Introducción a Vulkan

Samuel Iglesias Gonsálvez
siglesias@igalia.com

Sobre mí

Ingeniero de Telecomunicación por la Universidad de Oviedo

Experiencia laboral: INDRA, CERN, Igalia (desde 2012)

Socio de Igalia

Miembro de asociaciones que fomentan el software libre: AsturLiNUX, GPUL, FSF, FSFE

Sobre Igalia

Consultora centrada en el software libre

Áreas actuales de negocio: navegadores web, compiladores, gráficos, multimedia, networking, virtualización...

Fundada en 2001

Basada en A Coruña, España

~60 hackers distribuidos por todo el mundo: más de 12 países

3 principios fundamentales

Responsabilidad compartida

Software libre

Promover la dignidad profesional y la responsabilidad social.

¿Qué es Vulkan?

Vulkan es una API multiplataforma para el desarrollo de aplicaciones con gráficos 3D.

Microsoft Windows, Android, GNU/Linux, Tizen...

Está diseñada para ofrecer un mayor rendimiento y un mejor balanceo de carga entre CPU/GPU.

Pero... ¿por qué necesitamos Vulkan?

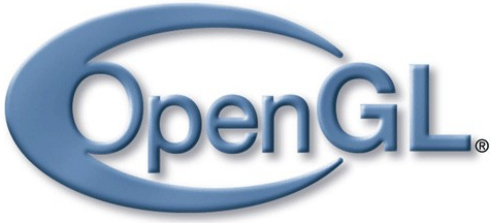
Un poco de historia

OpenGL 1.0 fue publicado en Enero de 1992

Basado en la API llamada Iris GL de Silicon Graphics (SGI)

SGI y otras empresas mantenían la especificación OpenGL bajo el grupo OpenGL Architecture Review Board (OpenGL ARB). En 2006 se transfiere el control a Khronos Group.

¡Ha pasado más de 25 años!



Un poco de historia

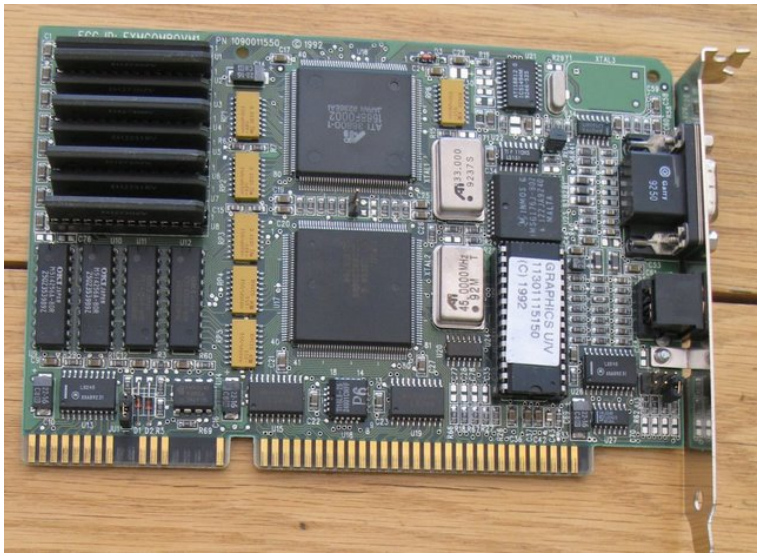
OpenGL fue evolucionado a la vez que las tarjetas gráficas y las necesidades de los usuarios

- OpenGL 1.X (1992-2003): fixed functions, soporte inicial de varios tipos de texturas, multisampling, mipmapping, VBO, etc.
- OpenGL 2.X (2004-2006): soporte de shaders (GLSL), etc.
- OpenGL 3.X (2008-2010): vertex array objects, framebuffer objects, Uniform buffer objects, etc.
- OpenGL 4.X (2010-actualidad): operaciones en coma flotante de 64 bits, subrutinas, instancias, atomic counters, compute shaders, etc.

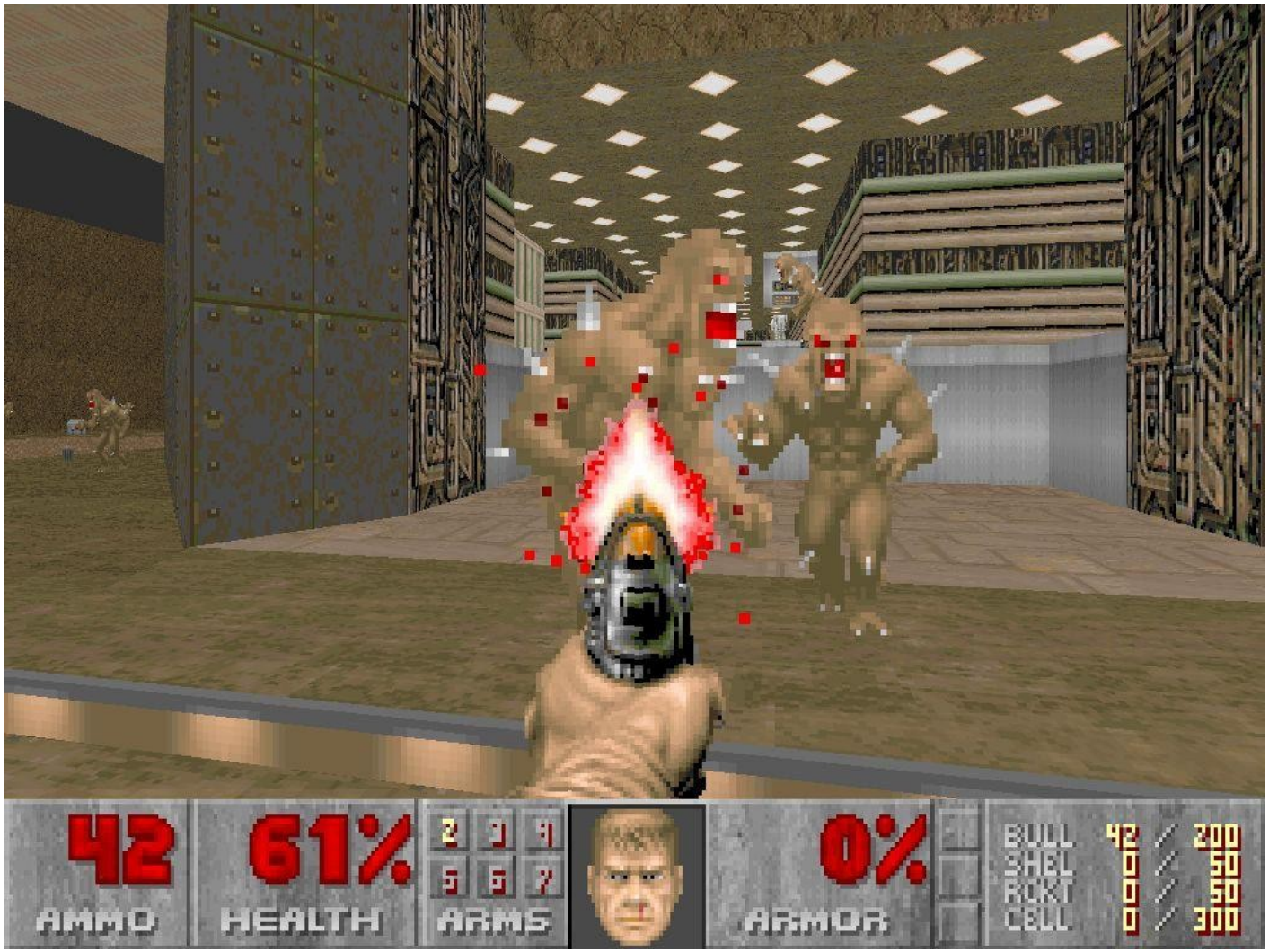
La última versión es OpenGL 4.6 (Julio 2017)

Un poco de historia

Hemos avanzado mucho en tecnología



Un poco de historia



Un poco de historia

Copyright id Software. Hobby Consolas

Un poco de historia

OpenGL ya sufre problemas debidos a su diseño original de 1992

- OpenGL API es una máquina de estados.
- El estado de OpenGL está asociado a un único on-screen context.
- OpenGL esconde lo que la hace realmente la GPU.
- Alto consumo de CPU: comprobación de errores en cada llamada, compilación de shaders se retrasa hasta cuando se ejecuta glDraw*(), etc.
- Cada implementación de OpenGL incluye un compilador propio de GLSL: posibles problemas debidos a bugs específicos en cada driver, etc.

