What are Descriptor Sets?

Descriptor Sets are an intermediate data structure that tells shaders how to connect information held in GPU memory to groups of related uniform variables and texture sampler declarations in shaders. There are three advantages in doing things this way:

- Related uniform variables can be updated as a group, gaining efficiency.
- Descriptor Sets are activated when the Command Buffer is filled. Different values for the uniform buffer variables can be toggled by just swapping out the Descriptor Set that points to GPU memory, rather than re-writing the GPU memory.
- Values for the shader's uniform buffer variables can be compartmentalized into what quantities change often and what change seldom (scene-level, model-level, draw-level), so that uniform variables need to be re-written no more often than is necessary.

```cpp
for( sporadically )
{
    Bind Descriptor Set #1
    for( each object in the scene )
        Bind Descriptor Set #2
    Do the drawing
}

for( the entire scene )
    Bind Descriptor Set #0
    Do the drawing
```

In OpenGL

OpenGL puts all uniform data in the same "set", but with different binding numbers, so you can get at each one.

Each uniform variable gets updated one-at-a-time.

Wouldn't it be nice if we could update a collection of related uniform variables all at once, without having to update the uniform variables that are not related to this collection?

```cpp
layout( std140, binding = 0 ) uniform mat4 uModelMatrix;
layout( std140, binding = 1 ) uniform mat4 uProjMatrix;
layout( std140, binding = 2 ) uniform mat4 uNormalMatrix;
layout( std140, binding = 3 ) uniform mat3 uLightKaKdKs;
layout( std140, binding = 4 ) uniform vec4 uLightPos;
layout( std140, binding = 5 ) uniform float uTime;
layout( std140, binding = 6 ) uniform int uMode;
layout( std140, binding = 7 ) uniform sampler2D uSampler;

std140 has to do with the alignment of the different data types. It is the simplest, and so we use it in class to give everyone the highest probability that their system will be compatible with the alignment.

Step 1: Descriptor Set Pools

You don't allocate Descriptor Sets on the fly - that is too slow. Instead, you allocate a "pool" of Descriptor Sets during initialization and then pull from that pool later.
Step 2: Define theDescriptor Set Layouts

I think of Descriptor Set Layouts as a kind of "Rowenta Stomach" that allows the Graphics Pipeline data structure to allocate room for the uniform variables and to access them.

1. Include the Descriptor Set Layouts in a Graphics Pipeline Layout
   - Pipeline data structure to allocate room for the uniform variables and to access them.

2. Define the Descriptor Set Layouts
   - Pipeline stage(s): 0, 1, 2, 3
   - Type: VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER, VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER, VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER
   - Stage Flags: VK_SHADER_STAGE_VERTEX_BIT | VK_SHADER_STAGE_FRAGMENT_BIT
   - Pipeline Layout:
     - Set 0
       - Descriptor Count: 1
       - Descriptor Type: VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER
     - Set 1
       - Descriptor Count: 1
       - Descriptor Type: VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER
     - Set 2
       - Descriptor Count: 1
       - Descriptor Type: VK_DESCRIPTOR_TYPE_STORAGE_IMAGE
     - Set 3
       - Descriptor Count: 1
       - Descriptor Type: VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER

Step 3: Include the Descriptor Set Layouts in a Graphics Pipeline Layout

- Pipeline data structure to allocate room for the uniform variables and to access them.
- Include the descriptors in the Graphics Pipeline Layout.
Step 4: Allocating the Memory for Descriptor Sets

```c
vkAllocateDescriptorSets( LogicalDevice, IN &vdsai
```
Step 7: Bind Descriptor Sets into the Command Buffer when Drawing

vkCmdBindDescriptorSets

So, the Pipeline Layout contains the structure of the Descriptor Sets. Any collection of Descriptor Sets that match that structure can be bound into that pipeline.

Sidebar: The Entire Descriptor Set Journey

1. Create the pool of Descriptor Sets for future use
   - VkDescriptorPoolCreateInfo
   - vkCreateDescriptorPool
2. Describe a particular Descriptor Set layout and use it in a specific Pipeline layout
   - VkDescriptorSetLayout
   - vkCreateDescriptorSetLayout
   - vkCreatePipelineLayout
3. Allocate memory for particular Descriptor Sets
   - VkDescriptorSetAllocateInfo
   - vkAllocateDescriptorSets
4. Make a particular Descriptor Set "current" for rendering
   - VkCmdBindDescriptorSets
5. Re-write CPU data into a particular Descriptor Set
   - VkWriteDescriptorSet
   - VkUpdateDescriptorSets

Sidebar: Why Do Descriptor Sets Need to Provide Layout Information to the Pipeline Data Structure?

The pieces of the Pipeline Data Structure are fixed in size – with the exception of the Descriptor Sets and the Push Constants. Each of these two can be any size, depending on what you allocate for them. So, the Pipeline Data Structure needs to know how these two are configured before it can set its own layout. Think of the DS layout as being a particular-sized hole in the Pipeline Data Structure. Any data you have that matches this hole's shape and size can be plugged in there.

The Pipeline Data Structure

Fixed Pipeline Elements
Specific Descriptor Set Layout

Sidebar: Why Do Descriptor Sets Need to Provide Layout Information to the Pipeline Data Structure?

Any set of data that matches the Descriptor Set Layout can be plugged in there.