERY_CONTROL_PRECISE_BIT

VkResult **vkEndCommandBuffer**(
  VkCommandBuffer *commandBuffer*);

Command Buffer Submission [5.4]

VkResult **vkQueueSubmit**(
  VkQueue *queue*,
  uint32_t *submitCount*,
  const VkSubmitInfo* *pSubmits*,
  VkFence *fence*);

typedef struct **VkSubmitInfo**{
  VkStructureType *sType*;
  const void* *pNext*;
  uint32_t *waitSemaphoreCount*;
  const VkSemaphore* *pWaitSemaphores*;
  const VkPipelineStageFlags* *pWaitDstStageMask*; P.12
  uint32_t *commandBufferCount*;
  const VkCommandBuffer* *pCommandBuffers*;
  uint32_t *signalSemaphoreCount*;
  const VkSemaphore* *pSignalSemaphores*;
} VkSubmitInfo;

Secondary Command Buffer Execution [5.6]

void **vkCmdExecuteCommands**(
  VkCommandBuffer *commandBuffer*,
  uint32_t *commandBufferCount*,
  const VkCommandBuffer* *pCommandBuffers*);

Commands Allowed Inside Command Buffers

The following table shows functions which record commands in command buffers. They are on the primary and secondary command buffer level, except for the Render pass and Execute commands, which are only on the primary.

Set state in the command buffer
(Both inside and outside the render pass.)

vkCmdBindPipeline	vkCmdBindDescriptorSets
vkCmdBindVertexBuffers	vkCmdBindIndexBuffer

Dynamic state functions
(Both inside and outside the render pass.)

vkCmdSetViewport	vkCmdSetStencilCompareMask
vkCmdSetScissor	vkCmdSetStencilWriteMask
vkCmdSetDepthBounds	vkCmdSetStencilReference
vkCmdSetLineWidth	vkCmdSetBlendConstants
vkCmdSetDepthBias	

Cause the device to perform processing
(Inside the render pass.)

vkCmdDraw	vkCmdDrawIndirect
vkCmdDrawIndexed	vkCmdDrawIndexedIndirect

Dispatch compute
(Outside the render pass.)

vkCmdDispatch	vkCmdDispatchIndirect

Update and modify images and buffers
(Outside the render pass.)

vkCmdCopyBuffer	vkCmdUpdateBuffer
vkCmdCopyImage	vkCmdFillBuffer
vkCmdBlitImage	vkCmdClearColorImage
vkCmdCopyBufferToImage	vkCmdClearDepthStencilImage
vkCmdCopyImageToBuffer	vkCmdResolveImage

Update and modify the currently bound framebuffer
(Inside the render pass.)

vkCmdClearAttachments

Synchronization
([O] outside only, or [B] both inside and outside the render pass.)

vkCmdSetEvent [O]	vkCmdWaitEvents [B]
vkCmdResetEvent [O]	vkCmdPipelineBarrier [B]

Queries
([O] outside only, or [B] both inside and outside the render pass.)

vkCmdBeginQuery [B]	vkCmdCopyQueryPoolResults [O]
vkCmdEndQuery [B]	vkCmdWriteTimestamp [B]
vkCmdResetQueryPool [O]	

Push constants
(Both inside and outside the render pass.)

vkCmdPushConstants

Render passes (Primary command buffer level)
([I] inside or [O] outside the render pass.)

vkCmdBeginRenderPass [O]	vkCmdEndRenderPass [I]
vkCmdNextSubpass [I]	

Execute commands (Primary command buffer level)
(Both inside and outside the render pass.)

vkCmdExecuteCommands

void **vkCmdWaitEvents**(
  VkCommandBuffer *commandBuffer*,
  uint32_t *eventCount*,
  const VkEvent* *pEvents*,
  VkPipelineStageFlags *srcStageMask*, P.12
  VkPipelineStageFlags *dstStageMask*, P.12
  uint32_t *memoryBarrierCount*,
  const VkMemoryBarrier* *pMemoryBarriers*,
  uint32_t *bufferMemoryBarrierCount*,
  const VkBufferMemoryBarrier* *pBufferMemoryBarriers*,
  uint32_t *imageMemoryBarrierCount*,
  const VkImageMemoryBarrier* *pImageMemoryBarriers*);

***ppMemoryBarriers:* See VkMemoryBarrier, VkBufferMemoryBarrier, or VkImageMemoryBarrier P.11

Pipeline Barriers [6.5]

Synchronizes an earlier set of commands against a later set of commands.

void **vkCmdPipelineBarrier**(
  VkCommandBuffer *commandBuffer*,
  VkPipelineStageFlags *srcStageMask*, P.12
  VkPipelineStageFlags *dstStageMask*, P.12
  VkDependencyFlags *dependencyFlags*,
  uint32_t *memoryBarrierCount*,
  const VkMemoryBarrier* *pMemoryBarriers*,
  uint32_t *bufferMemoryBarrierCount*,
  const VkBufferMemoryBarrier* *pBufferMemoryBarriers*,
  uint32_t *imageMemoryBarrierCount*,
  const VkImageMemoryBarrier* *pImageMemoryBarriers*);

dependencyFlags: VK_DEPENDENCY_BY_REGION_BIT

***ppMemoryBarriers:* See VkMemoryBarrier, VkBufferMemoryBarrier, or VkImageMemoryBarrier P.11

Synchronization and Cache Control [6]

Fences [6.1]

Fence status is always either *signaled* or *unsignaled*.

VkResult **vkCreateFence**(
  VkDevice *device*,
  const VkFenceCreateInfo* *pCreateInfo*,
  const VkAllocationCallbacks* *pAllocator*, P.10
  VkFence* *pFence*);

typedef struct **VkFenceCreateInfo** {
  VkStructureType *sType*;
  const void* *pNext*;
  VkFenceCreateFlags *flags*;
} VkFenceCreateInfo;

flags: VK_FENCE_CREATE_SIGNALED_BIT

void **vkDestroyFence**(
  VkDevice *device*,
  VkFence *fence*,
  const VkAllocationCallbacks* *pAllocator*); P.10

VkResult **vkGetFenceStatus**(
  VkDevice *device*,
  VkFence *fence*);

VkResult **vkResetFences**(
  VkDevice *device*,
  uint32_t *fenceCount*,
  const VkFence* *pFences*);

VkResult **vkWaitForFences**(
  VkDevice *device*,
  uint32_t *fenceCount*,
  const VkFence* *pFences*,
  VkBool32 *waitAll*,
  uint64_t *timeout*);

Semaphores [6.2]

Semaphore status is always either *signaled* or *unsignaled*.

VkResult **vkCreateSemaphore**(
  VkDevice *device*,
  const VkSemaphoreCreateInfo* *pCreateInfo*,
  const VkAllocationCallbacks* *pAllocator*, P.10
  VkSemaphore* *pSemaphore*);

typedef struct **VkSemaphoreCreateInfo** {
  VkStructureType *sType*;
  const void* *pNext*;
  VkSemaphoreCreateFlags *flags*; = 0
} VkSemaphoreCreateInfo;

void **vkDestroySemaphore**(
  VkDevice *device*,
  VkSemaphore *semaphore*,
  const VkAllocationCallbacks* *pAllocator*); P.10

Events [6.3]

Events represent a fine-grained synchronization primitive that can be used to gauge progress through a sequence of commands executed on a queue.

VkResult **vkCreateEvent**(
  VkDevice *device*,
  const VkEventCreateInfo* *pCreateInfo*,
  const VkAllocationCallbacks* *pAllocator*, P.10
  VkEvent* *pEvent*);

typedef struct **VkEventCreateInfo** {
  VkStructureType *sType*;
  const void* *pNext*;
  VkEventCreateFlags *flags*; = 0
} VkEventCreateInfo;

void **vkDestroyEvent**(
  VkDevice *device*,
  VkEvent *event*,
  const VkAllocationCallbacks* *pAllocator*); P.10

VkResult **vkGetEventStatus**(
  VkDevice *device*,
  VkEvent *event*);

VkResult **vk[Set, Reset]Event**(
  VkDevice *device*,
  VkEvent *event*);

VkResult **vkCmd[Set, Reset]Event**(
  VkCommandBuffer *commandBuffer*,
  VkEvent *event*,
  VkPipelineStageFlags *stageMask*); P.12

Render Pass [7]

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.

Render Pass Creation [7.1]

VkResult **vkCreateRenderPass**(
  VkDevice *device*,
  const VkRenderPassCreateInfo* *pCreateInfo*,
  const VkAllocationCallbacks* *pAllocator*, P.10
  VkRenderPass* *pRenderPass*);

typedef struct **VkRenderPassCreateInfo** {
  VkStructureType *sType*;
  const void* *pNext*;
  VkRenderPassCreateFlags *flags*; = 0
  uint32_t *attachmentCount*;
  const VkAttachmentDescription* *pAttachments*;
  uint32_t *subpassCount*;
  const VkSubpassDescription* *pSubpasses*;
  uint32_t *dependencyCount*;
  const VkSubpassDependency* *pDependencies*;
} VkRenderPassCreateInfo;

Continued on next page >

Render Pass (continued)

```
typedef struct VkAttachmentDescription {
    VkAttachmentDescriptionFlags flags;
    VkFormat format; P.11
    VkSampleCountFlagBits samples; P.12
    VkAttachmentLoadOp loadOp;
    VkAttachmentStoreOp storeOp;
    VkAttachmentLoadOp stencilLoadOp;
    VkAttachmentStoreOp stencilStoreOp;
    VkImageLayout initialLayout; P.11
    VkImageLayout finalLayout; P.11
} VkAttachmentDescription;
```

loadOp, stencilLoadOp: VK_ATTACHMENT_LOAD_OP_*X* where *X* is LOAD, CLEAR, DONT_CARE

storeOp, stencilStoreOp: VK_ATTACHMENT_STORE_OP_*X* where *X* is STORE, DONT_CARE

flags: VK_ATTACHMENT_DESCRIPTION_MAY_ALIAS_BIT

```
typedef struct VkSubpassDescription {
    VkSubpassDescriptionFlags flags; = 0
    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;
```

pipelineBindPoint: VK_PIPELINE_BIND_POINT_GRAPHICS

```
typedef struct VkAttachmentReference {
    uint32_t attachment;
    VkImageLayout layout; P.11
} VkAttachmentReference;
```

```
typedef struct VkSubpassDependency {
    uint32_t srcSubpass;
    uint32_t dstSubpass;
    VkPipelineStageFlags srcStageMask; P.10
    VkPipelineStageFlags dstStageMask; P.10
    VkAccessFlags srcAccessMask; P.10
    VkAccessFlags dstAccessMask; P.10
    VkDependencyFlags dependencyFlags;
} VkSubpassDependency;
```

```
void vkDestroyRenderPass(
    VkDevice device,
    VkRenderPass renderPass,
    const VkAllocationCallbacks* pAllocator); P.10
```

Framebuffers [7.3]

```
VkResult vkCreateFramebuffer(
    VkDevice device,
    const VkFramebufferCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.10
    VkFramebuffer* pFramebuffer);
```

```
typedef struct VkFramebufferCreateInfo {
    VkStructureType sType;
    const void* pNext;
    VkFramebufferCreateFlags flags; = 0
    VkRenderPass renderPass;
    uint32_t attachmentCount;
    const VkImageView* pAttachments;
    uint32_t width;
    uint32_t height;
    uint32_t layers;
} VkFramebufferCreateInfo;
```

```
void vkDestroyFramebuffer(
    VkDevice device,
    VkFramebuffer framebuffer,
    const VkAllocationCallbacks* pAllocator); P.10
```

Render Pass Commands [7.4]

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

contents: VK_SUBPASS_CONTENTS_*X* where *X* is INLINE, SECONDARY_COMMAND_BUFFERS

```
typedef struct VkRenderPassBeginInfo {
    VkStructureType sType;
    const void* pNext;
    VkRenderPass renderPass;
    VkFramebuffer framebuffer;
    VkRect2D renderArea; P.12
    uint32_t clearValueCount;
    const VkClearValue* pClearValues; P.10
} VkRenderPassBeginInfo;
```

```
void vkGetRenderAreaGranularity(
    VkDevice device,
    VkRenderPass renderPass,
    VkExtent2D* pGranularity); P.10
```

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

contents: VK_SUBPASS_CONTENTS_*X* where *X* is INLINE, SECONDARY_COMMAND_BUFFERS

```
void vkCmdEndRenderPass(
    VkCommandBuffer commandBuffer);
```

Shaders [8]

Shader Modules [8.1]

```
VkResult vkCreateShaderModule(
    VkDevice device,
    const VkShaderModuleCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.10
    VkShaderModule* pShaderModule);
```

```
typedef struct VkShaderModuleCreateInfo {
    VkStructureType sType;
    const void* pNext;
    VkShaderModuleCreateFlags flags; = 0
    size_t codeSize;
    const uint32_t* pCode;
} VkShaderModuleCreateInfo;
```

```
void vkDestroyShaderModule(
    VkDevice device,
    VkShaderModule shaderModule,
    const VkAllocationCallbacks* pAllocator); P.10
```

Built-in Variables [14.6]

The built-in variables listed below are accessed in shaders by declaring the variable using a **BuiltIn** decoration.

