Vulkan®

DEVELOPER DAY

Descriptor Update Templates

Markus Tavenrath, NVIDIA

KHRONOS™ GROUP

UBISOFT

MONTRÉAL
APRIL 2018

Descriptor Updates

- Descriptor updates can take a huge portion of time spent in renderer
- If possible try to reuse descriptors
- If it‘s not possible to reuse them let‘s see why descriptor updates are expensive

Descriptor Update Description

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

40 bytes payload to describe single update (64-bit)

Only one pointer of relevance 16 out of 24 bytes overhead

Descriptor Update Cost

- **Structure with description to update a single descriptor needs 64 bytes**
- **Structure has to be filled in for each and every descriptor which needs an update**
- **Classic Vulkan API is good for sparse descriptor updates,**
- **What abbout massive updates?**
  - Might have to recreate DescriptorSets each frame
- **Let‘s take a look at two strategies for large updates**

Update Strategy 1

- Naive approach

```
std::vector<VkWriteDescriptorSet> writes;
writes.reserve(renderEntities.size() * descriptorCount);
for (entity const &: renderEntities) {
  for (resource const &: entity.resources) {
    // VkWriteDescriptorSet content nearly the same for each entity
    writes.push_back(createUpdate(resources));
  }
}
vkUpdateDescriptorSets(..., writes.size(), writes.data(), ...);
```

- Good: Only a single call into the driver
- Bad: Have to fill the n times 64-byte struct for each renderEntitiy
  - 10k entities times 10 descriptors -> ~6.4Mb.
  - Puts stress on memory bandwidth and consumes a lot of memory
  - Might also not want to cache this data or use it as “backend data“ in render entities

Common update loop 2

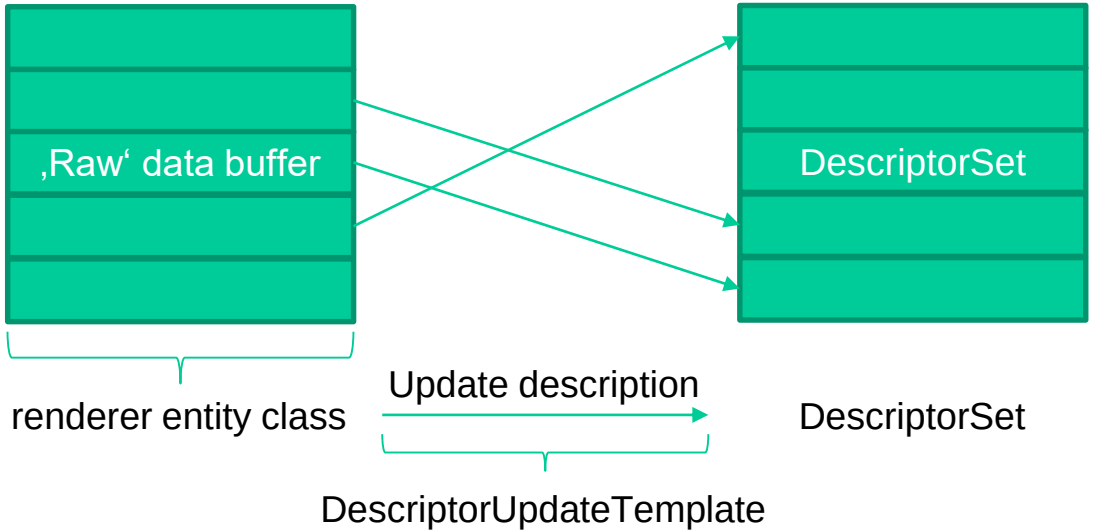
- Memory preserving approach

```
std::vector<VkWriteDescriptorSet> writes();
for (resourceDescription const & : resourceDescriptions) {
  writes.push_back(generateWriteDescriptorInformation(resource));
}
for (entity : renderEntities) {
  for (size_t i = 0; i < writes.size();++i) {
    // write only pImageInfo, pBufferInfo, or pTexelBufferView
    writeEntityDescriptorInformation(entity.res[i],write[i]);
  }
  vkUpdateDescriptorSets(..., writes.size(), writes.data(), ...);
}
```

- Good: Memory consumption down to 640 bytes for 10 descriptors
- Bad: 10k calls into the driver and cannot cache data

DescriptorUpdateTemple

- **Separate update description from update payload**



DescriptorUpdateTemplate

```
typedef struct VkDescriptorUpdateTemplateEntry {
  uint32_t             dstBinding;
  uint32_t             dstArrayElement;
  uint32_t             descriptorCount;
  VkDescriptorType     descriptorType;
  size_t               offset;
  size_t               stride;
} VkDescriptorUpdateTemplateEntryKHR;
```

Location in DescriptorSet

Location in raw pointer
stride is used for arrays

DescriptorUpdateTemplate creation

- During initialization create template for each render entity type

```
std::vector<VkDescriptorUpdateTemplateEntry>
    entries(descriptors.size());
for(size_t i = 0;i < descriptors.size();++i) {
  createUpdateTemplateEntry(i, entries[i]);
}
VkDescriptorUpdateTemplate dut;
VkDescriptorUpdateTemplateCreateInfo createInfo;
// fill createInfo here
vkCreateDescriptorUpdateTemplate(..., &createInfo, ..., &dut);
```

DescriptorUpdate Template Usage

- Renderer can not update a descriptor with a single call passing the renderer entity
- ```
  for (entity const &: renderEntities) {
    vkUpdateDescriptorSetWithTemplate(
      device, descriptorSet, dut, &entity
    );
  }
  ```
- One call required to update a full DescriptorSet from a renderer entity
- No need for intermediate copies
  - Memory friendly
  - Memory bandwidth friendly
- Driver gets a lot of knowlege of update
  - Gives opportunities for more efficient updates
- Makes DescriptorSet updates way cheaper
  - Though still not free, keep DescriptorSets alive and reuse them if possible!
  ```

Summary

- **DescriptorUpdateTemplates can remove the need to copy descripts for updates**
  - VK_KHR_descriptor_update_template extension in Vulkan >= 1.0.37
  - Core Vulkan 1.1
  - According to vulkan.gpuinfo.org it's supported by at least Intel, AMD and NVIDIA
- **Pass in your custom data structure and let the driver interpret it**
- **Saves memory, memory bandwidth and code logic for updates**
- **Also gives drivers more information ahead for optimizations**