Vertex Buffers

What is a Vertex Buffer?

Vertex Buffers are how you draw things in Vulkan. They are very much like Vertex Buffer Objects in OpenGL, but more detail is exposed to you (a lot more...).

But, the good news is that Vertex Buffers are really just ordinary Data Buffers, so some of the functions will look familiar to you.

First, a quick review of computer graphics geometry...

Geometry vs. Topology

Where things are (e.g., coordinates)

Geometry = changed
Topology = same (1-2-3-4-1)

Topology: How things are connected

Geometry = same
Topology = changed (3-2-4-3-1)

Vulkan Topologies

typedef enum VkPrimitiveTopology
{
 VK_PRIMITIVE_TOPOLOGY_POINT_LIST = 0,
 VK_PRIMITIVE_TOPOLOGY_LINE_LIST = 1,
 VK_PRIMITIVE_TOPOLOGY_LINE_STRIP = 2,
 VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST = 3,
 VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP = 4,
 VK_PRIMITIVE_TOPOLOGY_TRIANGLE_FAN = 5,
 VK_PRIMITIVE_TOPOLOGY_LINE_LIST_WITH_ADJACENCY = 6,
 VK_PRIMITIVE_TOPOLOGY_LINE_STRIP_WITH_ADJACENCY = 7,
 VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST_WITH_ADJACENCY = 8,
 VK_PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP_WITH_ADJACENCY = 9,
 VK_PRIMITIVE_TOPOLOGY_PATCH_LIST = 10,
} VkPrimitiveTopology;

OpenGL Topologies – Polygon Requirements

Polygons must be:
- Convex and
- Planar
Polygons are traditionally:
• CCW when viewed from outside the solid object

GL_TRIANGLES

It’s not absolutely necessary, but there are possible optimizations if you are consistent

Polygons must be:
• Convex and
• Planar

We can go all mathematical here, but let's go visual instead. In a convex polygon, a line between any two points inside the polygon never leaves the inside of the polygon.

Graphics polygon-filling hardware can be highly optimized if you know that, no matter what direction you fill the polygon in, there will be two and only two intersections between the scanline and the polygon's edges.

There is an open source library to break a non-convex polygon into convex polygons. It is called Polypartition, and is found here:
https://github.com/ivanfratric/polypartition

If you ever need to do this, contact me. I have working code …
**Vertex Orientation Issues**

Thanks to OpenGL, we are all used to drawing in a right-handed coordinate system. Internally, however, the Vulkan pipeline uses a left-handed system: the best way to handle this is to continue to draw in a RH coordinate system and then fix it up in the projection matrix, like this:

\[
\text{ProjectionMatrix}[1][1] = -1.0;
\]

This is like saying “Y’ = -Y”.

---

**A Colored Cube Example**

```
static GLuint CubeTriangleIndices[] = {
    { 0, 2, 3 },
    { 0, 3, 1 },
    { 4, 5, 7 },
    { 4, 7, 6 },
    { 1, 3, 7 },
    { 1, 7, 5 },
    { 0, 4, 6 },
    { 0, 6, 2 },
    { 2, 6, 7 },
    { 2, 7, 3 },
    { 0, 1, 5 },
    { 0, 5, 4 }
};
```

---

**Triangles in an Array of Structures**

```
struct vertex {
    glm::vec3 position;
    glm::vec3 normal;
    glm::vec3 color;
    glm::vec2 texCoord;
};
```

```
struct vertex VertexData[] = {
    // triangle 0-2-3:
    // vertex #0:
    { -1., -1., -1. },
    {  0.,  0., -1. },
    {  0.,  0.,  0. },
    {  1., 0. },
    // vertex #2:
    { -1.,  1., -1. },
    {  0.,  0., -1. },
    {  0.,  1.,  0. },
    {  1., 1. },
    // vertex #3:
    {  1.,  1., -1. },
    {  0.,  0., -1. },
    {  1.,  1.,  0. },
    {  0., 1. },
};
```

---

**Vertex Orientation Issues**

This object was modeled such that triangles that face the viewer will look like their vertices are oriented CCW (this is detected by looking at vertex orientation at the start of the rasterization).

Because this 3D object is closed, Vulkan can save rendering time by not even bothering with triangles whose vertices look like they are oriented CW. This is called backface culling.