Decoration	Type
ClipDistance	Array of 32-bit float values
CullDistance	Array of 32-bit float values
FragCoord	Four-component vector of 32-bit float values
FragDepth	Scalar 32-bit float value
FrontFacing	Scalar 32-bit integer
GlobalInvocationID	Three-component vector of 32-bit ints
HelperInvocation	Scalar 32-bit integer
InvocationID	Scalar 32-bit integer
InstanceIndex	Scalar 32-bit integer
Layer	Scalar 32-bit integer
LocalInvocationID	Three-component vector of 32-bit ints
NumWorkGroups	Three-component vector of 32-bit ints
PatchVertices	Scalar 32-bit integer
PointCoord	Two-component vector of 32-bit float values
PointSize	Scalar 32-bit float value
Position	Four-component vector of 32-bit float values
PrimitiveID	Scalar 32-bit integer
SampleID	Scalar 32-bit integer
SampleMask	Array of 32-bit integers
SamplePosition	Two-component vector of float values
TessellationCoord	Three-component vector of 32-bit float values
TessellationLevelOuter	Array of size two, containing 32-bit float values
TessellationLevelInner	Array of size four, containing 32-bit float values
VertexIndex	32-bit integer
ViewportIndex	32-bit integer
WorkgroupID	Three-component vector of 32-bit ints

Pipelines [9]

Processing pipelines are either compute or graphics pipelines.

Compute Pipelines [9.1]

Compute pipelines consist of a single static compute shader stage and the pipeline layout.

```
VkResult vkCreateComputePipelines(
    VkDevice device,
    VkPipelineCache pipelineCache,
    uint32_t createInfoCount,
    const VkComputePipelineCreateInfo* pCreateInfos,
    const VkAllocationCallbacks* pAllocator, P.10
    VkPipeline* pPipelines);
```

```
typedef struct VkComputePipelineCreateInfo {
    VkStructureType sType;
    const void* pNext;
    VkPipelineCreateFlags flags;
    VkPipelineShaderStageCreateInfo stage; P.12
    VkPipelineLayout layout;
    VkPipeline basePipelineHandle;
    int32_t basePipelineIndex;
} VkComputePipelineCreateInfo;
```

flags: Combination of VK_PIPELINE_CREATE_*X*_BIT where *X* is DISABLE_OPTIMIZATION, ALLOW_DERIVATIVES, DERIVATIVE

Graphics Pipelines [9.2]

```
VkResult vkCreateGraphicsPipelines(
    VkDevice device,
    VkPipelineCache pipelineCache,
    uint32_t createInfoCount,
    const VkGraphicsPipelineCreateInfo* pCreateInfos,
    const VkAllocationCallbacks* pAllocator, P.10
    VkPipeline* pPipelines);
```

In VkGraphicsPipelineCreateInfo below, replace *X* with VkPipeline and replace *Y* with StateCreateInfo.

```
typedef struct VkGraphicsPipelineCreateInfo {
    VkStructureType sType;
    const void* pNext;
    VkPipelineCreateFlags flags;
    uint32_t stageCount;
    const VkPipelineShaderStageCreateInfo* pStages; P.12
    const XVertexInputY* pVertexInputState;
    const XInputAssemblyY* pInputAssemblyState;
    const XTessellationY* pTessellationState;
    const XViewportY* pViewportState;
    const XRasterizationY* pRasterizationState;
    const XMultisampleY* pMultisampleState;
    const XDepthStencilY* pDepthStencilState;
    const XColorBlendY* pColorBlendState;
    const XDynamicY* pDynamicState;
    VkPipelineLayout layout;
    VkRenderPass renderPass;
    uint32_t subpass;
    VkPipeline basePipelineHandle;
    int32_t basePipelineIndex;
} VkGraphicsPipelineCreateInfo;
```

flags: VK_PIPELINE_CREATE_*Z*_BIT where *Z* is DISABLE_OPTIMIZATION, ALLOW_DERIVATIVES, DERIVATIVE

```
typedef struct VkPipelineVertexInputStateCreateInfo {
    VkStructureType sType;
    const void* pNext;
    VkPipelineVertexInputStateCreateFlags flags; = 0
    uint32_t vertexBindingDescriptionCount;
    const VkVertexInputBindingDescription*
        pVertexBindingDescriptions;
    uint32_t vertexAttributeDescriptionCount;
    const VkVertexInputAttributeDescription*
        pVertexAttributeDescriptions;
} VkPipelineVertexInputStateCreateInfo;
```

```
typedef struct VkVertexInputBindingDescription {
    uint32_t binding;
    uint32_t stride;
    VkVertexInputRate inputRate;
} VkVertexInputBindingDescription;
```

inputRate: VK_VERTEX_INPUT_RATE_{VERTEX, INSTANCE}

```
typedef struct VkVertexInputAttributeDescription {
    uint32_t location;
    uint32_t binding;
    VkFormat format; P.11
    uint32_t offset;
} VkVertexInputAttributeDescription;
```

```
typedef struct VkPipelineInputAssemblyStateCreateInfo {
    VkStructureType sType;
    const void* pNext;
    VkPipelineInputAssemblyStateCreateFlags flags; = 0
    VkPrimitiveTopology topology;
    VkBool32 primitiveRestartEnable;
} VkPipelineInputAssemblyStateCreateInfo;
```

topology: VK_PRIMITIVE_TOPOLOGY_*X* where *X* is POINT_LIST, LINE_LIST, LINE_STRIP, TRIANGLE_LIST, TRIANGLE_STRIP, TRIANGLE_FAN, LINE_{LIST, STRIP}_WITH_ADJACENCY, TRIANGLE_{LIST, STRIP}_WITH_ADJACENCY, PATCH_LIST

Continued on next page >

Pipelines (continued)

typedef struct **VkPipelineTessellationStateCreateInfo** {
    VkStructureType *sType*;
    const void* *pNext*;
    VkPipelineTessellationStateCreateFlags *flags*; = 0
    uint32_t *patchControlPoints*;
} VkPipelineTessellationStateCreateInfo;

typedef struct **VkPipelineViewportStateCreateInfo** {
    VkStructureType *sType*;
    const void* *pNext*;
    VkPipelineViewportStateCreateFlags *flags*; = 0
    uint32_t *viewportCount*;
    const VkViewport* *pViewports*; P.11
    uint32_t *scissorCount*;
    const VkRect2D* *pScissors*; P.12
} VkPipelineViewportStateCreateInfo;

typedef struct **VkPipelineRasterizationStateCreateInfo** {
    VkStructureType *sType*;
    const void* *pNext*;
    VkPipelineRasterizationStateCreateFlags *flags*; = 0
    VkBool32 *depthClampEnable*;
    VkBool32 *rasterizerDiscardEnable*;
    VkPolygonMode *polygonMode*;
    VkCullModeFlags *cullMode*;
    VkFrontFace *frontFace*;
    VkBool32 *depthBiasEnable*;
    float *depthBiasConstantFactor*;
    float *depthBiasClamp*;
    float *depthBiasSlopeFactor*;
    float *lineWidth*;
} VkPipelineRasterizationStateCreateInfo;

polygonMode: VK_POLYGON_MODE_{FILL, LINE, POINT}

cullMode: VK_CULL_MODE_*X* where *X* is NONE, FRONT_BIT, BACK_BIT, FRONT_AND_BACK

frontFace: VK_FRONT_FACE_[COUNTER_]CLOCKWISE

typedef struct **VkPipelineMultisampleStateCreateInfo** {
    VkStructureType *sType*;
    const void* *pNext*;
    VkPipelineMultisampleStateCreateFlags *flags*; = 0
    VkSampleCountFlagBits *rasterizationSamples*; P.12
    VkBool32 *sampleShadingEnable*;
    float *minSampleShading*;
    const VkSampleMask* *pSampleMask*;
    VkBool32 *alphaToCoverageEnable*;
    VkBool32 *alphaToOneEnable*;
} VkPipelineMultisampleStateCreateInfo;

typedef struct **VkPipelineDepthStencilStateCreateInfo** {
    VkStructureType *sType*;
    const void* *pNext*;
    VkPipelineDepthStencilStateCreateFlags *flags*; = 0
    VkBool32 *depthTestEnable*;
    VkBool32 *depthWriteEnable*;
    VkCompareOp *depthCompareOp*; P.11
    VkBool32 *depthBoundsTestEnable*;
    VkBool32 *stencilTestEnable*;
    VkStencilOpState *front*;
    VkStencilOpState *back*;
    float *minDepthBounds*;
    float *maxDepthBounds*;
} VkPipelineDepthStencilStateCreateInfo;

typedef struct **VkStencilOpState** {
    VkStencilOp *failOp*;
    VkStencilOp *passOp*;
    VkStencilOp *depthFailOp*;
    VkCompareOp *compareOp*; P.11
    uint32_t *compareMask*;
    uint32_t *writeMask*;
    uint32_t *reference*;
} VkStencilOpState;

enum VkStencilOp: VK_STENCIL_OP_*X* where *X* is KEEP, ZERO, REPLACE, INCREMENT_AND_{CLAMP, WRAP}, INVERT, DECREMENT_AND_{CLAMP, WRAP}

typedef struct **VkPipelineColorBlendStateCreateInfo** {
    VkStructureType *sType*;
    const void* *pNext*;
    VkPipelineColorBlendStateCreateFlags *flags*; = 0
    VkBool32 *logicOpEnable*;
    VkLogicOp *logicOp*;
    uint32_t *attachmentCount*;
    const VkPipelineColorBlendAttachmentState* *pAttachments*;
    float *blendConstants*[4];
} VkPipelineColorBlendStateCreateInfo;

logicOp: VK_LOGIC_OP_*X* where *X* is CLEAR, AND, AND_REVERSE, COPY, AND_INVERTED, NO_OP, XOR, OR, NOR, EQUIVALENT, INVERT, OR_REVERSE, COPY_INVERTED, OR_INVERTED, NAND, SET

blendOp: VK_BLEND_OP_*X* where *X* is ADD, SUBTRACT, REVERSE_SUBTRACT, MIN, MAX

colorWriteMask: VK_COLOR_COMPONENT_*X* where *X* is R_BIT, G_BIT, B_BIT, A_BIT

typedef struct **VkPipelineColorBlendAttachmentState** {
    VkBool32 *blendEnable*;
    VkBlendFactor *srcColorBlendFactor*;
    VkBlendFactor *dstColorBlendFactor*;
    VkBlendOp *colorBlendOp*;
    VkBlendFactor *srcAlphaBlendFactor*;
    VkBlendFactor *dstAlphaBlendFactor*;
    VkBlendOp *alphaBlendOp*;
    VkColorComponentFlags *colorWriteMask*;
} VkPipelineColorBlendAttachmentState;

enum VkBlendFactor:
VK_BLEND_FACTOR_*X* where *X* is ZERO, ONE, [ONE_MINUS_]SRC_COLOR, [ONE_MINUS_]DST_COLOR, [ONE_MINUS_]SRC_ALPHA, [ONE_MINUS_]DST_ALPHA, [ONE_MINUS_]CONSTANT_COLOR, [ONE_MINUS_]CONSTANT_ALPHA, SRC_ALPHA_SATURATE, [ONE_MINUS_]SRC1_COLOR, [ONE_MINUS_]SRC1_ALPHA

colorWriteMask:
VK_COLOR_COMPONENT_*X*_BIT where *X* is R, G, B, A

typedef struct **VkPipelineDynamicStateCreateInfo** {
    VkStructureType *sType*;
    const void* *pNext*;
    VkPipelineDynamicStateCreateFlags *flags*; = 0
    uint32_t *dynamicStateCount*;
    const VkDynamicState* *pDynamicStates*;
} VkPipelineDynamicStateCreateInfo;

pDynamicStates: Array of VK_DYNAMIC_STATE_*X* where *X* is VIEWPORT, SCISSOR, LINE_WIDTH, DEPTH_BIAS, BLEND_CONSTANTS, DEPTH_BOUNDS, STENCIL_REFERENCE, STENCIL_COMPARE_MASK, STENCIL_WRITE_MASK

Pipeline Destruction [9.3]

void **vkDestroyPipeline**(
    VkDevice *device*,
    VkPipeline *pipeline*,
    const VkAllocationCallbacks* *pAllocator*); P.10

Pipeline Cache [9.6]

Pipeline cache objects allow the result of pipeline construction to be reused between pipelines and between runs of an application.

