User code

One of the main concerns when designing a library is the user code. How the user code will look like, if it will be readable and comfortable to write.

Original examples repository from `ash` crate has 1820 lines of Rust code. Current examples repository adapted to Vulkayes has 1485 lines of Rust code. This is a difference of 335 lines of Rust code.

Below is an example of the code with same functionality from the original examples and from the current ones. The code after is two times shorter than the original code while exposing the same functionality and providing static validation guarantees.

Before:

```
let vertex_input_buffer_info = vk::BufferCreateInfo {
    size: std::mem::size_of_val(&vertices) as u64,
    usage: vk::BufferUsageFlags::VERTEX_BUFFER,
    sharing_mode: vk::SharingMode::EXCLUSIVE,
    ..Default::default()
};
let vertex_input_buffer = base
    .device
    .create_buffer(&vertex_input_buffer_info, None)
    .unwrap();
let vertex_input_buffer_memory_req = base
    .device
    .get_buffer_memory_requirements(vertex_input_buffer);
let vertex_input_buffer_memory_index = find_memorytype_index(
    &vertex_input_buffer_memory_req,
    &base.device_memory_properties,
    vk::MemoryPropertyFlags::HOST_VISIBLE | vk::MemoryPropertyFlags::HOST_COHERENT,
)
.expect("Unable to find suitable memorytype for the vertex buffer.");

let vertex_buffer_allocate_info = vk::MemoryAllocateInfo {
    allocation_size: vertex_input_buffer_memory_req.size,
    memory_type_index: vertex_input_buffer_memory_index,
    ..Default::default()
};
let vertex_input_buffer_memory = base
    .device
    .allocate_memory(&vertex_buffer_allocate_info, None)
    .unwrap();

let vert_ptr = base
    .device
    .map_memory(
        vertex_input_buffer_memory,
        0,
        vertex_input_buffer_memory_req.size,
        vk::MemoryMapFlags::empty(),
    )
    .unwrap();
let mut slice = Align::new(
    vert_ptr,
    align_of::<Vertex>() as u64,
    vertex_input_buffer_memory_req.size,
);
slice.copy_from_slice(&vertices);
```

```
base.device.unmap_memory(vertex_input_buffer_memory);
base.device
    .bind_buffer_memory(vertex_input_buffer, vertex_input_buffer_memory, 0)
    .unwrap();
```

After:

```
let vertex_buffer = {
    let buffer = Buffer::new(
        base.device.clone(),
        std::num::NonZeroU64::new(std::mem::size_of_val(&vertices) as u64).unwrap(),
        vk::BufferUsageFlags::VERTEX_BUFFER,
        base.present_queue.deref().into(),
        buffer::params::AllocatorParams::Some {
            allocator: &base.device_memory_allocator,
            requirements: vk::MemoryPropertyFlags::HOST_VISIBLE
                | vk::MemoryPropertyFlags::HOST_COHERENT
        },
        Default::default()
    )
    .expect("Could not create index buffer");

    let memory = buffer.memory().unwrap();
    memory
        .map_memory_with(|mut access| {
            access.write_slice(&vertices, 0, Default::default());
            MappingAccessResult::Unmap
        })
        .expect("could not map memory");

    buffer
};
```