**How It Work (Or should)**

-A draw command buffer

En premier on « bind » un ou plusieurs VkDescriptorSet(s) avec un VkPipelineLayout

Puis on « bind » un VkPipeline

Puis on « bind » toutes les ressources genre index/vertex buffer avec les offsets et tout

Example :

*vkCmdBindDescriptorSets*(VkPipelineLayout layout, uint32\_t firstDescriptor, uint32\_t descriptorCount, VkDescriptorSet\* sets) ;

*vkCmdBindPipeline*(VkPipeline pipeline);

Le **pipeline** c’est l’ensemble des **étapes** qui vont être exécute sur la carte graphique. Certaines de ces étapes sont des **programmes** éditables, **LES SHADERS**.

Lors de l’exécution d’un pipeline un shader peu être appelé plusieurs fois. Par exemple le Vertex Shader est appelé autant de fois qui il y a de points, le Fragment Shader autant de fois qui il y a de pixel.

Les **shaders** prennent en entrée des « **ressources** ». On en distingue **2** **types**, celles qui sont propre a chaque exécution du shader (exemple les coordonnés du point pour le vs). Et celle qui ne bougera pas (au moins d adresse) durant toute l’exécution du pipeline.

-La **forme** des données qui change a chaque exécution du shader sont appelé **attributs** et elles sont propre a **chaque** pipeline et sont **défini a la création** de celui-ci via l’élément VkPipelineVertexInputStateCreateInfo de la structure VkGraphicsPipelineCreateInfo.

-Les données qui ne bougent pas sont des **bindings** liée au shader via des **VkDescriptorSet** et peuvent changer a chaque exécution tant quelle reste **compatible** avec le shader.

Un shader prend donc en entrée des bindings, ont doit donc lui envoyé **le** **bon** **nombre** de ressources et du **bon type**. A la création du shader ont défini un **descriptorSetLayout** qui doit correspondre aux exigences du shader, c’est en fait un **genre de moule**. A l’appel de ckCmdDescriptorSets(…) on remarque que la commande prend en paramètre un **VkPipelineLayout** qui en fait crée à l’appel de vkCreatePipelineLayout(…) qui prend en paramètre un **descriptorSetLayout**.

Le **pipelineLayout est propre au Pipeline**, on peut donc changer de shader tant que le **DescriptorSetLayout du shader ne change pas**. On peut donc changer le **code source du shader** (sans toucher au layout). Si on veut changer les « valeurs » d’entrée, changer une texture ou on doit changer le **DescriptorSet**

Initialisation

Création du shader

Definition de **DescriptorSetLayoutCreateInfo** (VkDescriptorSetLayoutBindings\*bindings, uint32\_t bindingCount)

Definition de **PipelineLayoutCreateInfo** (**DesciptorSetLayout**\* descriptorSetLayout )

Création du **DescriptorSetLayout** (DescriptorSetLayoutCreateInfo\*)

Création du **PipelineLayout** (PipelineLayoutCreateInfo\*)

Definition des attributs, **VkPipelineVertexInputStateCreateInfo** (**BindingDescription**, **AttributeDescription**)

Definition de *VkGraphicsPipelineCreateInfo*(**VkPipelineLayout, VkPipelineVertexInputStateCreateInfo**)

*vkCreateGraphicsPipeline*(**VkGraphicsPipelineCreateInfo**\* info, VkPipeline\* pipelines)

**En récapitulatif:**

Le **Pipeline**, le **PipelineLayout**, **VkDescriptorSetLayout** sont créer en amont

Le DescriptorSet( et donc les entrées) peut etre changé