VkResult **vkCreatePipelineCache**(
    VkDevice *device*,
    const VkPipelineCacheCreateInfo* *pCreateInfo*,
    const VkAllocationCallbacks* *pAllocator*, P.10
    VkPipelineCache* *pPipelineCache*);

typedef struct **VkPipelineCacheCreateInfo** {
    VkStructureType *sType*;
    const void* *pNext*;
    VkPipelineCacheCreateFlags *flags*; = 0
    size_t *initialDataSize*;
    const void* *pInitialData*;
} VkPipelineCacheCreateInfo;

VkResult **vkMergePipelineCaches**(
    VkDevice *device*,
    VkPipelineCache *dstCache*,
    uint32_t *srcCacheCount*,
    const VkPipelineCache* *pSrcCaches*);

VkResult **vkGetPipelineCacheData**(
    VkDevice *device*,
    VkPipelineCache *pipelineCache*,
    size_t* *pDataSize*,
    void* *pData*);

void **vkDestroyPipelineCache**(
    VkDevice *device*,
    VkPipelineCache *pipelineCache*,
    const VkAllocationCallbacks* *pAllocator*); P.10

Pipeline Binding [9.8]

void **vkCmdBindPipeline**(
    VkCommandBuffer *commandBuffer*,
    VkPipelineBindPoint *pipelineBindPoint*,
    VkPipeline *pipeline*);

pipelineBindPoint:
VK_PIPELINE_BIND_POINT_[GRAPHICS, COMPUTE]

Memory Allocation [10]

Device Memory [10.2]

Device memory is memory that is visible to the device.

void **vkGetPhysicalDeviceMemoryProperties**(
    VkPhysicalDevice *physicalDevice*,
    VkPhysicalDeviceMemoryProperties* *pMemoryProperties*);

typedef struct **VkPhysicalDeviceMemoryProperties** {
    uint32_t *memoryTypeCount*;
    VkMemoryType *memoryTypes*[VK_MAX_MEMORY_TYPES];
    uint32_t *memoryHeapCount*;
    VkMemoryHeap *memoryHeaps*[VK_MAX_MEMORY_HEAPS];
} VkPhysicalDeviceMemoryProperties;

typedef struct **VkMemoryType** {
    VkMemoryPropertyFlags *propertyFlags*;
    uint32_t *heapIndex*;
} VkMemoryType;

propertyFlags: VK_MEMORY_PROPERTY_*X*_BIT where *X* is DEVICE_LOCAL, HOST_VISIBLE, HOST_COHERENT, HOST_CACHED, LAZILY_ALLOCATED

typedef struct **VkMemoryHeap** {
    VkDeviceSize *size*;
    VkMemoryHeapFlags *flags*;
} VkMemoryHeap;

flags: VK_MEMORY_HEAP_DEVICE_LOCAL_BIT

VkResult **vkAllocateMemory**(
    VkDevice *device*,
    const VkMemoryAllocateInfo* *pAllocateInfo*,
    const VkAllocationCallbacks* *pAllocator*, P.10
    VkDeviceMemory* *pMemory*);

typedef struct **VkMemoryAllocateInfo** {
    VkStructureType *sType*;
    const void* *pNext*;
    VkDeviceSize* *allocationSize*;
    uint32_t *memoryTypeIndex*;
} VkMemoryAllocateInfo;

void **vkFreeMemory**(
    VkDevice *device*,
    VkDeviceMemory *memory*,
    const VkAllocationCallbacks* *pAllocator*); P.10

Host Access to Device Memory Objects [10.2.1]

Memory objects created with vkAllocateMemory are not directly host accessible. Memory objects created with memory property VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT are considered mappable. Memory objects must be mappable in order to be successfully mapped on the host.

VkResult **vkMapMemory**(
    VkDevice *device*,
    VkDeviceMemory *memory*,
    VkDeviceSize *offset*,
    VkDeviceSize *size*,
    VkMemoryMapFlags *flags*, = 0
    void** *ppData*);

VkResult **vkFlushMappedMemoryRanges**(
    VkDevice *device*,
    uint32_t *memoryRangeCount*,
    const VkMappedMemoryRange* *pMemoryRanges*);

VkResult **vkInvalidateMappedMemoryRanges**(
    VkDevice *device*,
    uint32_t *memoryRangeCount*,
    const VkMappedMemoryRange* *pMemoryRanges*);

typedef struct **VkMappedMemoryRange** {
    VkStructureType *sType*;
    const void* *pNext*;
    VkDeviceMemory *memory*;
    VkDeviceSize *offset*;
    VkDeviceSize *size*;
} VkMappedMemoryRange;

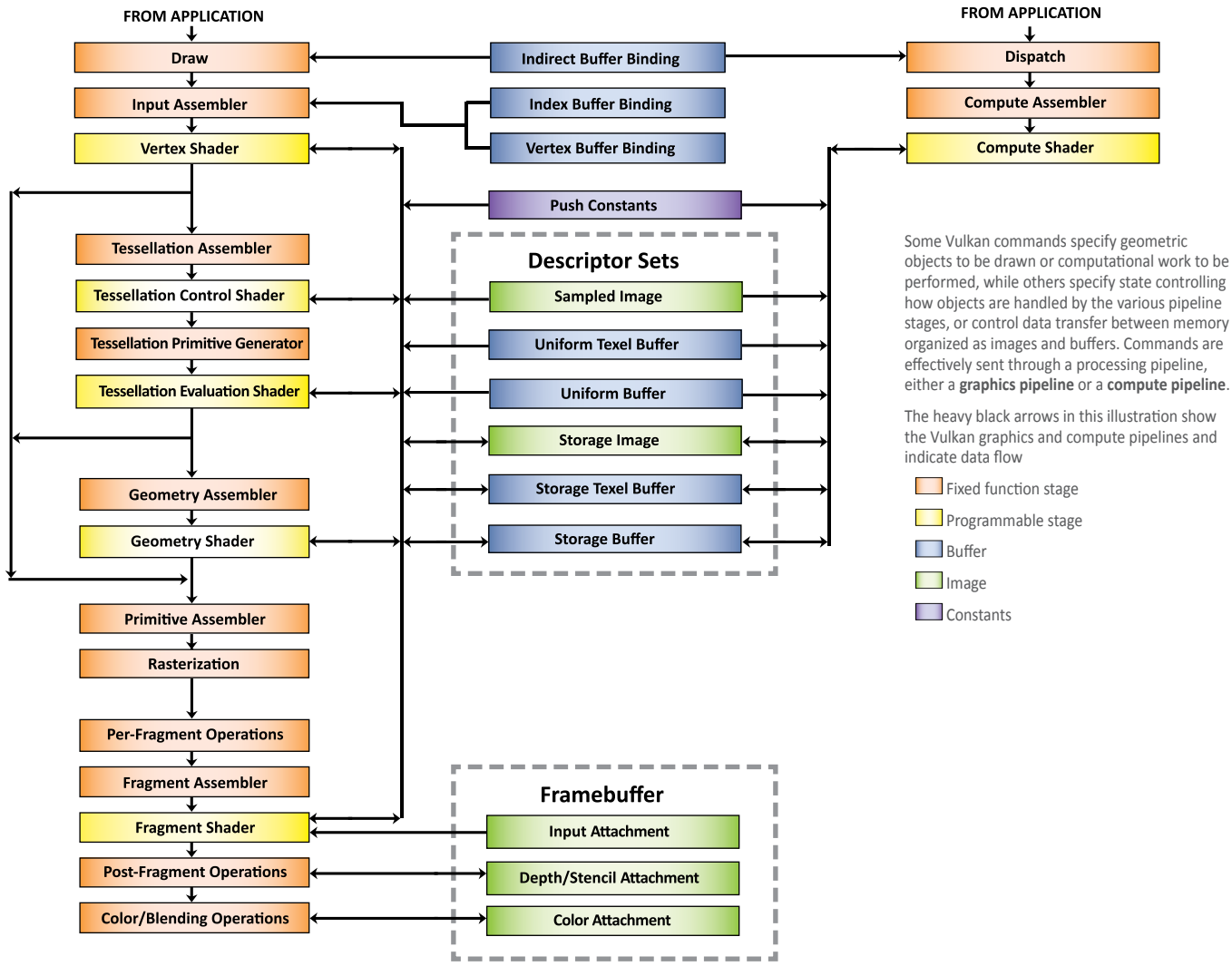
void **vkUnmapMemory**(
    VkDevice *device*,
    VkDeviceMemory *memory*);

Lazily Allocated Memory [10.2.2]

If the memory object is allocated from a heap with the VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT bit set, that object's backing memory may be provided by the implementation lazily.

void **vkGetDeviceMemoryCommitment**(
    VkDevice *device*,
    VkDeviceMemory *memory*,
    VkDeviceSize* *pCommittedMemoryInBytes*);

Vulkan Pipeline Diagram [9]



Resource Creation [11]

Buffers [11.1]

Buffers represent linear arrays of data which are used for various purposes by binding them to the graphics pipeline via descriptor sets or via certain commands, or by directly specifying them as parameters to certain commands.

```
VkResult vkCreateBuffer(
    VkDevice device,
    const VkBufferCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.10
    VkBuffer* pBuffer);

typedef struct VkBufferCreateInfo {
    VkStructureType sType;
    const void* pNext;
    VkBufferCreateFlags flags;
    VkDeviceSize size;
    VkBufferUsageFlags usage;
    VkSharingMode sharingMode; P.12
    uint32_t queueFamilyIndexCount;
    const uint32_t* pQueueFamilyIndices;
} VkBufferCreateInfo;
```

flags:
VK_BUFFER_CREATE_SPARSE_X_BIT where X is BINDING, RESIDENCY, ALIASED

usage:
VK_BUFFER_USAGE_X_BIT where X is TRANSFER_SRC, TRANSFER_DST, UNIFORM_TEXEL_BUFFER, STORAGE_TEXEL_BUFFER, UNIFORM_BUFFER, STORAGE_BUFFER, INDEX_BUFFER, VERTEX_BUFFER, INDIRECT_BUFFER

```
void vkDestroyBuffer(
    VkDevice device,
    VkBuffer buffer,
    const VkAllocationCallbacks* pAllocator); P.10
```

Buffer Views [11.2]

A buffer view represents a contiguous range of a buffer and a specific format to be used to interpret the data.

```
VkResult vkCreateBufferView(
    VkDevice device,
    const VkBufferViewCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.10
    VkBufferView* pView);

typedef struct VkBufferViewCreateInfo {
    VkStructureType sType;
    const void* pNext;
    VkBufferViewCreateFlags flags; = 0
    VkBuffer buffer;
    VkFormat format; P.11
    VkDeviceSize offset;
    VkDeviceSize range;
} VkBufferViewCreateInfo;

void vkDestroyBufferView(
    VkDevice device,
    VkBufferView bufferView,
    const VkAllocationCallbacks* pAllocator); P.10
```

Images [11.3]

Images represent multidimensional (up to 3) arrays of data which can be used for various purposes by binding them to the graphics pipeline via descriptor sets, or by directly specifying them as parameters to certain commands.

```
VkResult vkCreateImage(
    VkDevice device,
    const VkImageCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.10
    VkImage* pImage);

typedef struct VkImageCreateInfo {
    VkStructureType sType;
    const void* pNext;
    VkImageCreateFlags flags; P.11
    VkImageType imageType; P.11
    VkFormat format; P.11
    VkExtent3D extent; P.10
    uint32_t mipLevels;
    uint32_t arrayLayers;
    VkSampleCountFlagBits samples; P.12
    VkImageTiling tiling; P.11
    VkImageUsageFlags usage; P.11
    VkSharingMode sharingMode; P.12
    uint32_t queueFamilyIndexCount;
    const uint32_t* pQueueFamilyIndices;
    VkImageLayout initialLayout;
} VkImageCreateInfo;
```

initialLayout:
VK_IMAGE_LAYOUT_{PREINITIALIZED , UNDEFINED}

```
void vkGetImageSubresourceLayout(
    VkDevice device,
    VkImage image,
    const VkImageSubresource* pSubresource,
    VkSubresourceLayout* pLayout);

typedef struct VkImageSubresource {
    VkImageAspectFlags aspectMask; P.11
    uint32_t mipLevel;
    uint32_t arrayLayer;
} VkImageSubresource;

typedef struct VkSubresourceLayout {
    VkDeviceSize offset;
    VkDeviceSize size;
    VkDeviceSize rowPitch;
    VkDeviceSize arrayPitch;
    VkDeviceSize depthPitch;
} VkSubresourceLayout;
```

Continued on next page >

Resource Creation (continued)

```
void vkDestroyImage(
    VkDevice device,
    VkImage image,
    const VkAllocationCallbacks* pAllocator); P.10
```

Image Views [11.5]

Image objects are not directly accessed by pipeline shaders for reading or writing image data. Instead, image views representing contiguous ranges of the image subresources and containing additional metadata are used for that purpose.

```
VkResult vkCreateImageView(
    VkDevice device,
    const VkImageViewCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.10
    VkImageView* pView);

typedef struct VkImageViewCreateInfo {
    VkStructureType sType;
    const void* pNext;
    VkImageViewCreateFlags flags; = 0
    VkImage image;
    VkImageViewType viewType;
    VkFormat format; P.11
    VkComponentMapping components;
    VkImageSubresourceRange subresourceRange; P.11
} VkImageViewCreateInfo;
```

viewType: VK_IMAGE_VIEW_TYPE_*X* where *X* is 1D, 2D, 3D, CUBE, 1D_ARRAY, 2D_ARRAY, CUBE_ARRAY

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

enum VkComponentSwizzle: VK_COMPONENT_SWIZZLE_*X* where *X* is IDENTITY, ZERO, ONE, R, G, B, A

```
void vkDestroyImageView(
    VkDevice device,
    VkImageView imageView,
    const VkAllocationCallbacks* pAllocator); P.10
```

Resource Memory Association [11.6]

Resources are initially created as virtual allocations with no backing memory. Device memory is allocated separately and then associated with the resource.

```
void vkGetBufferMemoryRequirements(
    VkDevice device,
    VkBuffer buffer,
    VkMemoryRequirements* pMemoryRequirements);

void vkGetImageMemoryRequirements(
    VkDevice device,
    VkImage image,
    VkMemoryRequirements* pMemoryRequirements);

typedef struct VkMemoryRequirements {
    VkDeviceSize size;
    VkDeviceSize alignment;
    uint32_t memoryTypeBits;
} VkMemoryRequirements;

VkResult vkBindBufferMemory(
    VkDevice device,
    VkBuffer buffer,
    VkDeviceMemory memory,
    VkDeviceSize memoryOffset);

VkResult vkBindImageMemory(
    VkDevice device,
    VkImage image,
    VkDeviceMemory memory,
    VkDeviceSize memoryOffset);
```