Vulkan's change in coordinate systems can mess up the backface culling. So I recommend, at least at first, that you do no culling.

---

**Filling the Vertex Buffer**

```
MyBuffer MyVertexDataBuffer;
InitMyVertexDataBuffer(sizeof(VertexData), &MyVertexDataBuffer);
FillMyDataBuffer(MyVertexDataBuffer, (void *) VertexData);
```

---

**What Init05DataBuffer Does**

```
VkResult Init05DataBuffer(VkDeviceSize size, VkBufferUsageFlags usage, OUT MyBuffer * pMyBuffer)
{
    VkResult result = VK_SUCCESS;
    VkBufferCreateInfo vbci;
    vbci.sType = VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO;
    vbci.pNext = nullptr;
    vbci.flags = 0;
    vbci.size = pMyBuffer->size = size;
    vbci.usage = usage;
    vbci.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
    vbci.queueFamilyIndexCount = 0;
    vbci.pQueueFamilyIndices = (const uint32_t *)nullptr;
    result = vkCreateBuffer(LogicalDevice, &vbci, PALLOCATOR, OUT &pMyBuffer->buffer);
    VkMemoryRequirements vmr;
    vkGetBufferMemoryRequirements(LogicalDevice, pMyBuffer->buffer, OUT &vmr);
    VkMemoryAllocateInfo vmai;
    vmai.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
    vmai.pNext = nullptr;
    vmai.allocationSize = vmr.size;
    vmai.memoryTypeIndex = FindMemoryThatIsHostVisible();
    VkDeviceMemory vdm;
    result = vkAllocateMemory(LogicalDevice, &vmai, PALLOCATOR, OUT &vdm);
    pMyBuffer->vdm = vdm;
    result = vkBindBufferMemory(LogicalDevice, pMyBuffer->buffer, vdm, 0);
    return result;
}
```

---

**Filling the Vertex Buffer**

```
MyBuffer MyVertexDataBuffer;
InitMyVertexDataBuffer(sizeof(VertexData), &MyVertexDataBuffer);
FillMyDataBuffer(MyVertexDataBuffer, (void *) VertexData);
```

---

**A Colored Cube Example**

```
static GLuint CubeTriangleIndices[] = {
    { 0, 2, 3 },
    { 0, 3, 1 },
    { 4, 5, 7 },
    { 4, 7, 6 },
    { 1, 3, 7 },
    { 1, 7, 5 },
    { 0, 4, 6 },
    { 0, 6, 2 },
    { 2, 6, 7 },
    { 2, 7, 3 },
    { 0, 1, 5 },
    { 0, 5, 4 }
};
```
The Vulkan Pipeline

Vertex Input Stage

Input Assembly

Color Blending Stage

Presentation Shader, Geometry Shader

Fragment Shader Stage

Tessellation, Geometry

Input Stage

The Vulkan Pipeline

Vertex Shader module

Fragment Shader module

Creation, init

Colour Blending parameters

Specialization info

Vertex Shader module

Creation, init

Colour Blending Stage

Input Stage

Fragment Shader Stage

Tessellation, Geometry

Transformation Shaders, Geometry Shader

Vertex Input Stage

Input Assembly

Dynamic State

StencilTestEnable

DepthWriteEnable

DepthTestEnable

Which states are dynamic

FrontFace

CullMode

PolygonMode

DiscardEnable

Depth Clamping

Scissoring

Topology

Vertex Input binding

result = vkCreateGraphicsPipelines( LogicalDevice, VK_NULL_HANDLE, 1, IN & data structure that holds (what OpenGL would call) the state, including how to parse its input.

We will come to the Pipeline later, but for now, know that a Vulkan Pipeline is essentially a very large data structure that holds (what OpenGL would call) the state, including how to parse its input.

C/C++

GLSL

layout( location = 0 ) in vec3 aVertex;

layout( location = 1 ) in vec3 aColor;

layout( location = 3 ) in vec2 aTexCoord;

Vulkan Pipeline

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Telling the Pipeline about its Input

Telling the Pipeline about its Input

Telling the Command Buffer what Vertices to Draw

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage

Pipeline Layout

Vulkan

Shader Stage

Input Stage