Frank de Jong

[Email address]

Descriptors

How to generate them

Table of Contents

[General info 2](#_Toc534636709)

[Types 2](#_Toc534636710)

[Sets and bindings in GLSL 3](#_Toc534636711)

# General info

Descriptors are used to describe uniform data inside one or multiple shaders. They can describe matrices, vectors, images, samplers etc. These descriptors are closely interconnected with the SPIR-V language as they have some features that can only be used if one doesn’t use the GLSL or HLSL languages.

## Types

|  |  |  |
| --- | --- | --- |
| Name | Usefulness | GLSL equivalent |
| Storage Image | Low | - |
| Sampler | High | - |
| Sampled Image | High | - |
| Combined Image Sampler | High | Sampler |
| Uniform Texel Buffer | Medium | - |
| Storage Texel Buffer | Low | - |
| Storage Buffer | Medium | - |
| Uniform Buffer | High | Uniform {} |
| Dynamic Uniform Buffer | Low | Uniform {} |
| Dynamic Storage Buffer | Low | Uniform {} |
| Inline Uniform Block | Medium | Uniform {} |
| Input Attachment | High | - |

* **Storage Image:** This type can be used to store an image that can be read from a written to in the same shader. Atomic operations are supported for this type. Maybe this can be used to implement transparent objects into deferred shading by blending when the fragments hit?
* **Sampler:** This type defines how images can be sampled, these are required for any sample operation but cannot be made directly in GLSL.
* **Sampled Image:** This type defines an image resource, it cannot be sampled without a sampler. This can be utilized to minimize sampler creation if we don’t use GLSL.
* **Combined Image Sampler:** This is generally what all texture will be inside shaders build with GLSL because it as no way to separate the two. This simply combines a Sampler and a Sampled Image into one.
* **Uniform Texel Buffer:** This type can be used to store vast amounts of uniform data. For instance, if we need to store joint matrices for an animated model.
* **Storage Texel Buffer:** Same as a Uniform Texel Buffer but now we can also write to it.
* **Storage Buffer:** This is the same as a Uniform Buffer but now we can write to it.
* **Uniform Buffer:** This is how we store all non-sampler uniforms in Vulkan as it cannot be in global scope like in GLSL.
* **Dynamic Uniform Buffer:** Same as a Uniform Buffer but now we can change where the buffer starts dynamically.
* **Dynamic Storage Buffer:** A dynamic Uniform Buffer but now we can write to it as well.
* **Inline Uniform Block:** This will be our way of writing constants into our shader as we cannot use global scope. These values cannot be accessed or changed outside of the shader.
* **Input Attachment:** With this type we can write to the resource in one shader stage and read from it in a later one without having to switch render passes. In a very slim scenario this might be highly optimized (like a deferred renderer).

## Sets and bindings in GLSL

To work with uniform values of any kind in GLSL we need to specify the binding location and set index of the uniform. All uniforms must be stored together in a set. In practice all uniform will probably belong to the same set as the only difference between sets is the flags (which are not often used). So generally, we can specify our uniforms in two ways:

The compiler will add the set index of zero to the last type of layout for us.