Samplers [12]

VkSampler objects encapsulate the state of an image sampler which is used by the implementation to read image data and apply filtering and other transformations for the shader.

```
VkResult vkCreateSampler(
    VkDevice device,
    const VkSamplerCreateInfo* pCreateInfo,
    const VkAllocationCallbacks *pAllocator, P.10
    VkSampler *pSampler);

typedef struct VkSamplerCreateInfo {
    VkStructureType sType;
    const void *pNext;
    VkSamplerCreateFlags flags; = 0
    VkFilter magFilter;
    VkFilter minFilter;
    VkSamplerMipmapMode mipmapMode;
    VkSamplerAddressMode addressModeU;
    VkSamplerAddressMode addressModeV;
    VkSamplerAddressMode addressModeW;
    float mipLodBias;
    VkBool32 anisotropyEnable;
    float maxAnisotropy;
    VkBool32 compareEnable;
    VkCompareOp compareOp; P.11
    float minLod;
    float maxLod;
    VkBorderColor borderColor;
    VkBool32 unnormalizedCoordinates;
} VkSamplerCreateInfo;
```

magFilter, minFilter: VK_FILTER_NEAREST, VK_FILTER_LINEAR

mipmapMode: VK_SAMPLER_MIPMAP_MODE_{NEAREST, LINEAR}

borderColor: VK_BORDER_COLOR_{FLOAT, INT}_*X* where *X* is TRANSPARENT_BLACK, OPAQUE_BLACK, OPAQUE_WHITE

addressMode{U, V, W}: VK_SAMPLER_ADDRESS_MODE_*X* where *X* is REPEAT, MIRRORED_REPEAT, CLAMP_TO_EDGE, CLAMP_TO_BORDER

```
void vkDestroySampler(
    VkDevice device,
    VkSampler sampler,
    const VkAllocationCallbacks *pAllocator); P.10
```

Resource Descriptors [13]

A descriptor is an opaque data structure representing a shader resource such as a buffer view, image view, sampler, or combined image sampler.

Descriptor Set Layout [13.2.1]

```
VkResult vkCreateDescriptorSetLayout(
    VkDevice device,
    const VkDescriptorSetLayoutCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.10
    VkDescriptorSetLayout* pSetLayout);

typedef struct VkDescriptorSetLayoutCreateInfo {
    VkStructureType sType;
    const void *pNext;
    VkDescriptorSetLayoutCreateFlags flags; = 0
    uint32_t bindingCount;
    const VkDescriptorSetLayoutBinding* pBinding;
} VkDescriptorSetLayoutCreateInfo;

typedef struct VkDescriptorSetLayoutBinding {
    uint32_t binding;
    VkDescriptorType descriptorType; P.11
    uint32_t descriptorCount;
    VkShaderStageFlags stageFlags; P.12
    const VkSampler* pImmutableSamplers;
} VkDescriptorSetLayoutBinding;

void vkDestroyDescriptorSetLayout(
    VkDevice device,
    VkDescriptorSetLayout descriptorSetLayout,,
    const VkAllocationCallbacks *pAllocator); P.10
```

Pipeline Layouts [13.2.2]

```
VkResult vkCreatePipelineLayout(
    VkDevice device,
    const VkPipelineLayoutCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.10
    VkPipelineLayout* pPipelineLayout);

typedef struct VkPipelineLayoutCreateInfo {
    VkStructureType sType;
    const void* pNext;
    VkPipelineLayoutCreateFlags flags; = 0
    uint32_t setLayoutCount;
    const VkDescriptorSetLayout* pSetLayouts;
    uint32_t pushConstantRangeCount;
    const VkPushConstantRange* pPushConstantRanges;
} VkPipelineLayoutCreateInfo;

typedef struct VkPushConstantRange {
    VkShaderStageFlags stageFlags; P.12
    uint32_t offset;
    uint32_t size;
} VkPushConstantRange;

void vkDestroyPipelineLayout(
    VkDevice device,
    VkPipelineLayout pipelineLayout,
    const VkAllocationCallbacks* pAllocator); P.10
```

Allocation of Descriptor Sets [13.2.3]

```
VkResult vkCreateDescriptorPool(
    VkDevice device,
    const VkDescriptorPoolCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.10
    VkDescriptorPool* pDescriptorPool);

typedef struct VkDescriptorPoolCreateInfo {
    VkStructureType sType;
    const void* pNext;
    VkDescriptorPoolCreateFlags flags;
    uint32_t maxSets;
    uint32_t poolSizeCount;
    const VkDescriptorPoolSize* pPoolSizes;
} VkDescriptorPoolCreateInfo;
```

flags: VK_DESCRIPTOR_POOL_CREATE_FREE_-DESCRIPTOR_SET_BIT

```
typedef struct VkDescriptorPoolSize {
    VkDescriptorType type; P.11
    uint32_t descriptorCount;
} VkDescriptorPoolSize;

void vkDestroyDescriptorPool(
    VkDevice device,
    VkDescriptorPool descriptorPool,
    const VkAllocationCallbacks* pAllocator); P.10

VkResult vkAllocateDescriptorSets(
    VkDevice device,
    const VkDescriptorSetAllocateInfo* pAllocateInfo,
    VkDescriptorSet* pDescriptorSets);

typedef struct VkDescriptorSetAllocateInfo {
    VkStructureType sType;
    const void* pNext;
    VkDescriptorPool descriptorPool;
    uint32_t descriptorSetCount;
    const VkDescriptorSetLayout* pSetLayouts;
} VkDescriptorSetAllocateInfo;

VkResult vkFreeDescriptorSets(
    VkDevice device,
    VkDescriptorPool descriptorPool,
    uint32_t descriptorSetCount,
    const VkDescriptorSet* pDescriptorSets);

VkResult vkResetDescriptorPool(
    VkDevice device,
    VkDescriptorPool descriptorPool,
    VkDescriptorPoolResetFlags flags);
```

Descriptor Set Updates [13.2.4]

```
void vkUpdateDescriptorSets(
    VkDevice device,
    uint32_t descriptorWriteCount,
    const VkWriteDescriptorSet* pDescriptorWrites,
    uint32_t descriptorCopyCount,
    const VkCopyDescriptorSet* pDescriptorCopies);

typedef struct VkWriteDescriptorSet {
    VkStructureType sType;
    const void* pNext;
    VkDescriptorSet dstSet;
    uint32_t dstBinding;
    uint32_t dstArrayElement;
    uint32_t descriptorCount;
    VkDescriptorType descriptorType; P.11
    const VkDescriptorImageInfo* pImageInfo;
    const VkDescriptorBufferInfo* pBufferInfo;
    const VkBufferView* pTexelBufferView;
} VkWriteDescriptorSet;

typedef struct VkDescriptorImageInfo {
    VkSampler sampler;
    VkImageView imageView;
    VkImageLayout imageLayout; P.11
} VkDescriptorImageInfo;

typedef struct VkDescriptorBufferInfo {
    VkBuffer buffer;
    VkDeviceSize offset;
    VkDeviceSize range;
} VkDescriptorBufferInfo;

typedef struct VkCopyDescriptorSet {
    VkStructureType sType;
    const void* pNext;
    VkDescriptorSet srcSet;
    uint32_t srcBinding;
    uint32_t srcArrayElement;
    VkDescriptorSet dstSet;
    uint32_t dstBinding;
    uint32_t dstArrayElement;
    uint32_t descriptorCount;
} VkCopyDescriptorSet;
```

Continued on next page >

Resource Descriptors (continued)

Descriptor Set Binding [13.2.5]

```
void vkCmdBindDescriptorSets(
    VkCommandBuffer commandBuffer,
    VkPipelineBindPoint pipelineBindPoint,
    VkPipelineLayout layout, P.12
    uint32_t firstSet,
    uint32_t descriptorSetCount,
    const VkDescriptorSet* pDescriptorSets,
    uint32_t dynamicOffsetCount,
    const uint32_t* pDynamicOffsets);
```

pipelineBindPoint:
VK_PIPELINE_BIND_POINT_GRAPHICS,
VK_PIPELINE_BIND_POINT_COMPUTE

Push Constant Updates [13.2.6]

The pipeline layout defines shader push constants which are updated via Vulkan commands rather than via writes to memory or copy commands.

```
void vkCmdPushConstants(
    VkCommandBuffer commandBuffer,
    VkPipelineLayout layout, P.12
    VkShaderStageFlags stageFlags, P.12
    uint32_t offset,
    uint32_t size,
    const void* pValues);
```

Clear Commands [17]

Outside a Render Pass Instance [17.1]

```
void vkCmdClearColorImage(
    VkCommandBuffer commandBuffer,
    VkImage image,
    VkImageLayout imageLayout,
    const VkClearColorValue* pColor, P.10
    uint32_t rangeCount,
    const VkImageSubresourceRange* pRanges); P.11
```

imageLayout:
VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL,
VK_IMAGE_LAYOUT_GENERAL

```
void vkCmdClearDepthStencilImage(
    VkCommandBuffer commandBuffer,
    VkImage image,
    VkImageLayout imageLayout,
    const VkClearDepthStencilValue* pDepthStencil, P.10
    uint32_t rangeCount,
    const VkImageSubresourceRange* pRanges); P.11
```

imageLayout:
VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL,
VK_IMAGE_LAYOUT_GENERAL

Inside a Render Pass Instance [17.2]

```
void vkCmdClearAttachments(
    VkCommandBuffer commandBuffer,
    uint32_t attachmentCount,
    const VkClearAttachment* pAttachments,
    uint32_t rectCount,
    const VkClearRect* pRects);

typedef struct VkClearRect {
    VkRect2D rect; P.12
    uint32_t baseArrayLayer;
    uint32_t layerCount;
} VkClearRect;

typedef struct VkClearAttachment {
    VkImageAspectFlags aspectMask; P.11
    uint32_t colorAttachment;
    VkClearValue clearValue; P.10
} VkClearAttachment;
```

Filling Buffers [17.4]

```
void vkCmdFillBuffer(
    VkCommandBuffer commandBuffer,
    VkBuffer dstBuffer,
    VkDeviceSize dstOffset,
    VkDeviceSize size,
    uint32_t data);
```

Updating Buffers [17.5]

```
void vkCmdUpdateBuffer(
    VkCommandBuffer commandBuffer,
    VkBuffer dstBuffer,
    VkDeviceSize dstOffset,
    VkDeviceSize dataSize,
    const uint32_t* pData);
```

Queries [16]

Query Pools [16.1]

Each query pool is a collection of a specific number of queries of a particular type.

```
VkResult vkCreateQueryPool(
    VkDevice device,
    const VkQueryPoolCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.10
    VkQueryPool* pQueryPool);

typedef struct VkQueryPoolCreateInfo {
    VkStructureType sType;
    const void* pNext;
    VkQueryPoolCreateFlags flags; = 0
    VkQueryType queryType;
    uint32_t entryCount;
    VkQueryPipelineStatisticFlags pipelineStatistics; P.12
} VkQueryPoolCreateInfo;
```

queryType:
VK_QUERY_TYPE_OCCLUSION,
VK_QUERY_TYPE_PIPELINE_STATISTICS,
VK_QUERY_TYPE_TIMESTAMP

```
void vkDestroyQueryPool(
    VkDevice device,
    VkQueryPool queryPool,
    const VkAllocationCallbacks* pAllocator); P.10
```

Query Operation [16.2]

```
void vkCmdResetQueryPool(
    VkCommandBuffer commandBuffer,
    VkQueryPool queryPool,
    uint32_t firstQuery,
    uint32_t queryCount);

void vkCmdBeginQuery(
    VkCommandBuffer commandBuffer,
    VkQueryPool queryPool,
    uint32_t entry,
    VkQueryControlFlags flags);
```

flags: VK_QUERY_CONTROL_PRECISE_BIT

```
void vkCmdEndQuery(
    VkCommandBuffer commandBuffer,
    VkQueryPool queryPool,
    uint32_t query);

VkResult vkGetQueryPoolResults(
    VkDevice device,
    VkQueryPool queryPool,
    uint32_t firstQuery,
    uint32_t queryCount,
    size_t dataSize,
    void* pData,
    VkDeviceSize stride,
    VkQueryResultFlags flags);
```

flags: VK_QUERY_RESULT_X_BIT where *X* is 64, WAIT, WITH_AVAILABILITY, PARTIAL

```
void vkCmdCopyQueryPoolResults(
    VkCommandBuffer commandBuffer,
    VkQueryPool queryPool,
    uint32_t firstQuery,
    uint32_t queryCount,
    VkBuffer dstBuffer,
    VkDeviceSize dstOffset,
    VkDeviceSize stride,
    VkQueryResultFlags flags);
```

flags: VK_QUERY_RESULT_X_BIT where *X* is 64, WAIT, WITH_AVAILABILITY, PARTIAL

Timestamp Queries [16.5]

```
void vkCmdWriteTimestamp(
    VkCommandBuffer commandBuffer,
    VkPipelineStageFlagBits pipelineStage, P.15
    VkQueryPool queryPool,
    uint32_t query);
```

Copy Commands [18]

Copying Data Between Buffers [18.2]

```
void vkCmdCopyBuffer(
    VkCommandBuffer commandBuffer,
    VkBuffer srcBuffer,
    VkBuffer dstBuffer,
    uint32_t regionCount,
    const VkBufferCopy* pRegions);

typedef struct VkBufferCopy {
    VkDeviceSize srcOffset;
    VkDeviceSize dstOffset;
    VkDeviceSize size;
} VkBufferCopy;
```