OpenGL ha sobrevivido muy bien todos estos años... ¡y los que le quedan!

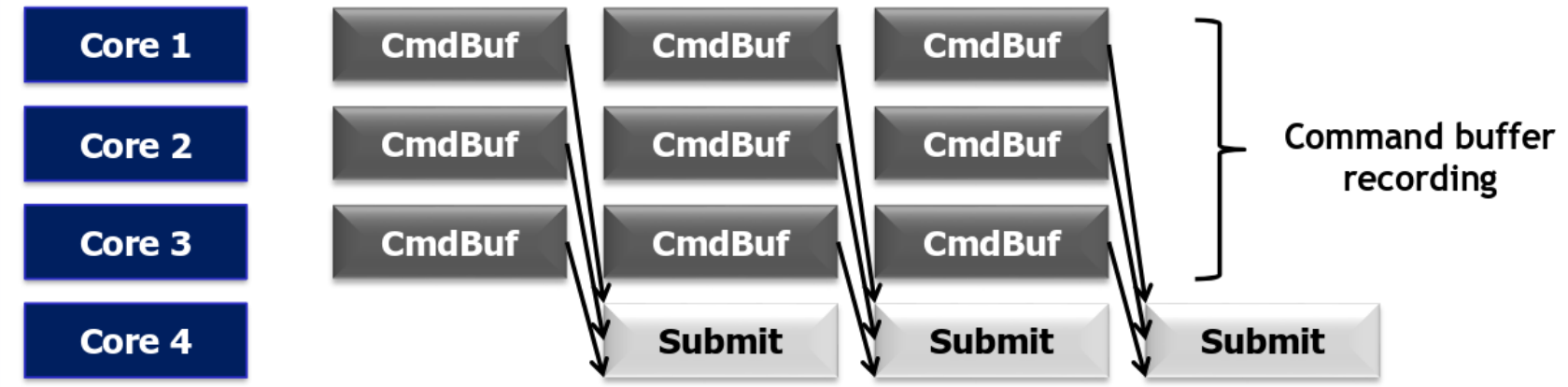
Características de Vulkan

Vulkan 1.0 fue lanzado en Febrero 2016.

API basada en objetos sin un estado global

Command buffers y dispatch queues.

Multithreading/Multicore



Características de Vulkan

WSI (Window System Integration) es una extensión de Vulkan.

Vulkan es mucho más explícito sobre qué hace la GPU

Texture formats, gestión de memoria, sincronización... son controlados por la aplicación, no por el driver.

La compilación de shaders y la generación de comandos se hace en tiempos predecibles.

¡Los drivers de Vulkan no hacen comprobación de errores!

Se hace de manera opcional con una capa de validación.

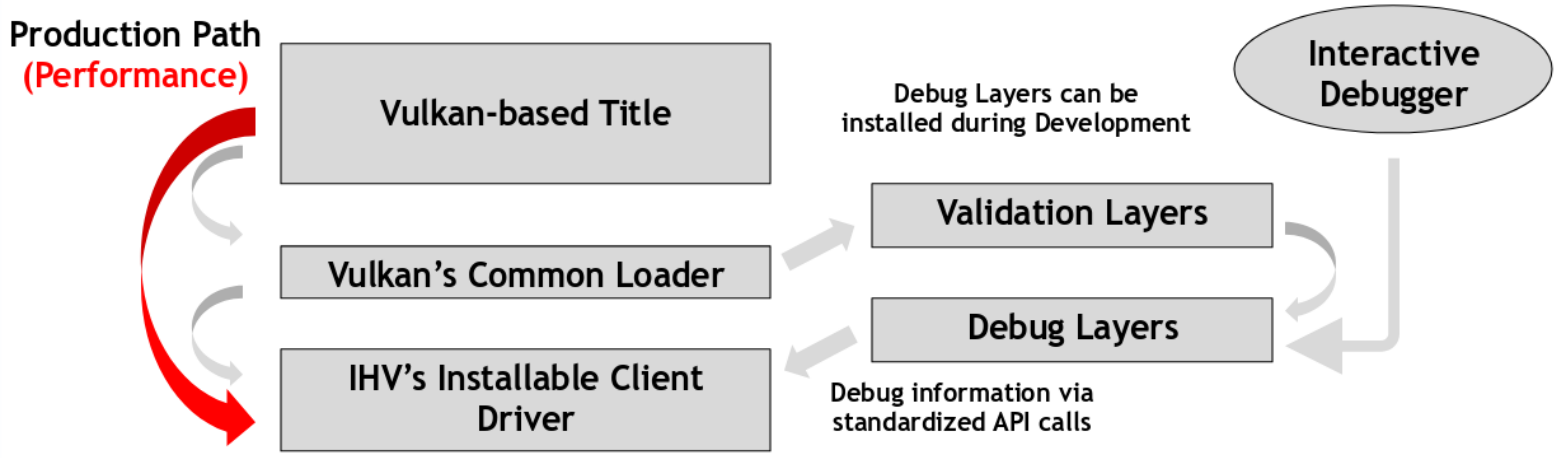
Conceptos en Vulkan

Loader

Es común en cada plataforma.

Detecta las layers disponibles, los drivers instalados en el sistema, etc.

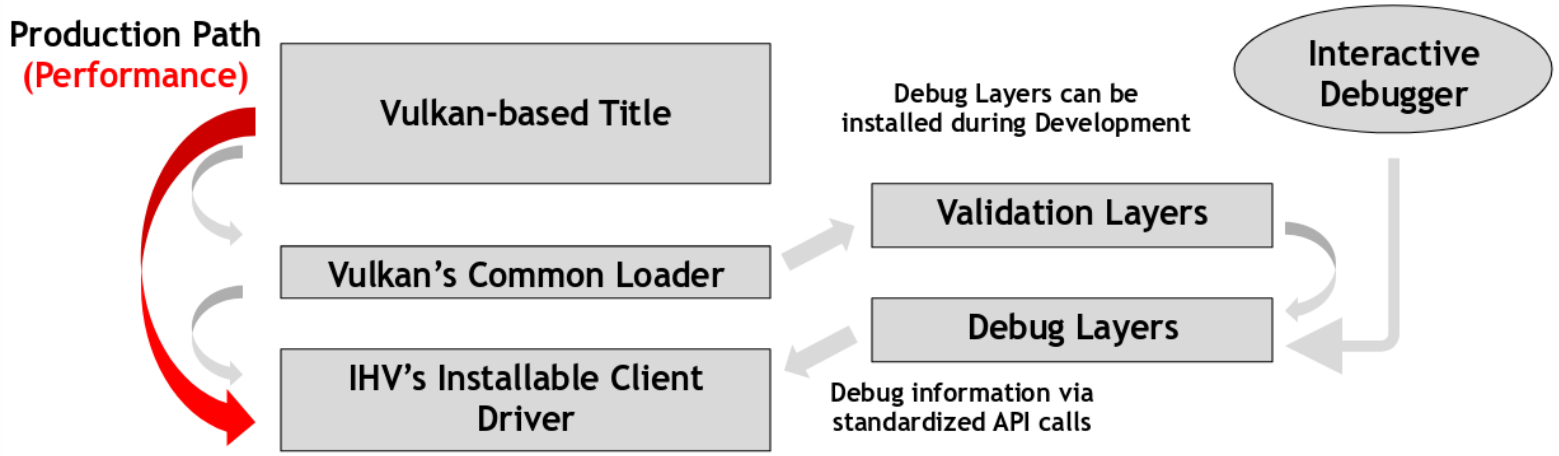
https://github.com/KhronosGroup/Vulkan-LoaderAndValidationLayers



