Instancing

Instancing -- What and why?

- Instancing is the ability to draw the same object multiple times
- It uses all the same vertices and graphics pipeline each time
- It avoids the overhead of the program asking to have the object drawn again, letting the GPU/driver handle all of that

vkCmdDraw(CommandBuffers[nextImageIndex], vertexCount, instanceCount, firstVertex, firstInstance);

But, this will only get us multiple instances of identical objects drawn on top of each other. How can we make each instance look differently?

Making each instance look differently -- Approach #1

Use the built-in vertex shader variable gl_InstanceIndex to define a unique display property, such as position or color.

\[
\text{gl_InstanceIndex} \text{ starts at 0}
\]

In the vertex shader:

```cpp
int NUMINSTANCES = 16;
float DELTA = 3.0;
float xdelta = DELTA * float(gl_InstanceIndex % 4);
float ydelta = DELTA * float(gl_InstanceIndex / 4);
vec3 vColor = vec3(1., float((1.+gl_InstanceIndex))/float(NUMINSTANCES), 0.);
vec4 vertex = vec4(aVertex.xyz + vec3(xdelta, ydelta, 0.), 1.);
gl_Position = PVM * vertex;
```

Making each instance look differently -- Approach #2

Put the unique characteristics in a uniform buffer and reference them.

```cpp
layout( std140, set = 3, binding = 0 ) uniform colorBuf
{
vec3 uColors[1024];
}
Colors;
```

```
int index = gl_InstanceIndex % 1024; // 0 - 1023
vColor = Colors.uColors[index];
gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
```

Making each instance look differently -- Approach #3

Put a series of unique characteristics in a data buffer, one element per instance.

Read a new characteristic for each instance.

```
int index = gl_InstanceIndex % 1024; // 0 - 1023
vColor = Colors.uColors[index];
gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
```
This is just the Vertex Input State Portion of the Graphics Pipeline Structure.

How We Constructed the Graphics Pipeline Structure Before

- `VkVertexInputAttributeDescription` array:
  - `offset`, `format`, `binding`, `location`
  - This array contains information about each vertex attribute.

- `VkVertexInputBindingDescription` array:
  - `inputRate`, `stride`, `binding`
  - Describes the layout of the vertex attributes.

- `vkCreateGraphicsPipelines()` function:
  - Creates a graphics pipeline using the specified parameters.

How We Constructed the Graphics Pipeline Structure Now

- `VkPipelineVertexInputStateCreateInfo`:
  - `vertexBindingDescriptionCount`, `vertexAttributeDescriptionCount`
  - More modern way of describing the vertex input state.

- Creating a data buffer with one glm::vec3 (to hold r, g, b) for each Instance.

This definition says that we should advance through the input buffer by this much every time we hit a new vertex.
How We Construct the Graphics Pipeline Structure Now

Let's assign a different color per Instance.
Create a data buffer with one glm::vec3 (to hold r, g, b) for each Instance.

```
VkPipelineVertexInputStateCreateInfo vpvisci;
vpvisci.sType = VK_STRUCTURE_TYPE_PIPELINE_VERTEX_INPUT_STATE_CREATE_INFO;
vpvisci.pNext = nullptr;
vpvisci.flags = 0;
vpvisci.vertexBindingDescriptionCount = 2;
vpvisci.pVertexBindingDescriptions = vvibd;
vpvisci.vertexAttributeDescriptionCount = 5;
vpvisci.pVertexAttributeDescriptions = vviad;
```

Note: same names as before, but different sizes

```
VkGraphicsPipelineCreateInfo vgpci;
vgpci.sType = VK_STRUCTURE_TYPE_GRAPHICS_PIPELINE_CREATE_INFO;
vgpci.pNext = nullptr;
vgpci.flags = 0;
.
vgpci.pVertexInputState = &vpvisci;
.
result = vkCreateGraphicsPipelines( LogicalDevice, VK_NULL_HANDLE, 1, IN &vgpci, PALLOCATOR, OUT pGraphicsPipeline );
```

How We Write the Vertex Shader Now

```
#version 400
#extension GL_ARB_separate_shader_objects : enable
#extension GL_ARB_shading_language_420pack : enable
.
layout( location = 0 ) in vec3 aVertex;
layout( location = 1 ) in vec3 aNormal;
layout( location = 2 ) in vec3 aColor;
layout( location = 3 ) in vec2 aTexCoord;
layout( location = 4 ) in vec3 aInstanceColor;
layout ( location = 0 ) out vec3 vNormal;
layout ( location = 1 ) out vec3 vColor;
layout ( location = 2 ) out vec2 vTexCoord;

void main( )
{
  mat4 PVM = Matrices.uProjectionMatrix * Matrices.uViewMatrix * Matrices.uModelMatrix;
  vNormal = normalize( vec3( Matrices.uNormalMatrix * vec4(aNormal, 1.) ) );
  //vColor = aColor;
  vColor = aInstanceColor;
  vTexCoord = aTexCoord;
  gl_Position = PVM * vec4( aVertex, 1. );
}
```