Copying Data Between Images [18.3]

```
void vkCmdCopyImage(
    VkCommandBuffer commandBuffer,
    VkImage srcImage,
    VkImageLayout srcImageLayout,
    VkImage dstImage,
    VkImageLayout dstImageLayout,
    uint32_t regionCount,
    const VkImageCopy* pRegions);
```

enum VkImageLayout: VK_IMAGE_LAYOUT_GENERAL, VK_IMAGE_LAYOUT_TRANSFER_{SRC, DST}_OPTIMAL

```
typedef struct VkImageCopy {
    VkImageSubresourceLayers srcSubresource; P.11
    VkOffset3D srcOffset; P.11
    VkImageSubresourceLayers dstSubresource; P.11
    VkOffset3D dstOffset; P.11
    VkExtent3D extent; P.10
} VkImageCopy;
```

Copying Data Between Buffers and Images [18.4]

```
void vkCmdCopyBufferToImage(
    VkCommandBuffer commandBuffer,
    VkBuffer srcBuffer,
    VkImage dstImage,
    VkImageLayout dstImageLayout,
    uint32_t regionCount,
    const VkBufferImageCopy* pRegions);
```

dstImageLayout: VK_IMAGE_LAYOUT_GENERAL, VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL

```
void vkCmdCopyImageToBuffer(
    VkCommandBuffer commandBuffer,
    VkImage srcImage,
    VkImageLayout srcImageLayout,
    VkBuffer dstBuffer,
    uint32_t regionCount,
    const VkBufferImageCopy* pRegions);
```

srcImageLayout: VK_IMAGE_LAYOUT_GENERAL, VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL

```
typedef struct VkBufferImageCopy {
    VkDeviceSize bufferOffset;
    uint32_t bufferRowLength;
    uint32_t bufferImageHeight;
    VkImageSubresourceLayers imageSubresource; P.11
    VkOffset3D imageOffset; P.11
    VkExtent3D imageExtent; P.10
} VkBufferImageCopy;
```

Image Copies With Scaling [18.5]

```
void vkCmdBlitImage(
    VkCommandBuffer commandBuffer,
    VkImage srcImage,
    VkImageLayout srcImageLayout,
    VkImage dstImage,
    VkImageLayout dstImageLayout,
    uint32_t regionCount,
    const VkImageBlit* pRegions,
    VkFilter filter);
```

enum VkImageLayout: VK_IMAGE_LAYOUT_GENERAL, VK_IMAGE_LAYOUT_TRANSFER_{SRC, DST}_OPTIMAL

filter: VK_FILTER_NEAREST, VK_FILTER_LINEAR

```
typedef struct VkImageBlit {
    VkImageSubresourceLayers srcSubresource; P.11
    VkOffset3D srcOffsets[2]; P.11
    VkImageSubresourceLayers dstSubresource; P.11
    VkOffset3D dstOffsets[2]; P.11
} VkImageBlit;
```

Resolving Multisample Images [18.6]

```
void vkCmdResolveImage(
    VkCommandBuffer commandBuffer,
    VkImage srcImage,
    VkImageLayout srcImageLayout,
    VkImage dstImage,
    VkImageLayout dstImageLayout,
    uint32_t regionCount,
    const VkImageResolve* pRegions);
```

enum VkImageLayout: VK_IMAGE_LAYOUT_GENERAL, VK_IMAGE_LAYOUT_TRANSFER_{SRC, DST}_OPTIMAL

```
typedef struct VkImageResolve {
    VkImageSubresourceLayers srcSubresource; P.11
    VkOffset3D srcOffset; P.11
    VkImageSubresourceLayers dstSubresource; P.11
    VkOffset3D dstOffset; P.11
    VkExtent3D extent; P.10
} VkImageResolve;
```

Drawing Commands [19]

```
void vkCmdBindIndexBuffer(
    VkCommandBuffer commandBuffer,
    VkBuffer buffer,
    VkDeviceSize offset,
    VkIndexType indexType);
```
indexType: VK_INDEX_TYPE_UINT{16, 32}

```
void vkCmdDraw(
    VkCommandBuffer commandBuffer,
    uint32_t vertexCount,
    uint32_t instanceCount,
    uint32_t firstVertex,
    uint32_t firstInstance);
```

```
void vkCmdDrawIndexed(
    VkCommandBuffer commandBuffer,
    uint32_t indexCount,
    uint32_t instanceCount,
    uint32_t firstIndex,
    int32_t vertexOffset,
    uint32_t firstInstance);
```

```
void vkCmdDrawIndirect(
    VkCommandBuffer commandBuffer,
    VkBuffer buffer,
    VkDeviceSize offset,
    uint32_t drawCount,
    uint32_t stride);
```

```
typedef struct VkDrawIndirectCommand {
    uint32_t vertexCount;
    uint32_t instanceCount;
    uint32_t firstVertex;
    uint32_t firstInstance;
} VkDrawIndirectCommand;
```

```
void vkCmdDrawIndexedIndirect(
    VkCommandBuffer commandBuffer,
    VkBuffer buffer,
    VkDeviceSize offset,
    uint32_t drawCount,
    uint32_t stride);
```

```
typedef struct VkDrawIndexedIndirectCommand {
    uint32_t indexCount;
    uint32_t instanceCount;
    uint32_t firstIndex;
    int32_t vertexOffset;
    uint32_t firstInstance;
} VkDrawIndexedIndirectCommand;
```

Vertex Input Description [20.2]

```
void vkCmdBindVertexBuffers(
    VkCommandBuffer commandBuffer,
    uint32_t firstBinding,
    uint32_t bindingCount,
    const VkBuffer* pBuffers,
    const VkDeviceSize* pOffsets);
```

Fixed-Function Vertex Postprocessing [23]

Controlling the Viewport [23.5]

```
void vkCmdSetViewport(
    VkCommandBuffer commandBuffer,
    uint32_t firstViewport,
    uint32_t viewportCount,
    const VkViewport* pViewports); P.11
```

Rasterization [24]

Basic Line Segment Rasterization [24.5.1]

```
void vkCmdSetLineWidth(
    VkCommandBuffer commandBuffer,
    float lineWidth);
```

Depth Bias [24.6.3]

```
void vkCmdSetDepthBias(
    VkCommandBuffer commandBuffer,
    float depthBiasConstantFactor,
    float depthBiasClamp,
    float depthBiasSlopeFactor);
```

Framebuffer: Blend Factors [26.1.1]

```
void vkCmdSetBlendConstants(
    VkCommandBuffer commandBuffer,
    const float blendConstants[4]);
```

Fragment Operations [25]

Scissor Test [25.2]

```
void vkCmdSetScissor(
    VkCommandBuffer commandBuffer,
    uint32_t firstScissor,
    uint32_t scissorCount,
    const VkRect2D* pScissors); P.12
```

Depth Bounds Test [25.8]

```
void vkCmdSetDepthBounds(
    VkCommandBuffer commandBuffer,
    float minDepthBounds,
    float maxDepthBounds);
```

Stencil Test [25.9]

```
void vkCmdSetStencilCompareMask(
    VkCommandBuffer commandBuffer,
    VkStencilFaceFlags faceMask,
    uint32_t compareMask);
```

```
void vkCmdSetStencilWriteMask(
    VkCommandBuffer commandBuffer,
    VkStencilFaceFlags faceMask,
    uint32_t writeMask);
```

```
void vkCmdSetStencilReference(
    VkCommandBuffer commandBuffer,
    VkStencilFaceFlags faceMask,
    uint32_t reference);
```
faceMask:
VK_STENCIL_FACE_{FRONT, BACK}_BIT,
VK_STENCIL_FRONT_AND_BACK

Sparse Resources [28]

Sparse Image Format Properties [28.7.3]

```
void vkGetPhysicalDeviceSparseImageFormatProperties(
    VkPhysicalDevice physicalDevice,
    VkFormat format, P.11
    VkImageType type, P.11
    VkSampleCountFlagBits samples, P.12
    VkImageUsageFlags usage, P.11
    VkImageTiling tiling, P.11
    uint32_t* pPropertyCount,
    VkSparseImageFormatProperties* pProperties);
```

```
typedef struct VkSparseImageFormatProperties {
    VkImageAspectFlags aspectMask; P.11
    VkExtent3D imageGranularity; P.11
    VkSparseImageFormatFlags flags;
} VkSparseImageFormatProperties;
```
flags: VK_SPARSE_IMAGE_FORMAT_*X* where *X* is
SINGLE_MIPTAIL_BIT, ALIGNED_MIP_SIZE_BIT,
NONSTANDARD_BLOCK_SIZE_BIT

Sparse Resource Memory Requirements [28.7.5]

```
void vkGetImageSparseMemoryRequirements(
    VkDevice device,
    VkImage image,
    uint32_t* pSparseMemoryRequirementCount,
    VkSparseImageMemoryRequirements*
        pSparseMemoryRequirements);
```

```
typedef struct VkSparseImageMemoryRequirements {
    VkSparseImageFormatProperties formatProperties;
    uint32_t imageMipTailFirstLod;
    VkDeviceSize imageMipTailSize;
    VkDeviceSize imageMipTailOffset;
    VkDeviceSize imageMipTailStride;
} VkSparseImageMemoryRequirements;
```

Binding Resource Memory [28.7.6]

```
typedef struct VkBindSparseInfo {
    VkStructureType sType;
    const void* pNext;
    uint32_t waitSemaphoreCount;
    const VkSemaphore* pWaitSemaphores;
    uint32_t bufferBindCount;
    const VkSparseBufferMemoryBindInfo* pBufferBinds;
    uint32_t imageOpaqueBindCount;
    const VkSparseImageOpaqueMemoryBindInfo*
        pImageOpaqueBinds;
    uint32_t imageBindCount;
    const VkSparseImageMemoryBindInfo* pImageBinds;
    uint32_t signalSemaphoreCount;
    const VkSemaphore* pSignalSemaphores;
} VkBindSparseInfo;
```

```
typedef struct VkSparseBufferMemoryBindInfo {
    VkBuffer buffer;
    uint32_t bindCount;
    const VkSparseMemoryBind* pBinds; P.12
} VkSparseBufferMemoryBindInfo;
```

```
typedef struct VkSparseImageOpaqueMemoryBindInfo {
    VkImage image;
    uint32_t bindCount;
    const VkSparseMemoryBind* pBinds; P.12
} VkSparseImageOpaqueMemoryBindInfo;
```

```
typedef struct VkSparseImageMemoryBindInfo {
    VkImage image;
    uint32_t bindCount;
    const VkSparseImageMemoryBind* pBinds;
} VkSparseImageMemoryBindInfo;
```

```
typedef struct VkSparseImageMemoryBind {
    VkImageSubresource subresource;
    VkOffset3D offset; P.11
    VkExtent3D extent; P.11
    VkDeviceMemory memory;
    VkDeviceSize memoryOffset;
    VkSparseMemoryBindFlags flags;
} VkSparseImageMemoryBind;
```
flags:
VK_SPARSE_MEMORY_BIND_METADATA_BIT

```
VkResult vkQueueBindSparse(
    VkQueue queue,
    uint32_t bindInfoCount,
    const VkBindSparseInfo* pBindInfo,
    VkFence fence);
```

Dispatching Commands [27]

```
void vkCmdDispatch(
    VkCommandBuffer commandBuffer,
    uint32_t x,
    uint32_t y,
    uint32_t z);
```

```
void vkCmdDispatchIndirect(
    VkCommandBuffer commandBuffer,
    VkBuffer buffer,
    VkDeviceSize offset);
```

```
typedef struct VkDispatchIndirectCommand {
    uint32_t x;
    uint32_t y;
    uint32_t z;
} VkDispatchIndirectCommand;
```

Window System Integration (WSI) [29]

Android Platform [29.2.1]

```
VkResult vkCreateAndroidSurfaceKHR(
    VkInstance instance,
    const VkAndroidSurfaceCreateInfoKHR* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.10
    VkSurfaceKHR* pSurface);

typedef struct VkAndroidSurfaceCreateInfoKHR {
    VkStructureType sType;
    const void* pNext;
    VkAndroidSurfaceCreateFlagsKHR flags; = 0
    ANativeWindow* window;
} VkAndroidSurfaceCreateInfoKHR;
```

Mir Platform [29.2.2]

```
VkResult vkCreateMirSurfaceKHR(
    VkInstance instance,
    const VkMirSurfaceCreateInfoKHR* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.10
    VkSurfaceKHR* pSurface);

typedef struct VkMirSurfaceCreateInfoKHR {
    VkStructureType sType;
    const void* pNext;
    VkMirSurfaceCreateFlagsKHR flags; = 0
    MirConnection* connection;
    MirSurface* mirSurface;
} VkMirSurfaceCreateInfoKHR;
```

Wayland Platform [29.2.3]

```
VkResult vkCreateWaylandSurfaceKHR(
    VkInstance instance,
    const VkWaylandSurfaceCreateInfoKHR* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.10
    VkSurfaceKHR* pSurface);

typedef struct VkWaylandSurfaceCreateInfoKHR {
    VkStructureType sType;
    const void* pNext;
    VkWaylandSurfaceCreateFlagsKHR flags; = 0
    struct wl_display* display;
    struct wl_surface* surface;
} VkWaylandSurfaceCreateInfoKHR;
```