Conceptos en Vulkan

Layer

Permiten cargar en tiempo de ejecución y/o desde el código de la aplicación distintas funcionalidades: validación, depuración, hacer un log de llamadas a la API, etc.



Conceptos en Vulkan

Instancia

Es un objecto que inicializa la librería de Vulkan y permite pasar información sobre la aplicación a la propia implementación.

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

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

Conceptos en Vulkan

Physical device

Representa un dispositivo único en el sistema (aunque puede un conjunto de dispositivos HW trabajando juntos), de los cuales hay un número finito.

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

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;
    VkPhysicalDeviceSparseProperties    sparseProperties;
} VkPhysicalDeviceProperties;
```

Conceptos en Vulkan

Logical device

Representa la vista de la aplicación de uno o varios dispositivos físicos. Se representa como un objecto VkDevice.

Cada VkDevice expone una serie de familias de colas (queue). Todas las colas de una misma familia soportan las mismas operaciones.

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

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

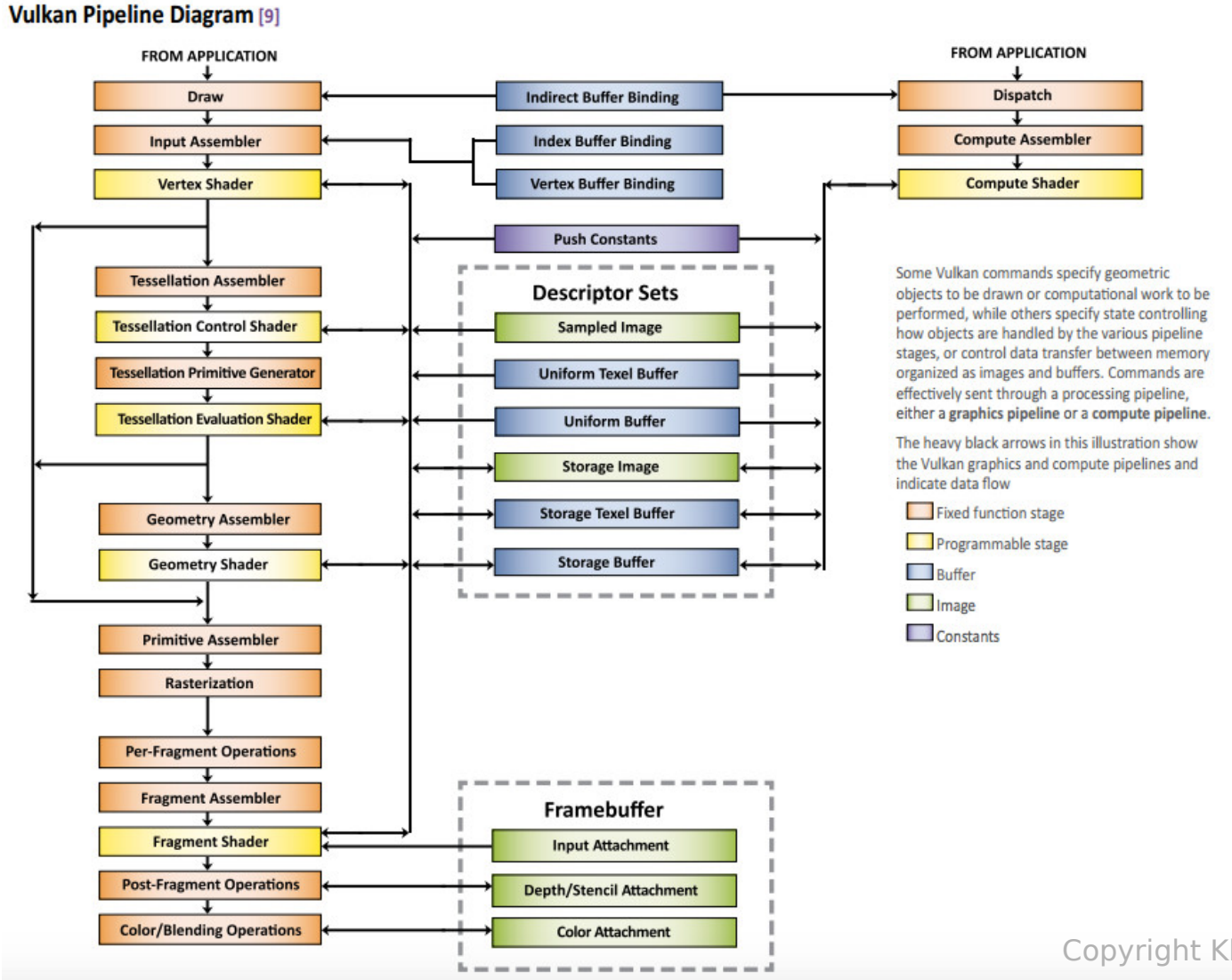
Conceptos en Vulkan

VkPipeline

Es un objeto que describe el estado de un pipeline: los shaders que se van a ejecutar, el pipeline layout, el viewport, specialization constants, el estado para los vertex inputs, la rasterización, color blending, etc.

Es decir, es un objecto que describe el pipeline que se quiere emplear y qué tipo de recursos se van a usar en el mismo.

Conceptos en Vulkan



Conceptos de Vulkan

Shaders

Se cargan como objetos binarios en lenguaje SPIR-V.

