At this point, `BottomLevelAccelerationStructure` is just a handle. We need to call `vkCmdBuildAccelerationStructure( )` to populate it.

```cpp
vasbgi.scratchData.deviceAddress = ...;
vasbgi.dstAccelerationStructure = VK_NULL_HANDLE; // will be set later
result = vkCmdBuildAccelerationStructure( CommandBuffer, 1, IN &vasbgi, IN &vasbri );
```

**Creating the Bottom Level Acceleration Structures**

```cpp
// 3
```

**Acceleration Structures**

- A Bottom-level Acceleration Structure (BLAS) reads the vertex data from vertex (and possibly index)
  `VkBuffer`s to determine Axis-Aligned Bounding Boxes (AABBs).
- You can also supply your own AABB information to a BLAS.
- A single Top-level Acceleration Structure (TLAS) holds Instances, which are transformations and
  pointers to (potentially) multiple BLASes.
- Each BLAS is essentially used as a Model Coordinate bounding box, while the single TLAS is used as a World Coordinate bounding box.
Submitting the BLAS `vkCmdBuildAccelerationStructure` Command

```cpp
vkCmdBuildAccelerationStructure(CommandBuffer, 1, &vasbgi, &vasbri);
```

Creating the Top Level Acceleration Structure

```cpp
vkCreateAccelerationStructureKHR(LogicalDevice, &vasci, &vas);
```

Building the Top Level Acceleration Structure

```cpp
vkCreateAccelerationStructureKHR(LogicalDevice, &vasci, &vas);
```

Creating the Bottom Level Acceleration Structure

```cpp
vkCreateAccelerationStructureKHR(LogicalDevice, &vasci, &vas);
```

Submitting the `vkCmdBuildAccelerationStructure` Structure

```cpp
evkaCmdBuildAccelerationStructureKHR(LogicalDevice, &vas);
```
Other Information for Creating the Top Level Acceleration Structure

VkAccelerationStructureGeometryAabbsData

```
vsgad.sType = VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_GEOMETRY_AABBS_DATA;
vsgad.pNext = nullptr;
vsgad.data = << VkDeviceOrHostAddressConst >>;
vsgad.stride = 0;
```

VkAccelerationStructureInstance

```
vasi.transform = << VkTransformMatrix >>;
vasi.instanceCustomIndex = << uint32_t:24 >>
vasi.mask = 0xff
instanceShaderBindingTableRecordOffset = << uint32_t:24 >>
vasi.flags = 0;
vasi.accelerationStructureReference = << uint64_t >>;
```

VkAabbPositions

```
vap minX, . minY, . minZ;
vap maxX, . maxY, . maxZ;
```

VkTransformMatrix

```
vtm.matrix = float [3][4]; // glm::mat3x4
```

Why a 3x4 Matrix?

Because we are not doing perspective here, we really don't need the bottom row

```
glm::mat4 mat = glm::mat4( 1 );
mat = glm::rotate( mat, rotAngle, zaxis );
vtm.matrix = glm::mat3x4( mat );
```