Win32 Platform [29.2.4]

```
VkResult vkCreateWin32SurfaceKHR(
    VkInstance instance,
    const VkWin32SurfaceCreateInfoKHR* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.10
    VkSurfaceKHR* pSurface);

typedef struct VkWin32SurfaceCreateInfoKHR {
    VkStructureType sType;
    const void* pNext;
    VkWin32SurfaceCreateFlagsKHR flags; = 0
    HINSTANCE hinstance;
    HWND hwnd;
} VkWin32SurfaceCreateInfoKHR;
```

XCB Platform [29.2.5]

```
VkResult vkCreateXcbSurfaceKHR(
    VkInstance instance,
    const VkXcbSurfaceCreateInfoKHR* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.10
    VkSurfaceKHR* pSurface);

typedef struct VkXcbSurfaceCreateInfoKHR {
    VkStructureType sType;
    const void* pNext;
    VkXcbSurfaceCreateFlagsKHR flags;
    xcb_connection_t* connection;
    xcb_window_t window;
} VkXcbSurfaceCreateInfoKHR;
```

Xlib Platform [29.2.6]

```
VkResult vkCreateXlibSurfaceKHR(
    VkInstance instance,
    const VkXlibSurfaceCreateInfoKHR* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.10
    VkSurfaceKHR* pSurface);

typedef struct VkXlibSurfaceCreateInfoKHR {
    VkStructureType sType;
    const void* pNext;
    VkXlibSurfaceCreateFlagsKHR flags;
    Display* dpy;
    Window window;
} VkXlibSurfaceCreateInfoKHR;
```

Platform-Independent Information [29.2.7]

```
void vkDestroySurfaceKHR(
    VkInstance instance,
    VkSurfaceKHR surface,
    const VkAllocationCallbacks* pAllocator); P.10
```

Display Enumeration [29.3.1]

```
VkResult vkGetPhysicalDeviceDisplayPropertiesKHR(
    VkPhysicalDevice physicalDevice,
    uint32_t* pPropertyCount,
    VkDisplayPropertiesKHR* pProperties);

typedef struct VkDisplayPropertiesKHR {
    VkDisplayKHR display;
    const char* displayName;
    VkExtent2D physicalDimensions; P.11
    VkExtent2D physicalResolution; P.11
    VkSurfaceTransformFlagsKHR supportedTransforms;
    VkBool32 planeReorderPossible;
    VkBool32 persistentContent;
} VkDisplayPropertiesKHR;
```

Display Planes [29.3.1.1]

```
VkResult vkGetPhysicalDeviceDisplayPlanePropertiesKHR(
    VkPhysicalDevice physicalDevice,
    uint32_t* pPropertyCount,
    VkDisplayPlanePropertiesKHR* pProperties);

typedef struct VkDisplayPlanePropertiesKHR {
    VkDisplayKHR currentDisplay;
    uint32_t currentStackIndex;
} VkDisplayPlanePropertiesKHR;

VkResult vkGetDisplayPlaneSupportedDisplaysKHR(
    VkPhysicalDevice physicalDevice,
    uint32_t planeIndex,
    uint32_t* pDisplayCount,
    VkDisplayKHR* pDisplays);
```

Display Modes [29.3.1.2]

```
VkResult vkGetDisplayModePropertiesKHR(
    VkPhysicalDevice physicalDevice,
    VkDisplayKHR display,
    uint32_t* pPropertyCount,
    VkDisplayModePropertiesKHR* pProperties);

typedef struct VkDisplayModePropertiesKHR {
    VkDisplayModeKHR displayMode;
    VkDisplayModeParametersKHR parameters;
} VkDisplayModePropertiesKHR;

typedef struct VkDisplayModeParametersKHR {
    VkExtent2D visibleRegion; P.11
    uint32_t refreshRate;
} VkDisplayModeParametersKHR;

VkResult vkCreateDisplayModeKHR(
    VkPhysicalDevice physicalDevice,
    VkDisplayKHR display,
    const VkDisplayModeCreateInfoKHR* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.10
    VkDisplayModeKHR* pMode);

typedef struct VkDisplayModeCreateInfoKHR {
    VkStructureType sType;
    const void* pNext;
    VkDisplayModeCreateFlagsKHR flags;
    VkDisplayModeParametersKHR parameters;
} VkDisplayModeCreateInfoKHR;

VkResult vkGetDisplayPlaneCapabilitiesKHR(
    VkPhysicalDevice physicalDevice,
    VkDisplayModeKHR mode,
    uint32_t planeIndex,
    VkDisplayPlaneCapabilitiesKHR* pCapabilities);

typedef struct VkDisplayPlaneCapabilitiesKHR {
    VkDisplayPlaneAlphaFlagsKHR supportedAlpha;
    VkOffset2D minSrcPosition; P.11
    VkOffset2D maxSrcPosition; P.11
    VkExtent2D minSrcExtent; P.11
    VkExtent2D maxSrcExtent; P.11
    VkOffset2D minDstPosition; P.11
    VkOffset2D maxDstPosition; P.11
    VkExtent2D minDstExtent; P.11
    VkExtent2D maxDstExtent; P.11
} VkDisplayPlaneCapabilitiesKHR;
```

Display Surfaces [29.3.2]

```
VkResult vkCreateDisplayPlaneSurfaceKHR(
    VkInstance instance,
    const VkDisplaySurfaceCreateInfoKHR* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.10
    VkSurfaceKHR* pSurface);

typedef struct VkDisplaySurfaceCreateInfoKHR {
    VkStructureType sType;
    const void* pNext;
    VkDisplaySurfaceCreateFlagsKHR flags;
    VkDisplayModeKHR displayMode;
    uint32_t planeIndex;
    uint32_t planeStackIndex;
    VkSurfaceTransformFlagBitsKHR transform;
    float globalAlpha;
    VkDisplayPlaneAlphaFlagBitsKHR alphaMode;
    VkExtent2D imageExtent; P.11
} VkDisplaySurfaceCreateInfoKHR;
```

Querying for WSI Support [29.4]

```
VkResult vkGetPhysicalDeviceSurfaceSupportKHR(
    VkPhysicalDevice physicalDevice,
    uint32_t queueFamilyIndex,
    VkSurfaceKHR surface,
    VkBool32* pSupported);
```

MIR Platform Querying [29.4.2]

```
VkBool32
    vkGetPhysicalDeviceMirPresentationSupportKHR(
    VkPhysicalDevice physicalDevice,
    uint32_t queueFamilyIndex,
    MirConnection* connection);
```

Wayland Platform Querying [29.4.3]

```
VkBool32
    vkGetPhysicalDeviceWaylandPresentationSupportKHR(
    VkPhysicalDevice physicalDevice,
    uint32_t queueFamilyIndex,
    struct wl_display* display);
```

Win32 Platform Querying [29.4.4]

```
VkBool32
    vkGetPhysicalDeviceWin32PresentationSupportKHR(
    VkPhysicalDevice physicalDevice,
    uint32_t queueFamilyIndex);
```

XCB Platform Querying [29.4.5]

```
VkBool32
    vkGetPhysicalDeviceXcbPresentationSupportKHR(
    VkPhysicalDevice physicalDevice,
    uint32_t queueFamilyIndex,
    xcb_connection_t* connection,
    xcb_visualid_t visual_id);
```

Xlib Platform Querying [29.4.6]

```
VkBool32
    vkGetPhysicalDeviceXlibPresentationSupportKHR(
    VkPhysicalDevice physicalDevice,
    uint32_t queueFamilyIndex,
    Display* dpy,
    VisualID visualID);
```

Surface Queries [29.5]

```
VkResult vkGetPhysicalDeviceSurfaceCapabilitiesKHR(
    VkPhysicalDevice physicalDevice,
    VkSurfaceKHR surface,
    VkSurfaceCapabilitiesKHR* pSurfaceCapabilities);

typedef struct VkSurfaceCapabilitiesKHR {
    uint32_t minImageCount;
    uint32_t maxImageCount;
    VkExtent2D currentExtent; P.11
    VkExtent2D minImageExtent; P.11
    VkExtent2D maxImageExtent; P.11
    uint32_t maxImageArrayLayers;
    VkSurfaceTransformFlagsKHR supportedTransforms;
    VkSurfaceTransformFlagBitsKHR currentTransform;
    VkCompositeAlphaFlagsKHR supportedCompositeAlpha; P.11
    VkImageUsageFlags supportedUsageFlags;
} VkSurfaceCapabilitiesKHR;

VkResult vkGetPhysicalDeviceSurfaceFormatsKHR(
    VkPhysicalDevice physicalDevice,
    VkSurfaceKHR surface,
    uint32_t* pSurfaceFormatCount,
    VkSurfaceFormatKHR* pSurfaceFormats);

typedef struct VkSurfaceFormatKHR {
    VkFormat format;
    VkColorSpaceKHR colorSpace;
} VkSurfaceFormatKHR;
```

colorSpace: VK_COLORSPACE_SRGB_NONLINEAR_KHR

```
VkResult vkGetPhysicalDeviceSurfacePresentModesKHR(
    VkPhysicalDevice physicalDevice,
    VkSurfaceKHR surface,
    uint32_t* pPresentModeCount,
    VkPresentModeKHR* pPresentModes);
```

pPresentModes: VK_PRESENT_MODE_*X*_KHR
where *X* is IMMEDIATE, MAILBOX, FIFO, FIFO_RELAXED

Continued on next page >

WSI (continued)

WSI Swapchain [29.6]

```
VkResult vkCreateSwapchainKHR(
    VkDevice device,
    const VkSwapchainCreateInfoKHR* pCreateInfo,
    const VkAllocationCallbacks* pAllocator, P.10
    VkSwapchainKHR* pSwapchain);

typedef struct VkSwapchainCreateInfoKHR {
    VkStructureType sType;
    const void* pNext;
    VkSwapchainCreateFlagsKHR flags;
    VkSurfaceKHR surface;
    uint32_t minImageCount;
    VkFormat imageFormat;
    VkColorSpaceKHR imageColorSpace;
    VkExtent2D imageExtent; P.11
    uint32_t imageArrayLayers;
    VkImageUsageFlags imageUsage;
    VkSharingMode imageSharingMode; P.12
    uint32_t queueFamilyIndexCount;
    const uint32_t* pQueueFamilyIndices;
    VkSurfaceTransformFlagBitsKHR preTransform;
    VkCompositeAlphaFlagBitsKHR compositeAlpha; P.11
    VkPresentModeKHR presentMode;
    VkBool32 clipped;
    VkSwapchainKHR oldSwapchain;
} VkSwapchainCreateInfoKHR;
```

colorSpace: VK_COLORSPACE_SRGB_NONLINEAR_KHR

presentMode: VK_PRESENT_MODE_*X*_KHR where *X* is IMMEDIATE, MAILBOX, FIFO, FIFO_RELAXED

```
void vkDestroySwapchainKHR(
    VkDevice device,
    VkSwapchainKHR swapchain,
    const VkAllocationCallbacks* pAllocator); P.10

VkResult vkCreateSharedSwapchainsKHR(
    VkDevice device,
    uint32_t swapchainCount,
    const VkSwapchainCreateInfoKHR* pCreateInfos,
    const VkAllocationCallbacks* pAllocator, P.10
    VkSwapchainKHR* pSwapchains);

VkResult vkGetSwapchainImagesKHR(
    VkDevice device,
    VkSwapchainKHR swapchain,
    uint32_t* pSwapchainImageCount,
    VkImage* pSwapchainImages);

VkResult vkAcquireNextImageKHR(
    VkDevice device,
    VkSwapchainKHR swapchain,
    uint64_t timeout,
    VkSemaphore semaphore,
    VkFence fence,
    uint32_t* pImageIndex);

VkResult vkQueuePresentKHR(
    VkQueue queue,
    const VkPresentInfoKHR* pPresentInfo);

typedef struct VkPresentInfoKHR {
    VkStructureType sType;
    const void* pNext;
    uint32_t waitSemaphoreCount;
    const VkSemaphore* pWaitSemaphores;
    uint32_t swapchainCount;
    const VkSwapchainKHR* pSwapchains;
    const uint32_t* pImageIndices;
    VkResult* pResults;
} VkPresentInfoKHR;

typedef struct VkDisplayPresentInfoKHR {
    VkStructureType sType;
    const void* pNext;
    VkRect2D srcRect; P.12
    VkRect2D dstRect; P.12
    VkBool32 persistent;
} VkDisplayPresentInfoKHR;
```

Extended Functionality [30]

Layers [30.1]

```
VkResult vkEnumerateInstanceLayerProperties(
    uint32_t* pPropertyCount,
    VkLayerProperties* pProperties);

VkResult vkEnumerateDeviceLayerProperties(
    VkPhysicalDevice physicalDevice,
    uint32_t* pPropertyCount,
    VkLayerProperties* pProperties);

typedef struct VkLayerProperties {
    char layerName [VK_MAX_EXTENSION_NAME_SIZE];
    uint32_t specVersion;
    uint32_t implementationVersion;
    char description [VK_MAX_DESCRIPTION_SIZE];
} VkLayerProperties;
```

Extensions [30.2]

```
VkResult vkEnumerateInstanceExtensionProperties(
    const char* pLayerName,
    uint32_t* pPropertyCount,
    VkExtensionProperties* pProperties);

VkResult vkEnumerateDeviceExtensionProperties(
    VkPhysicalDevice physicalDevice,
    const char* pLayerName,
    uint32_t* pPropertyCount,
    VkExtensionProperties* pProperties);

typedef struct VkExtensionProperties {
    char layerName [VK_MAX_EXTENSION_NAME_SIZE];
    uint32_t specVersion;
} VkExtensionProperties;
```