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

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

Conceptos de Vulkan

SPIR-V

Es un lenguaje intermedio binario definido por Khronos que sirve tanto para shaders gráficos como de computación.

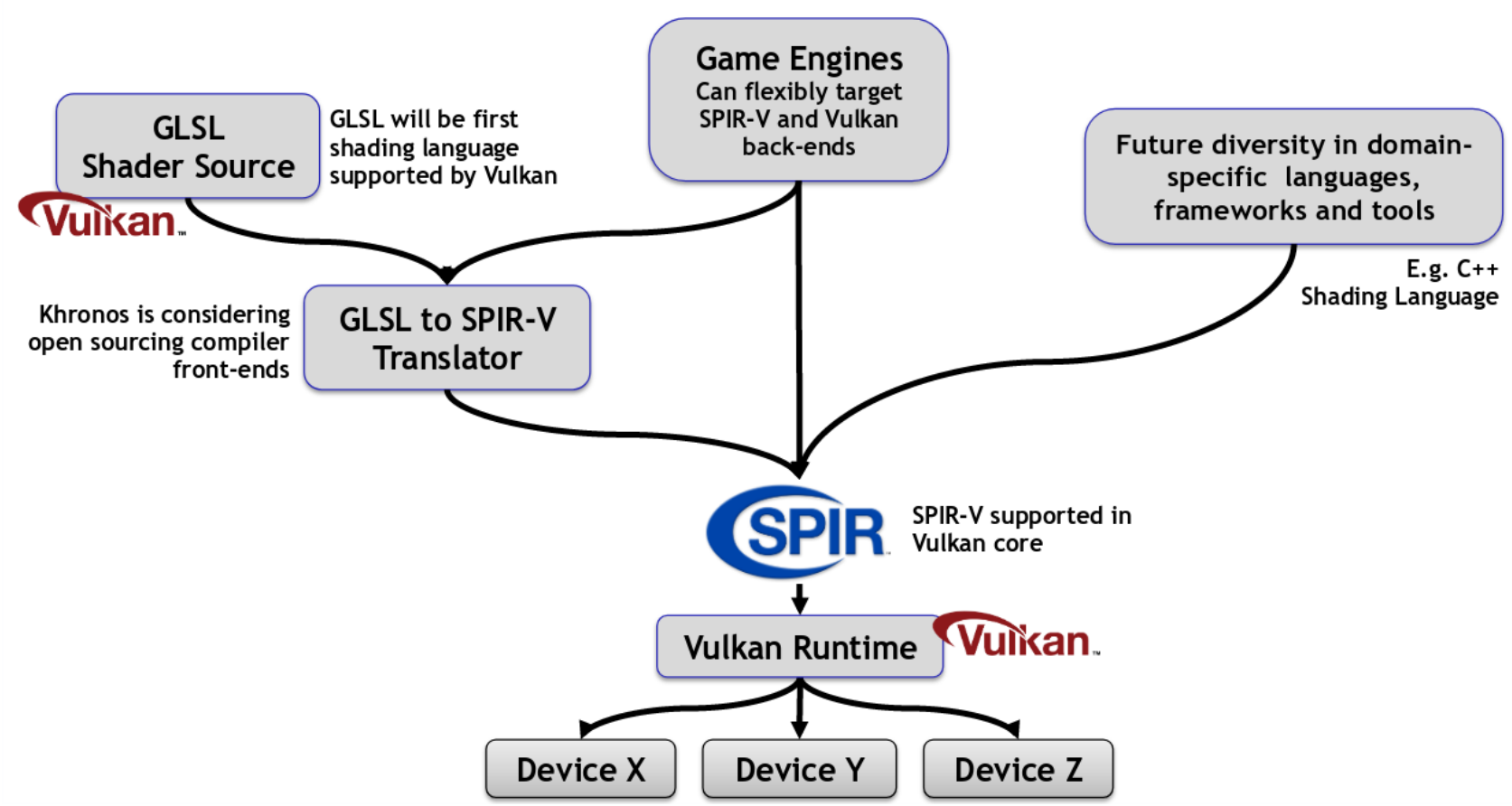
La compilación se hace offline mediante un front-end compiler desde otro lenguaje: GLSL -> SPIR-V, HLSL -> SPIR-V, etc.

https://github.com/KhronosGroup/glslang

Hay herramientas para realizar validación, optimizador, ensamblador (texto a binario), desemsamblador (binario a texto), etc

https://github.com/KhronosGroup/SPIRV-Tools

Conceptos de Vulkan



Conceptos de Vulkan

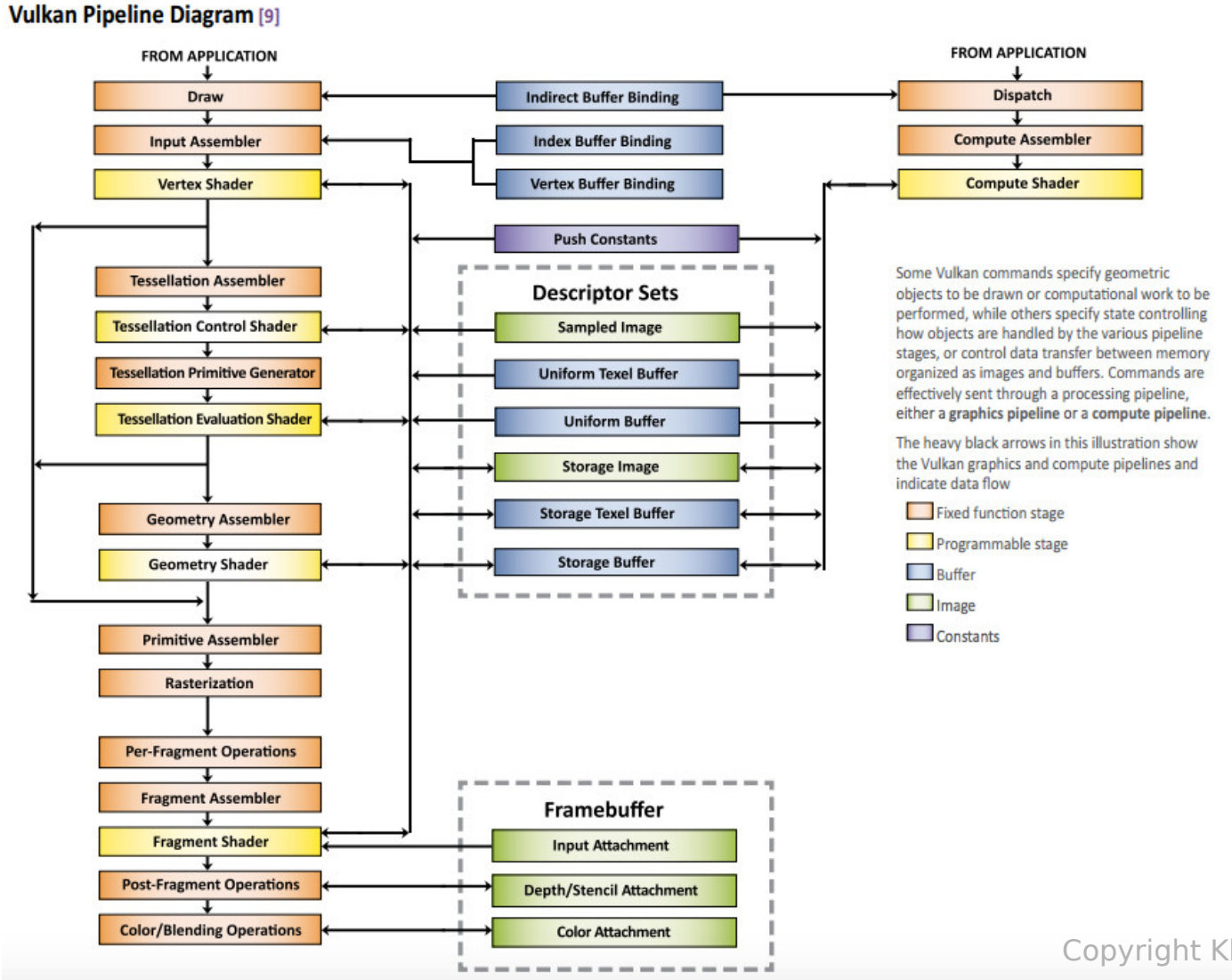
```
#version 310 es
precision mediump float;
uniform sampler2D s;
in vec2 texcoord;
out vec4 color;

void main()
{
        color = texture(s, texcoord);
}
```

```
; SPIR-V
; Version: 1.0
; Generator: Khronos Glslang Reference Front End; 1
; Bound: 20
; Schema: 0
                OpCapability Shader
         %1 = OpExtInstImport "GLSL.std.450"
                OpMemoryModel Logical GLSL450
                OpEntryPoint Fragment %4 "main" %9 %17
                OpExecutionMode %4 OriginUpperLeft
                OpSource ESSL 310
                OpName %4 "main"
                OpName %9 "color"
                OpName %13 "s"
                OpName %17 "texcoord"
                OpDecorate %9 RelaxedPrecision
                OpDecorate %13 RelaxedPrecision
                OpDecorate %13 DescriptorSet 0
                OpDecorate %14 RelaxedPrecision
                OpDecorate %17 RelaxedPrecision
                OpDecorate %18 RelaxedPrecision
                OpDecorate %19 RelaxedPrecision
         %2 = OpTypeVoid
         %3 = OpTypeFunction %2
```

