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...
OpenGL Topologies – Vertex Order Matters

VK_LINE_STRIP

V0
V1
V2
V3

VK_LINE_STRIP

V0
V1
V2
V3

What does “Convex Polygon” Mean?

We could go all mathematical, 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.

Convex
Not Convex

V0
V1
V2
V3

OK, now let’s go all mathematical. In a convex polygon, every interior angle is between 0° and 180°.

Convex
Not Convex

V0
V1
V2
V3

Between 0° and 180°
Greater than 180°

Why is there a Requirement for Polygons to be Convex?

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.

Convex
Not Convex

Why is there a Requirement for Polygons to be Planar?

Graphics hardware assumes that a polygon has a definite front and a definite back, and that you can only see one of them at a time.

OK
Not OK

What if you need to display Polygons that are not Convex?

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 …

OK
Not OK
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

- **Cube Triangle Indices**

\[
\begin{align*}
\text{Triangle 0-2-3:} & \quad \begin{bmatrix} 0, 2, 3 \end{bmatrix}\\
\text{Triangle 0-3-1:} & \quad \begin{bmatrix} 0, 3, 1 \end{bmatrix}\\
\text{Triangle 4-5-7:} & \quad \begin{bmatrix} 4, 5, 7 \end{bmatrix}\\
\text{Triangle 4-7-6:} & \quad \begin{bmatrix} 4, 7, 6 \end{bmatrix}\\
\text{Triangle 1-3-7:} & \quad \begin{bmatrix} 1, 3, 7 \end{bmatrix}\\
\text{Triangle 1-7-5:} & \quad \begin{bmatrix} 1, 7, 5 \end{bmatrix}\\
\text{Triangle 0-4-6:} & \quad \begin{bmatrix} 0, 4, 6 \end{bmatrix}\\
\text{Triangle 0-6-2:} & \quad \begin{bmatrix} 0, 6, 2 \end{bmatrix}\\
\text{Triangle 2-6-7:} & \quad \begin{bmatrix} 2, 6, 7 \end{bmatrix}\\
\text{Triangle 2-7-3:} & \quad \begin{bmatrix} 2, 7, 3 \end{bmatrix}\\
\text{Triangle 0-1-5:} & \quad \begin{bmatrix} 0, 1, 5 \end{bmatrix}\\
\text{Triangle 0-5-4:} & \quad \begin{bmatrix} 0, 5, 4 \end{bmatrix}
\end{align*}
\]

### Triangles in an Array of Structures

- **Vertex Structure**

```
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. },
};
```

### Filling the Vertex Buffer

- **Function**

```
MyBuffer MyVertexBuffer;
Init05MyVertexBuffer(sizeof(VertexData), &MyVertexBuffer);
Fill05DataBuffer(MyVertexBuffer, (void *) VertexData);
```

```
VkResult Init05MyVertexBuffer(IN VkDeviceSize size, OUT MyBuffer * pMyBuffer)
{
    VkResult result = Init05DataBuffer(size, VK_BUFFER_USAGE_VERTEX_BUFFER_BIT, pMyBuffer);
    return result;
}
```

```
VkResult Init05DataBuffer(IN 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(myLogicalDevice, &vbci, pAllocator, OUT &pMyBuffer->buffer);

    VkMemoryRequirements vmr;
    vkGetBufferMemoryRequirements(myLogicalDevice, IN 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(myLogicalDevice, IN &vmai, pAllocator, OUT &vdm);
    pMyBuffer->vdm = vdm;

    result = vkBindBufferMemory(myLogicalDevice, pMyBuffer->buffer, IN vdm, 0); // 0 is the offset
    return result;
}
```

### A Reminder of What Init05DataBuffer Does

```
void vkGetBufferMemoryRequirements(VkDevice device, VkBuffer buffer, VkMemoryRequirements * pMemoryRequirements)
{
    VkDeviceMemoryInfo memInfo;
    vkGetDeviceMemoryInfo(device, IN deviceMemory, OUT &memInfo);

    VkBufferMemoryRequirementsInfo memoryRequirementsInfo;
    memoryRequirementsInfo.sType = VK_STRUCTURE_TYPE_BUFFER_MEMORY_REQUIREMENTS_INFO;
    memoryRequirementsInfo.pNext = nullptr;
    memoryRequirementsInfo.buffer = buffer;

    VkResult result = vkGetBufferMemoryRequirements(myLogicalDevice, IN buffer, OUT &memoryRequirementsInfo);

    result = vkMapMemory(myLogicalDevice, deviceMemory, &memoryRequirementsInfo, VK_MAP_READ_BIT, IN alignment, OUT mappedMemory);
    memoryRequirementsInfo.alignment = alignment;

    result = vkUnmapMemory(myLogicalDevice, mappedMemory);
    return result;
}
```

```
void vkGetDeviceMemoryInfo(VkDevice device, VkMemoryGetRequirementsInfo * pRequirements)
{
    VkMemoryRequirementsInfo memoryRequirementsInfo;
    memoryRequirementsInfo.sType = VK_STRUCTURE_TYPE_MEMORY_REQUIREMENTS_INFO;
    memoryRequirementsInfo.pNext = nullptr;
    memoryRequirementsInfo.device = device;

    VkResult result = vkGetDeviceMemoryRequirements(device, IN device, OUT &memoryRequirementsInfo);

    memoryRequirementsInfo.alignment = 16;
    memoryRequirementsInfo.buffer = buffer;
    memoryRequirementsInfo.alignment = alignment;

    result = vkGetBufferMemoryRequirements(device, IN buffer, OUT &memoryRequirementsInfo);

    result = vkMapMemory(device, deviceMemory, &buffer, VK_MAP_READ_BIT, IN alignment, OUT mappedMemory);
    memoryRequirementsInfo.alignment = alignment;

    result = vkUnmapMemory(device, mappedMemory);
    return result;
}
```
The Vulkan Pipeline

Telling the Pipeline about its Input

Telling the Command Buffer what Vertices to Draw