Features, Limits, and Formats [31]

Features [31.1]

```
void vkGetPhysicalDeviceFeatures(
    VkPhysicalDevice physicalDevice, P.11
    VkPhysicalDeviceFeatures* pFeatures);
```

Format Properties [31.3.2]

```
void vkGetPhysicalDeviceFormatProperties(
    VkPhysicalDevice physicalDevice,
    VkFormat format, P.11
    VkFormatProperties* pFormatProperties);

typedef struct VkFormatProperties {
    VkFormatFeatureFlags linearTilingFeatures;
    VkFormatFeatureFlags optimalTilingFeatures;
    VkFormatFeatureFlags bufferFeatures;
} VkFormatProperties;
```

enum VkFormatFeatureFlagBits:
VK_FORMAT_FEATURE_*X*_BIT where *X* is
SAMPLED_IMAGE,
STORAGE_IMAGE[_ATOMIC],
UNIFORM_TEXEL_BUFFER,
STORAGE_TEXEL_BUFFER[_ATOMIC],
VERTEX_BUFFER,
COLOR_ATTACHMENT[_BLEND],
DEPTH_STENCIL_ATTACHMENT,
BLIT_{SRC, DST},
SAMPLED_IMAGE_FILTER_LINEAR

Additional Image Capabilities [31.4]

```
VkResult vkGetPhysicalDeviceImageFormatProperties(
    VkPhysicalDevice physicalDevice,
    VkFormat format, P.11
    VkImageType type, P.11
    VkImageTiling tiling, P.11
    VkImageUsageFlags usage, P.11
    VkImageCreateFlags flags, P.11
    VkImageFormatProperties* pImageFormatProperties);

typedef struct VkImageFormatProperties {
    VkExtent3D maxExtent; P.10
    uint32_t maxMipLevels;
    uint32_t maxArrayLayers;
    VkSampleCountFlags sampleCounts; P.12
    VkDeviceSize maxResourceSize;
} VkImageFormatProperties;
```

Structures and Enumerations

This section contains types that are referenced in multiple places on preceding pages, in alphabetical order.

enum VkAccessFlagBits [6.5.4]

VK_ACCESS_*X*_BIT where *X* is
INDIRECT_COMMAND_READ,
INDEX_READ,
VERTEX_ATTRIBUTE_READ,
UNIFORM_READ,
INPUT_ATTACHMENT_READ,
SHADER_[READ, WRITE],
COLOR_ATTACHMENT_[READ, WRITE],
DEPTH_STENCIL_ATTACHMENT_[READ, WRITE],
TRANSFER_[READ, WRITE],
HOST_[READ, WRITE],
MEMORY_[READ, WRITE]

struct VkAllocationCallbacks [10.1]

```
typedef struct VkAllocationCallbacks {
    void* pUserData;
    PFN_vkAllocationFunction pfnAllocation;
    PFN_vkReallocationFunction pfnReallocation;
    PFN_vkFreeFunction pfnFree;
    PFN_vkInternalAllocationNotification
        pfnInternalAllocation;
    PFN_vkInternalFreeNotification pfnInternalFree;
} VkAllocationCallbacks;

typedef void* (VKAPI_PTR* PFN_vkAllocationFunction)(
    void* pUserData,
    size_t size,
    size_t alignment,
    VkSystemAllocationScope allocationScope);

typedef void* (
    VKAPI_PTR* PFN_vkReallocationFunction)(
    void* pUserData,
    void* pOriginal,
    size_t size,
    size_t alignment,
    VkSystemAllocationScope allocationScope);

typedef void (VKAPI_PTR* PFN_vkFreeFunction)(
    void* pUserData,
    void* pMemory);

typedef void (
    VKAPI_PTR* PFN_vkInternalAllocationNotification)(
    void* pUserData,
    size_t size,
    VkInternalAllocationType allocationType,
    VkSystemAllocationScope allocationScope);

typedef void (
    VKAPI_PTR* PFN_vkInternalFreeNotification)(
    void* pUserData,
    size_t size,
    VkInternalAllocationType allocationType,
    VkSystemAllocationScope allocationScope);
```

allocationType:
VK_INTERNAL_ALLOCATION_TYPE_EXECUTABLE

allocationScope: VK_SYSTEM_ALLOCATION_SCOPE_*X* where *X* is COMMAND, OBJECT, CACHE, DEVICE, INSTANCE

struct VkBufferMemoryBarrier [6.5.5]

```
typedef struct VkBufferMemoryBarrier {
    VkStructureType sType;
    const void* pNext;
    VkAccessFlags srcAccessMask; P.10
    VkAccessFlags dstAccessMask; P.10
    uint32_t srcQueueFamilyIndex;
    uint32_t dstQueueFamilyIndex;
    VkBuffer buffer;
    VkDeviceSize offset;
    VkDeviceSize size;
} VkBufferMemoryBarrier;
```

union VkClearColorValue [17.3]

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

struct VkClearDepthStencilValue [17.3]

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

union VkClearValue [17.3]

```
typedef union VkClearValue {
    VkClearColorValue color; P.10
    VkClearDepthStencilValue depthStencil; P.10
} VkClearValue;
```

Continued on next page >

Structures and Enumerations (continued)

enum VkCompareOp [25.8]

VK_COMPARE_OP_*X* where *X* is
NEVER,
LESS,
EQUAL,
LESS_OR_EQUAL,
GREATER,
NOT_EQUAL,
GREATER_OR_EQUAL,
ALWAYS

enum VkCompositeAlphaFlagBitsKHR

VK_COMPOSITE_ALPHA_*X*_BIT_KHR where *X* is
OPAQUE,
PRE_MULTIPLIED,
POST_MULTIPLIED,
INHERIT

enum VkDescriptorType [13.2.4]

VK_DESCRIPTOR_TYPE_*X* where *X* is
SAMPLER,
COMBINED_IMAGE_SAMPLER,
SAMPLED_IMAGE,
STORAGE_IMAGE,
UNIFORM_TEXEL_BUFFER,
STORAGE_TEXEL_BUFFER,
UNIFORM_BUFFER,
STORAGE_BUFFER,
UNIFORM_BUFFER_DYNAMIC,
STORAGE_BUFFER_DYNAMIC,
INPUT_ATTACHMENT

structs VkExtent2D, VkExtent3D [2.9.2]

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

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

enum VkFormat [31.3.1]

VK_FORMAT_*X* where *X* is
UNDEFINED,
R4G4_UNORM_PACK8,
R4G4B4A4_UNORM_PACK16,
B4G4R4A4_UNORM_PACK16,
R5G6B5_UNORM_PACK16,
B5G6R5_UNORM_PACK16,
R5G5B5A1_UNORM_PACK16,
B5G5R5A1_UNORM_PACK16,
A1R5G5B5_UNORM_PACK16,
R8_[UNORM, SNORM, USCALED],
R8_[SSCALED, UINT, SINT, SRGB],
R8G8_[UNORM, SNORM, USCALED],
R8G8_[SSCALED, UINT, SINT, SRGB],
R8G8B8_[UNORM, SNORM, USCALED],
R8G8B8_[SSCALED, UINT, SINT, SRGB],
B8G8R8_[UNORM, SNORM, USCALED],
B8G8R8_[SSCALED, UINT, SINT, SRGB],
R8G8B8A8_[UNORM, SNORM, USCALED],
R8G8B8A8_[SSCALED, UINT, SINT, SRGB],
B8G8R8A8_[UNORM, SNORM, USCALED],
B8G8R8A8_[SSCALED, UINT, SINT, SRGB],
A8B8G8R8_[UNORM, SNORM, USCALED]_PACK32,
A8B8G8R8_[SSCALED, UINT, SINT, SRGB]_PACK32,
A2R10G10B10_[UNORM, SNORM, USCALED]_PACK32,
A2R10G10B10_[SSCALED, UINT, SINT]_PACK32,
A2B10G10R10_[UNORM, SNORM, USCALED]_PACK32,
A2B10G10R10_[SSCALED, UINT, SINT]_PACK32,
R16_[UNORM, SNORM, USCALED],
R16_[SSCALED, UINT, SINT, SFLOAT],
R16G16_[UNORM, SNORM, USCALED],
R16G16_[SSCALED, UINT, SINT, SFLOAT],
R16G16B16_[UNORM, SNORM, USCALED],
R16G16B16_[SSCALED, UINT, SINT, SFLOAT],
R16G16B16A16_[UNORM, SNORM, USCALED],
R16G16B16A16_[SSCALED, UINT, SINT, SFLOAT],
R32_[UINT, SINT, SFLOAT],
R32G32_[UINT, SINT, SFLOAT],
R32G32B32_[UINT, SINT, SFLOAT],
R32G32B32A32_[UINT, SINT, SFLOAT],
R64_[UINT, SINT, SFLOAT],
R64G64_[UINT, SINT, SFLOAT],
R64G64B64_[UINT, SINT, SFLOAT],
R64G64B64A64_[UINT, SINT, SFLOAT],
B10G11R11_UFLOAT_PACK32,
E5B9G9R9_UFLOAT_PACK32,
D16_UNORM[_S8_UINT],
X8_D24_UNORM_PACK32,
D32_SFLOAT[_S8_UINT],
S8_UINT,
D24_UNORM_S8_UINT,
BC1_[RGB, RGBA]_UNORM_BLOCK,
BC1_[RGB, RGBA]_SRGB_BLOCK,
BC2_[UNORM, SRGB]_BLOCK,
BC3_[UNORM, SRGB]_BLOCK,
BC4_[UNORM, SRGB]_BLOCK,
BC5_[UNORM, SRGB]_BLOCK,
BC6H_[UFLOAT, SFLOAT]_BLOCK,
BC7_[UNORM, SRGB]_BLOCK,
ETC2_R8G8B8_[UNORM, SRGB]_BLOCK,
ETC2_R8G8B8A1_[UNORM, SRGB]_BLOCK,
ETC2_R8G8B8A8_[UNORM, SRGB]_BLOCK,
EAC_R11_[UNORM, SRGB]_BLOCK,
EAC_R11G11_[UNORM, SRGB]_BLOCK,
ASTC_4x4_[UNORM, SRGB]_BLOCK,
ASTC_5x4_[UNORM, SRGB]_BLOCK,
ASTC_5x5_[UNORM, SRGB]_BLOCK,
ASTC_6x5_[UNORM, SRGB]_BLOCK,
ASTC_6x6_[UNORM, SRGB]_BLOCK,
ASTC_8x5_[UNORM, SRGB]_BLOCK,
ASTC_8x6_[UNORM, SRGB]_BLOCK,
ASTC_8x8_[UNORM, SRGB]_BLOCK,
ASTC_10x5_[UNORM, SRGB]_BLOCK,
ASTC_10x6_[UNORM, SRGB]_BLOCK,
ASTC_10x8_[UNORM, SRGB]_BLOCK,
ASTC_10x10_[UNORM, SRGB]_BLOCK,
ASTC_12x10_[UNORM, SRGB]_BLOCK,
ASTC_12x12_[UNORM, SRGB]_BLOCK

enum VkImageAspectFlagBits [11.5]

VK_IMAGE_ASPECT_*X*_BIT where *X* is
COLOR,
DEPTH,
STENCIL,
METADATA

enum VkImageCreateFlagBits [11.3]

VK_IMAGE_CREATE_*X*_BIT where *X* is
SPARSE_{BINDING, RESIDENCY, ALIASED},
MUTABLE_FORMAT,
CUBE_COMPATIBLE

enum VkImageLayout [11.4]

VK_IMAGE_LAYOUT_*X* where *X* is
UNDEFINED,
GENERAL,
COLOR_ATTACHMENT_OPTIMAL,
DEPTH_STENCIL_ATTACHMENT_OPTIMAL,
DEPTH_STENCIL_READ_ONLY_OPTIMAL,
SHADER_READ_ONLY_OPTIMAL,
TRANSFER_SRC_OPTIMAL,
TRANSFER_DST_OPTIMAL,
PREINITIALIZED,
PRESENT_SRC_KHR

struct VkImageMemoryBarrier [6.5.6]

```
typedef struct VkImageMemoryBarrier {
    VkStructureType sType;
    const void* pNext;
    VkAccessFlags srcAccessMask; P.10
    VkAccessFlags dstAccessMask; P.10
    VkImageLayout oldLayout; P.11
    VkImageLayout newLayout; P.11
    uint32_t srcQueueFamilyIndex;
    uint32_t dstQueueFamilyIndex;
    VkImage image;
    VkImageSubresourceRange subresourceRange;
} VkImageMemoryBarrier;
```

struct VkImageSubresourceLayers [18.3]

```
typedef struct VkImageSubresourceLayers {
    VkImageAspectFlags aspectMask; P.11
    uint32_t mipLevel;
    uint32_t baseArrayLayer;
    uint32_t layerCount;
} VkImageSubresourceLayers;
```

struct VkImageSubresourceRange [11.5]

```
typedef struct VkImageSubresourceRange {
    VkImageAspectFlags aspectMask; P.11
    uint32_t baseMipLevel;
    uint32_t levelCount;
    uint32_t baseArrayLayer;
    uint32_t layerCount;
} VkImageSubresourceRange;
```

enum VkImageTiling [11.3]