```
%6 = OpTypeFloat 32
         %7 = OpTypeVector %6 4
         %8 = OpTypePointer Output %7
         %9 = OpVariable %8 Output
        %10 = OpTypeImage %6 2D 0 0 0 1 Unknown
        %11 = OpTypeSampledImage %10
        %12 = OpTypePointer UniformConstant %11
        %13 = OpVariable %12 UniformConstant
        %15 = OpTypeVector %6 2
        %16 = OpTypePointer Input %15
        %17 = OpVariable %16 Input
         %4 = OpFunction %2 None %3
         %5 = OpLabel
        %14 = OpLoad %11 %13
        %18 = OpLoad %15 %17
        %19 = OpImageSampleImplicitLod %7 %14 %18
                OpStore %9 %19
                OpReturn
                OpFunctionEnd
```

Conceptos en Vulkan



Conceptos en Vulkan

Descriptor sets

Un **descriptor** es una estructura de datos que representa un recurso que usa el shader como un buffer view, image view, sampler.

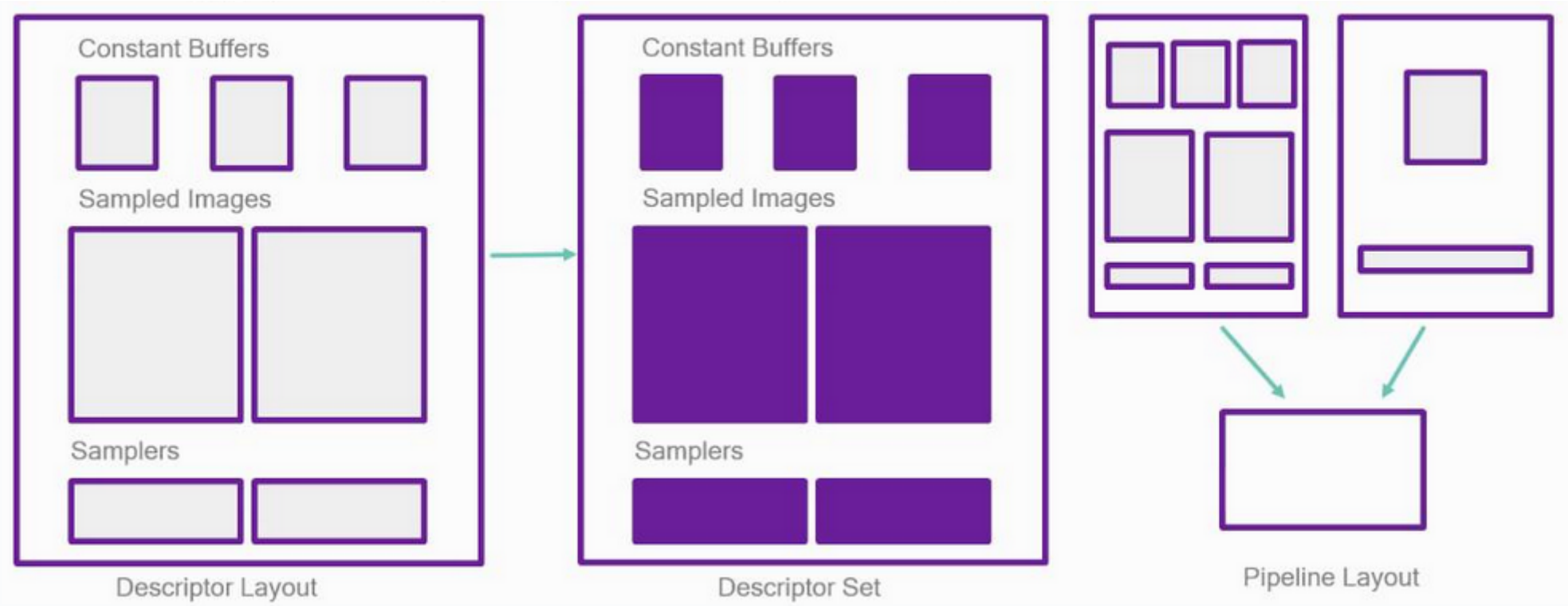
El **descriptor set layout** define cuántos, en qué orden y de qué tipo son estos recursos.

Un **descriptor set** es un conjunto que descriptors que se asocian en un pipeline.

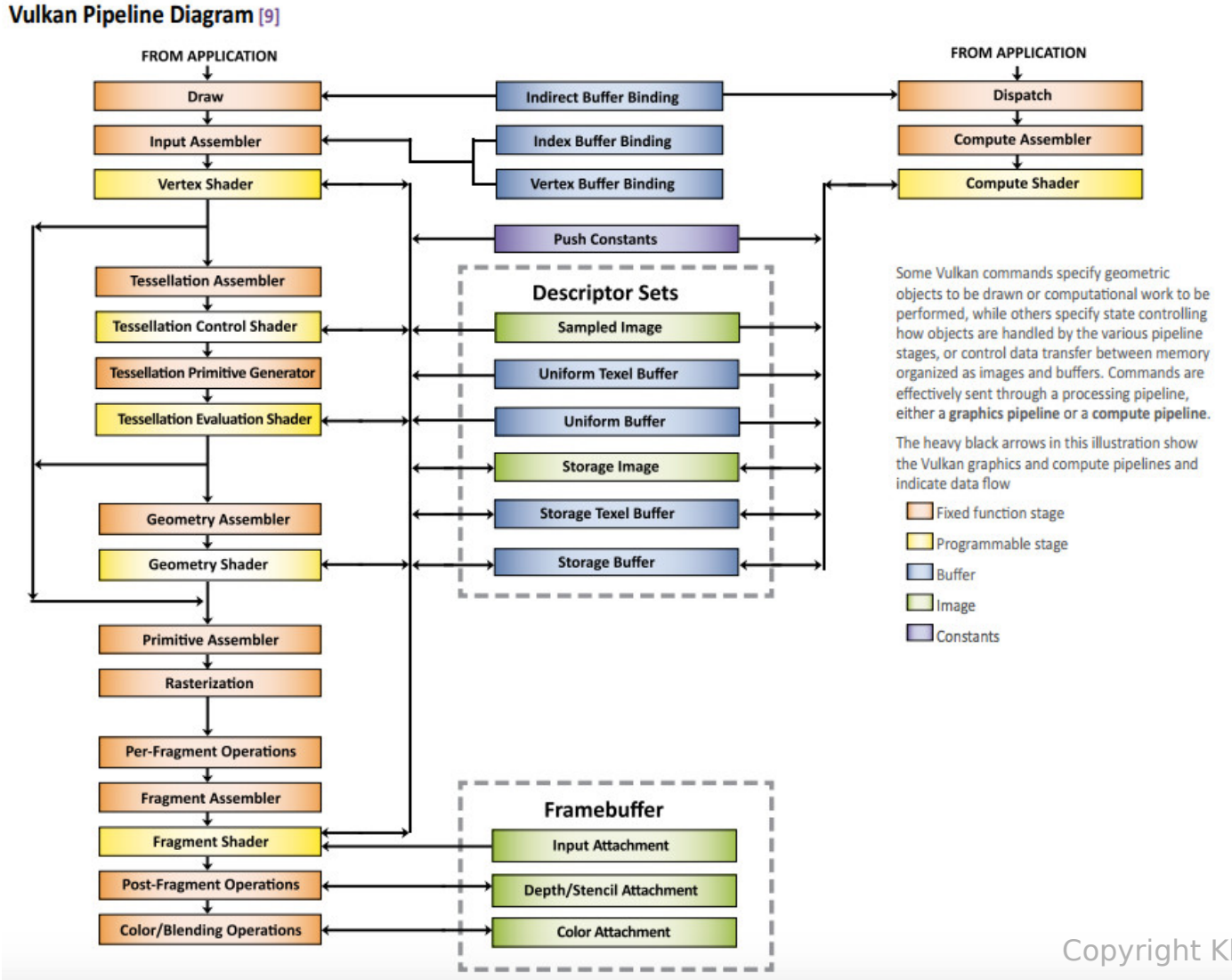
Se pueden utilizar uno o varios descriptor sets en un mismo pipeline, los cuales forman el pipeline layout.

Conceptos en Vulkan

Descriptor sets



Conceptos en Vulkan



Conceptos en Vulkan

Framebuffer

Representan una colección de attachments de memoria que un render pass utiliza. Básicamente es un conjunto de VkImageViews, que representan color attachments, depth-stencil attachment.

Conceptos en Vulkan

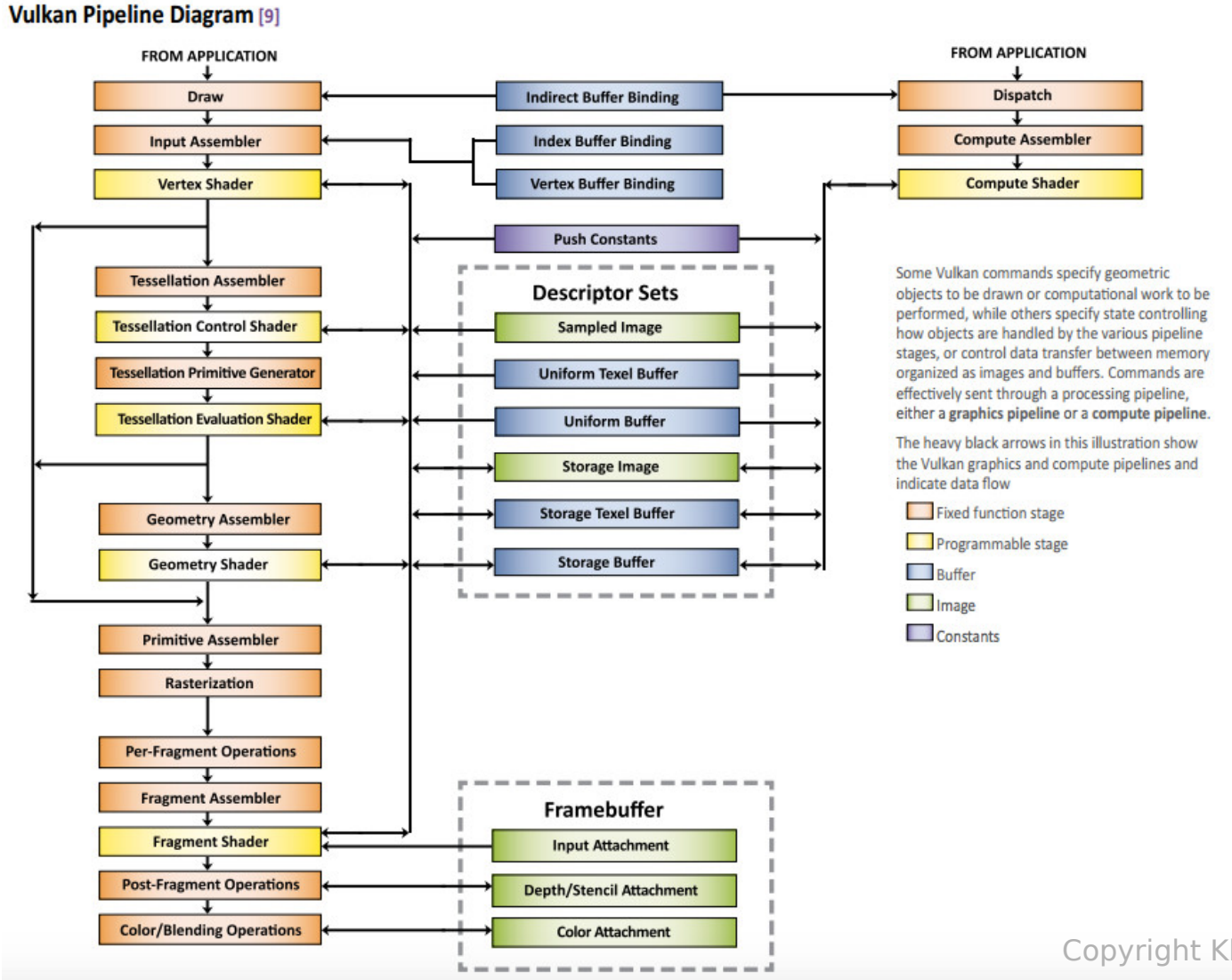
Swapchain

Es la infraestructura que proporciona imágenes para ser pintadas que luego se mostrarán en pantalla. Básicamente es una cola de imágenes que se usarán para pintar: double/triple buffering, etc.

Forma parte de la extensión **VK_KHR_swapchain**.

La aplicación va a adquirir una imagen, pintar sobre ella y devolverla a la cola para su presentación, siguiendo el modo configurado: immediate, fifo y otros.

Conceptos en Vulkan



Conceptos en Vulkan

Queues

Cualquier trabajo que se quiera mandar a la GPU se hace a través de colas (queues), incluyendo sincronización con semáforos y fences.

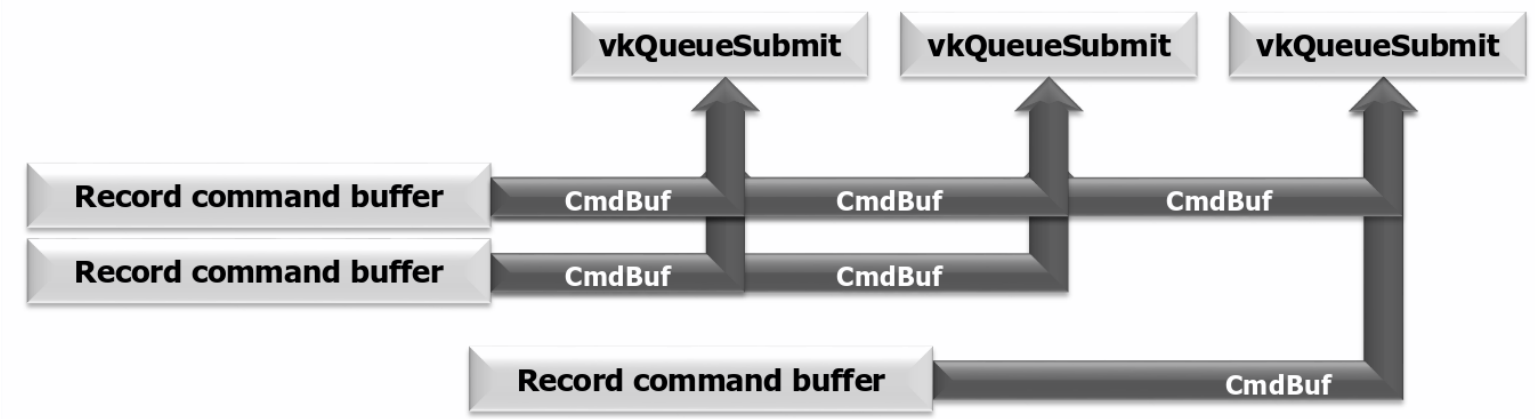
Un dispositivo podría tener distintas familias de colas, cada una con distintas propiedades: una cola para operaciones gráficas, otra para operaciones de computación, otra para realizar operaciones de transferencia.

Conceptos en Vulkan

Command Buffers

Son objetos donde se guardan los comandos que queremos que se ejecuten por el dispositivo y que se mandaran a través de la correspondiente cola.

Básicamente se guarda en ellos cualquier comando de la API que empiece por vkCmd*()



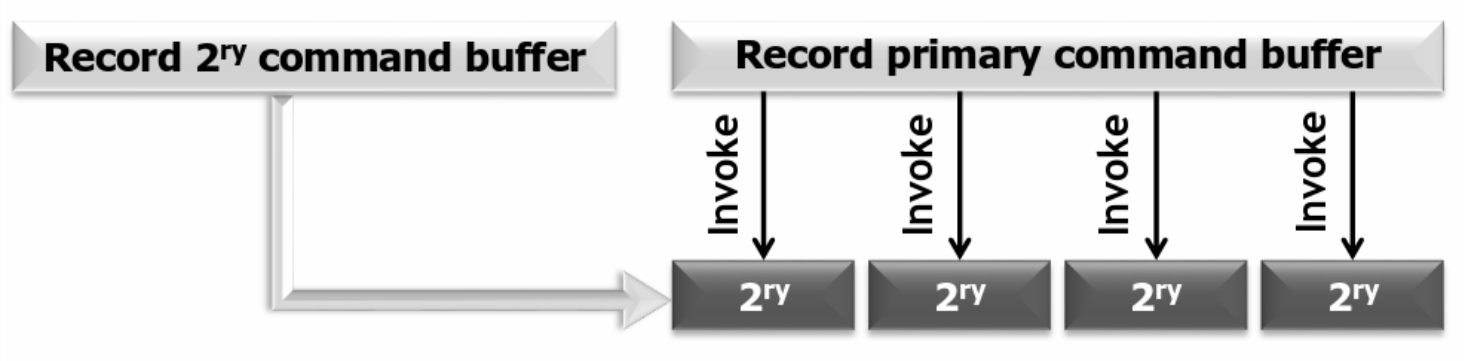