VK_IMAGE_TILING_{OPTIMAL, LINEAR}

enum VkImageType [11.3]

VK_IMAGE_TYPE_{1D, 2D, 3D}

enum VkImageUsageFlagBits [11.3]

VK_IMAGE_USAGE_*X*_BIT where *X* is
TRANSFER_SRC,
TRANSFER_DST,
SAMPLED,
STORAGE,
COLOR_ATTACHMENT,
DEPTH_STENCIL_ATTACHMENT,
INPUT_ATTACHMENT,
TRANSIENT_ATTACHMENT

struct VkMemoryBarrier [6.5.4]

```
typedef struct VkMemoryBarrier {
    VkStructureType sType;
    const void* pNext;
    VkAccessFlags srcAccessMask; P.10
    VkAccessFlags dstAccessMask; P.10
} VkMemoryBarrier;
```

struct VkOffset2D, VkOffset3D [2.9.1]

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

typedef struct VkOffset3D {
    int32_t x;
    int32_t y;
    int32_t z;
} VkOffset3D;
```

struct VkPhysicalDeviceFeatures [31.1]

```
typedef struct VkPhysicalDeviceFeatures {
    VkBool32 robustBufferAccess;
    VkBool32 fullDrawIndexUint32;
    VkBool32 imageCubeArray;
    VkBool32 independentBlend;
    VkBool32 geometryShader;
    VkBool32 tessellationShader;
    VkBool32 sampleRateShading;
    VkBool32 dualSrcBlend;
    VkBool32 logicOp;
    VkBool32 multiDrawIndirect;
    VkBool32 drawIndirectFirstInstance;
    VkBool32 depthClamp;
    VkBool32 depthBiasClamp;
    VkBool32 fillModeNonSolid;
    VkBool32 depthBounds;
    VkBool32 wideLines;
    VkBool32 largePoints;
    VkBool32 alphaToOne;
    VkBool32 multiViewport;
    VkBool32 samplerAnisotropy;
    VkBool32 textureCompressionETC2;
    VkBool32 textureCompressionASTC_LDR;
    VkBool32 textureCompressionBC;
    VkBool32 occlusionQueryPrecise;
    VkBool32 pipelineStatisticsQuery;
    VkBool32 vertexPipelineStoresAndAtomics;
    VkBool32 fragmentStoresAndAtomics;
    VkBool32 shaderTessellationAndGeometryPointSize;
    VkBool32 shaderImageGatherExtended;
    VkBool32 shaderStorageImageExtendedFormats;
    VkBool32 shaderStorageImageMultisample;
    VkBool32 shaderStorageImageReadWithoutFormat;
    VkBool32 shaderStorageImageWriteWithoutFormat;
    VkBool32 shaderUniformBufferArrayDynamicIndexing;
    VkBool32 shaderSampledImageArrayDynamicIndexing;
    VkBool32 shaderStorageBufferArrayDynamicIndexing;
    VkBool32 shaderStorageImageArrayDynamicIndexing;
    VkBool32 shaderClipDistance;
    VkBool32 shaderCullDistance;
    VkBool32 shaderFloat64;
    VkBool32 shaderInt64;
    VkBool32 shaderInt16;
    VkBool32 shaderResourceResidency;
    VkBool32 shaderResourceMinLod;
    VkBool32 sparseBinding;
    VkBool32 sparseResidencyBuffer;
    VkBool32 sparseResidencyImage2D;
    VkBool32 sparseResidencyImage3D;
    VkBool32 sparseResidency2Samples;
    VkBool32 sparseResidency4Samples;
    VkBool32 sparseResidency8Samples;
    VkBool32 sparseResidency16Samples;
    VkBool32 sparseResidencyAliased;
    VkBool32 variableMultisampleRate;
    VkBool32 inheritedQueries;
} VkPhysicalDeviceFeatures;
```

Continued on next page >

Structures and Enumerations (continued)

struct VkPhysicalDeviceLimits [31.2]

```
typedef struct VkPhysicalDeviceLimits {
    uint32_t maxImageDimension1D;
    uint32_t maxImageDimension2D;
    uint32_t maxImageDimension3D;
    uint32_t maxImageDimensionCube;
    uint32_t maxImageArrayLayers;
    uint32_t maxTexelBufferElements;
    uint32_t maxUniformBufferRange;
    uint32_t maxStorageBufferRange;
    uint32_t maxPushConstantsSize;
    uint32_t maxMemoryAllocationCount;
    uint32_t maxSamplerAllocationCount;
    VkDeviceSize bufferImageGranularity;
    VkDeviceSize sparseAddressSpaceSize;
    uint32_t maxBoundDescriptorSets;
    uint32_t maxPerStageDescriptorSamplers;
    uint32_t maxPerStageDescriptorUniformBuffers;
    uint32_t maxPerStageDescriptorStorageBuffers;
    uint32_t maxPerStageDescriptorSampledImages;
    uint32_t maxPerStageDescriptorStorageImages;
    uint32_t maxPerStageDescriptorInputAttachments;
    uint32_t maxPerStageResources;
    uint32_t maxDescriptorSetSamplers;
    uint32_t maxDescriptorSetUniformBuffers;
    uint32_t maxDescriptorSetUniformBuffersDynamic;
    uint32_t maxDescriptorSetStorageBuffers;
    uint32_t maxDescriptorSetStorageBuffersDynamic;
    uint32_t maxDescriptorSetSampledImages;
    uint32_t maxDescriptorSetStorageImages;
    uint32_t maxDescriptorSetInputAttachments;
    uint32_t maxVertexInputAttributes;
    uint32_t maxVertexInputBindings;
    uint32_t maxVertexInputAttributeOffset;
    uint32_t maxVertexInputBindingStride;
    uint32_t maxVertexOutputComponents;
    uint32_t maxTessellationGenerationLevel;
    uint32_t maxTessellationPatchSize;
    uint32_t
        maxTessellationControlPerVertexInputComponents;
    uint32_t
        maxTessellationControlPerVertexOutputComponents;
    uint32_t
        maxTessellationControlPerPatchOutputComponents;
    uint32_t maxTessellationControlTotalOutputComponents;
    uint32_t maxTessellationEvaluationInputComponents;
    uint32_t maxTessellationEvaluationOutputComponents;
    uint32_t maxGeometryShaderInvocations;
    uint32_t maxGeometryInputComponents;
    uint32_t maxGeometryOutputComponents;
    uint32_t maxGeometryOutputVertices;
    uint32_t maxGeometryTotalOutputComponents;
    uint32_t maxFragmentInputComponents;
    uint32_t maxFragmentOutputAttachments;
    uint32_t maxFragmentDualSrcAttachments;
    uint32_t maxFragmentCombinedOutputResources;
    uint32_t maxComputeSharedMemorySize;
    uint32_t maxComputeWorkGroupCount[3];
    uint32_t maxComputeWorkGroupInvocations;
    uint32_t maxComputeWorkGroupSize[3];
    uint32_t subPixelPrecisionBits;
    uint32_t subTexelPrecisionBits;
    uint32_t mipmapPrecisionBits;
    uint32_t maxDrawIndexedIndexValue;
    uint32_t maxDrawIndirectCount;
    float maxSamplerLodBias;
    float maxSamplerAnisotropy;
    uint32_t maxViewports;
    uint32_t maxViewportDimensions[2];
    float viewportBoundsRange[2];
    uint32_t viewportSubPixelBits;
    size_t minMemoryMapAlignment;
    VkDeviceSize minTexelBufferOffsetAlignment;
    VkDeviceSize minUniformBufferOffsetAlignment;
    VkDeviceSize minStorageBufferOffsetAlignment;
    int32_t minTexelOffset;
    uint32_t maxTexelOffset;
    int32_t minTexelGatherOffset;
    uint32_t maxTexelGatherOffset;
    float minInterpolationOffset;
    float maxInterpolationOffset;
    uint32_t subPixelInterpolationOffsetBits;
    uint32_t maxFramebufferWidth;
    uint32_t maxFramebufferHeight;
    uint32_t maxFramebufferLayers;
    VkSampleCountFlags framebufferColorSampleCounts; P.12
    VkSampleCountFlags framebufferDepthSampleCounts; P.12
    VkSampleCountFlags framebufferStencilSampleCounts; P.12
    VkSampleCountFlags
        framebufferNoAttachmentsSampleCounts; P.12
    uint32_t maxColorAttachments;
    VkSampleCountFlags
        sampledImageColorSampleCounts; P.12
    VkSampleCountFlags
        sampledImageIntegerSampleCounts; P.12
    VkSampleCountFlags
        sampledImageDepthSampleCounts; P.12
    VkSampleCountFlags
        sampledImageStencilSampleCounts; P.12
    VkSampleCountFlags storageImageSampleCounts;
    uint32_t maxSampleMaskWords;
    VkBool32 timestampComputeAndGraphics;
    float timestampPeriod;
    uint32_t maxClipDistances;
    uint32_t maxCullDistances;
    uint32_t maxCombinedClipAndCullDistances;
    uint32_t discreteQueuePriorities;
    float pointSizeRange[2];
    float lineWidthRange[2];
    float pointSizeGranularity;
    float lineWidthGranularity;
    VkBool32 strictLines;
    VkBool32 standardSampleLocations;
    VkDeviceSize optimalBufferCopyOffsetAlignment;
    VkDeviceSize optimalBufferCopyRowPitchAlignment;
    VkDeviceSize nonCoherentAtomSize;
} VkPhysicalDeviceLimits;
```

struct VkPipelineShaderStageCreateInfo [9.1]

```
typedef struct VkPipelineShaderStageCreateInfo {
    VkStructureType sType;
    const void* pNext;
    VkPipelineShaderStageCreateFlags flags; = 0
    VkShaderStageFlagBits stage; P.12
    VkShaderModule module;
    const char* pName;
    const VkSpecializationInfo* pSpecializationInfo;
} VkPipelineShaderStageCreateInfo;

typedef struct VkSpecializationInfo {
    uint32_t mapEntryCount;
    const VkSpecializationMapEntry* pMapEntries;
    size_t dataSize;
    const void* pData;
} VkSpecializationInfo;

typedef struct VkSpecializationMapEntry {
    uint32_t constantID;
    uint32_t offset;
    size_t size;
} VkSpecializationMapEntry;
```

enum VkPipelineStageFlagBits [6.5.2]

VK_PIPELINE_STAGE_*X*_BIT where *X* is
TOP_OF_PIPE,
DRAW_INDIRECT,
VERTEX_[INPUT, SHADER],
TESSELLATION_[CONTROL,
EVALUATION]_SHADER,
[COMPUTE, GEOMETRY, FRAGMENT]_SHADER,
[EARLY, LATE]_FRAGMENT_TESTS,
COLOR_ATTACHMENT_OUTPUT,
TRANSFER, BOTTOM_OF_PIPE, HOST,
ALL_{GRAPHICS, COMMANDS}

enum VkQueryPipelineStatisticFlagBits [16.4]

VK_QUERY_PIPELINE_STATISTIC_*X*_BIT where *X* is
INPUT_ASSEMBLY_{VERTICES, PRIMITIVES},
VERTEX_SHADER_INVOCATIONS,
GEOMETRY_SHADER_{INVOCATIONS, PRIMITIVES},
CLIPPING_{INVOCATIONS, PRIMITIVES},
FRAGMENT_SHADER_INVOCATIONS,
TESSELLATION_CONTROL_SHADER_PATCHES,
TESSELLATION_EVALUATION_SHADER_INVOCATIONS,
COMPUTE_SHADER_INVOCATIONS

struct VkRect2D [2.9.3]

```
typedef struct VkRect2D {
    VkOffset2D offset; P.10
    VkExtent2D extent; P.10
} VkRect2D;
```

enum VkSampleCountFlagBits [31.2]

VK_SAMPLE_COUNT_*X*_BIT where *X* is
1, 2, 4, 8, 16, 32, 64

enum VkShaderStageFlagBits [9.1]

VK_SHADER_STAGE_*X* where *X* is
{VERTEX, GEOMETRY, FRAGMENT, COMPUTE}_BIT,
TESSELLATION_CONTROL_BIT,
TESSELLATION_EVALUATION_BIT,
ALL_GRAPHICS, ALL

enum VkSharingMode [11.7]

VK_SHARING_MODE_EXCLUSIVE,
VK_SHARING_MODE_CONCURRENT

struct VkSparseMemoryBind [28.7.6]

```
typedef struct VkSparseMemoryBind {
    VkDeviceSize resourceOffset;
    VkDeviceSize size;
    VkDeviceMemory memory;
    VkDeviceSize memoryOffset;
    VkSparseMemoryBindFlags flags;
} VkSparseMemoryBind;
```

flags: VK_SPARSE_MEMORY_BIND_METADATA_BIT

enum VkSurfaceTransformFlagBitsKHR

VK_SURFACE_TRANSFORM_*X*_BIT_KHR where *X* is
IDENTITY,
ROTATE_{90, 180, 270},
HORIZONTAL_MIRROR,
HORIZONTAL_MIRROR_ROTATE_{90, 180, 270},
INHERIT

struct VkViewport [23.5]

```
typedef struct VkViewport {
    float x;
    float y;
    float width;
    float height;
    float minDepth;
    float maxDepth;
} VkViewport;
```

Notes

Vulkan Reference Guide Index

The following index shows each item included on this card along with the page on which it is described. The color of the row in the table below is the color of the pane to which you should refer.

Vulkan™ KHRONOS™ GROUP