Conceptos en Vulkan

Command Buffers

Hay dos tipos de command buffers:

Primarios: pueden ser enviados a una cola y pueden llamar a command buffer secundarios, aunque no pueden ser llamados por otros primarios.

Secundarios: no se pueden enviar directamente a una cola. Son ejecutados por otros command buffers primarios.

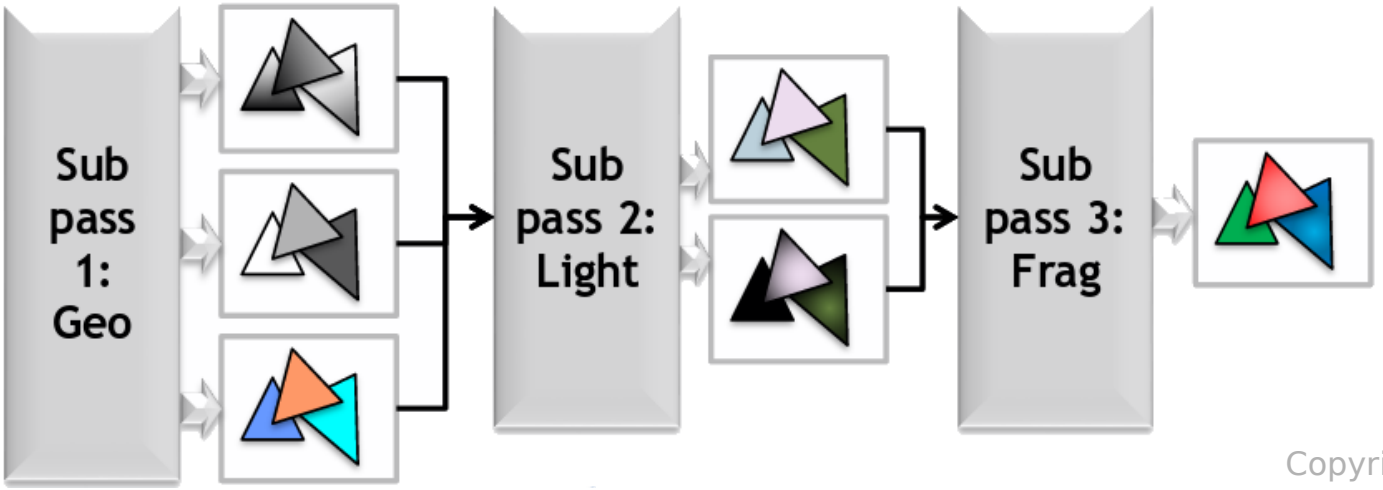


Conceptos en Vulkan

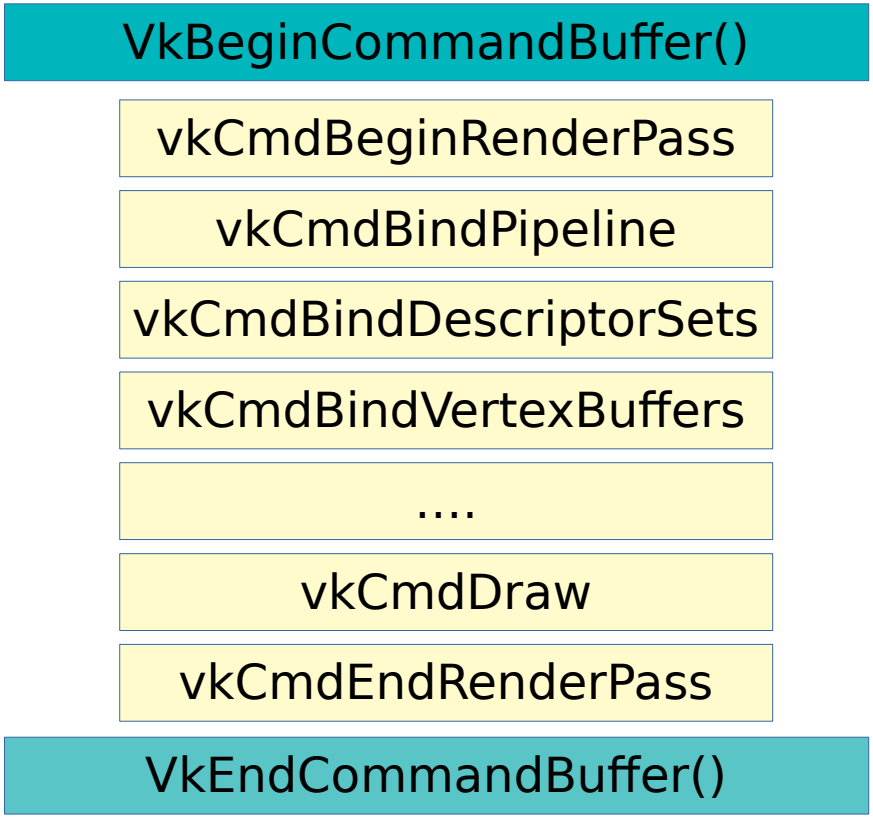
Render pass

Representa una colección de attachments (color, depth, stencil), subpases y las dependencias entre subpasses. Es decir, describe cómo los attachments son utilizados en los subpases.

Es decir, es una representación de las fases lógicas de un frame.



Conceptos en Vulkan



Conceptos en Vulkan

Sincronización

Hay varias maneras de realizar sincronización:

Fences: comunicar al host que una cierta tarea ha terminado de ejecutarse en el dispositivo.

Semaphore: controla el acceso a recursos entre distintas colas.

Events: se puede utilizar tanto por el host como dentro de un command buffer para hacer que éste último espere por el evento o el host consulte el estado de un evento.

Pipeline barriers: sincronización dentro de un command buffer.

Conclusiones

Vulkan es un API enfocada al rendimiento

- Se configura prácticamente todo antes de mandar trabajo a la GPU.
- Soporte de multithreading.
- La comprobación de errores no se hace en el driver.

Vulkan es un API explícita y configurable

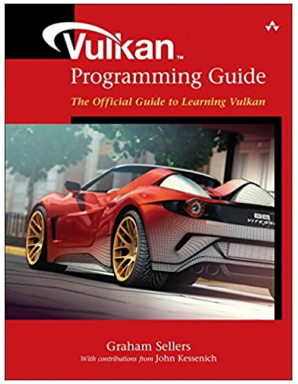
- Alta curva de aprendizaje.
- Habilita casos de uso que no se podrían cubrir en OpenGL.

Vulkan está evolucionando

- Nuevas extensiones se añaden cada poco para cubrir huecos en la spec.

Vulkan es presente y futuro.

Más información



Specs

https://www.khronos.org/vulkan/

https://www.khronos.org/registry/vulkan/#apispecs

Book

"Vulkan Programming Guide: The Official Guide to Learning Vulkan", Graham Sellers. ISBN-13: 978-0134464541

Tutoriales & Ejemplos

https://vulkan-tutorial.com/

https://software.intel.com/en-us/articles/api-without-secrets-introduction-to-vulkan-part-1

https://github.com/SaschaWillems/Vulkan

https://renderdoc.org/vulkan-in-30-minutes.html

Más información

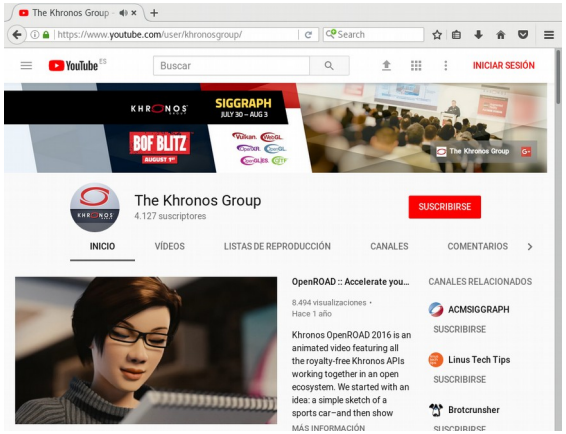
Videos

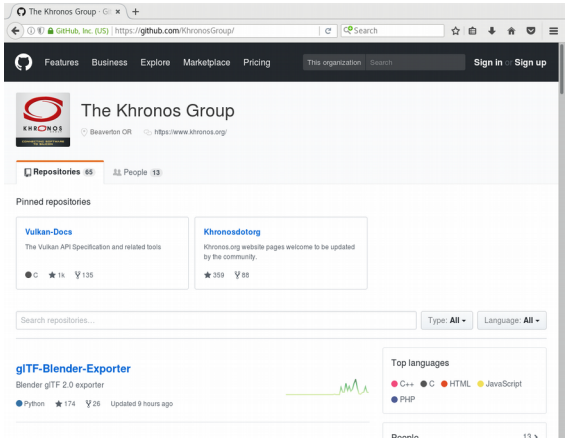
https://www.youtube.com/user/khronosgroup/

https://www.youtube.com/playlist?list=PLYO7XTAX41FPp1AQbwr6wA-IUy4cAHLNN

Github

https://github.com/KhronosGroup/

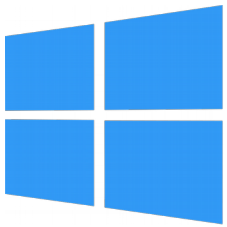


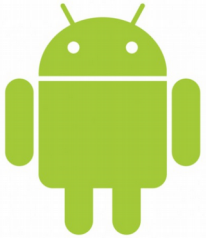


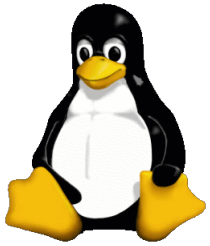
Más información

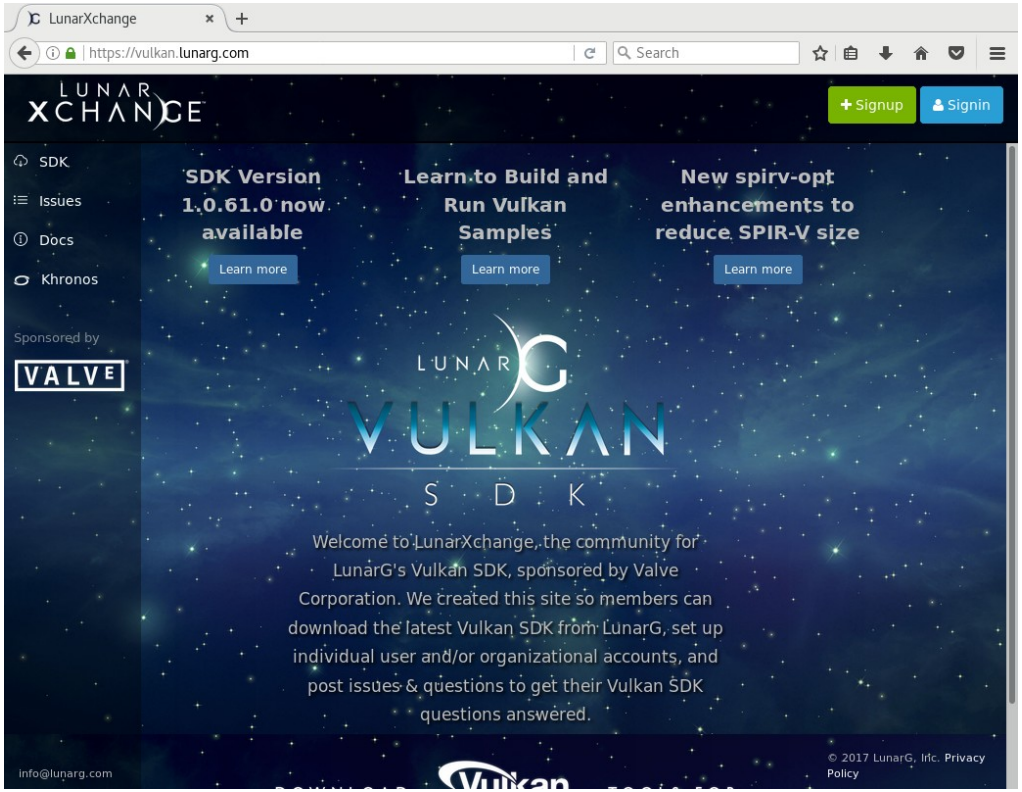
SDK

https://vulkan.lunarg.com/









Vídeo

Dudas/Preguntas

¡Muchas gracias!

Samuel Iglesias Gonsálvez
siglesias@igalia.com