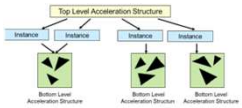
1

Ray-Tracing: Acceleration Structures

Mike Bailey
mjb@cs.oregonstate.edu

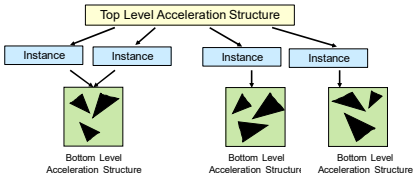


1

Acceleration Structures

2

- A Bottom-level Acceleration Structure (BLAS) reads the vertex data from vertex (and possibly index VkBuffers) 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 9potentially) 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.



2

Creating the *Bottom Level* Acceleration Structures

3

```
VkAccelerationStructure                 BottomLevelAccelerationStructure;

VkAccelerationStructureGeometryTrianglesData    vasgtd;
        vasgtd.sType = VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_GEOMETRY_TRIANGLES_DATA;
        vasgtd.pNext = nullptr;
        vasgtd.vertexFormat = VK_FORMAT_VEC3;                   // = VK_FORMAT_R32G32B32_SFLOAT
        vasgtd.vertexData.deviceAddress = MyVertexDataBuffer.vdm;  // device address of the array of vertex structs
        vasgtd.vertexStride = sizeof( struct vertex );          // how to get from one vertex to the next
        vasgtd.maxVertex = sizeof(Vertices) / sizeof( struct vertex ) - 1;
        vasgtd.indexData = VK_NULL_HANDLE;                      // we're not using index data,
        vasgtd.indexType = VK_INDEX_TYPE_UINT_32;               //      but if we were, they would be 32-bit unsigned in
        vasgtd.transformData = 0;                               // no transform here

VkAccelerationStructureGeometryData             vasgd;          // this is a union, not a struct
        vasgd.triangles = vasgtd;       // VkAccelerationStructureGeometryTrianglesData
        //vasgd.aabbs = << VkAccelerationStructureGeometryAabbsData >>;
        //vasgd.instances = << VkAccelerationStructureGeometryInstancesData >>;

VkAccelerationStructureGeometry                 vasg;
        vasg.sType = VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_GEOMETRY;
        vasg.pNext = nullptr;
        vasg.geometryType = VK_GEOMETRY_TYPE_TRIANGLES;
                VK_GEOMETRY_TYPE_TRIANGLES or VK_GEOMETRY_TYPE_AABS or VK_GEOMETRY_TYPE_INSTANCE
        vasg.geometry = vasgd;
        vasg.flags = VK_GEOMETRY_OPAQUE_BIT;            // members of VkGeometryFlagBits
        vasg.dstAccelerationStructure = BottomLevelAccelerationStructure;
```

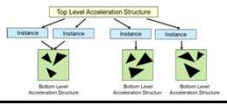
3

Creating the *Bottom Level* Acceleration Structures

4

```
VkAccelerationStructureBuildRangeInfo           vasbri;
        vasbri.pritmitiveCount = sizeof(Vertices) / sizeof(struct vertex) / 3;      // # triangles, # bounding boxes, or # instances
        vasbri.primitiveOffset = 0;
        vasbri.firstVertex = 0;
        vasbri.transformOffset = 0;

VkAccelerationStructureBuildGeometryInfo        vasbgi;
        vasbgi.sType = VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_BUILD_GEOMETRY_INFO;
        vasbgi.pNext = nullptr;
        vasbgi.type = VK_ACCELERATION_STRUCTURE_TYPE_BOTTOM_LEVEL;
        vasbgi.flags = VK_BUILD_ACCELERATION_STRUCTURE_PREFER_FAST_TRACE_BIT;
        vasbgi.mode = VK_BUILD_ACCELERATION_STRUCTURE_MODE_BUILD;
        vasbgi.srcAccelerationStructure = VK_NULL_HANDLE;           // not re-building
        vasbgi.geometryCount = 1;
        vasbgi.pGeometries = vasg;                                  // ptr to array of VkAccelerationStructureGeometry structs
        vasbgi.ppGeometries = nullptr;
        vasbgi.dstAccelerationStructure     = VK_NULL_HANDLE;       // will be set later
        vasbgi.scratchData.deviceAddress = <<VkDeviceOrHostAddress >>;  // will be set later
```



4

Creating the *Bottom Level* Acceleration Structures

5

```
VkAccelerationStructureBuildSizesInfo           vasbsi;
        vasbsi.sType = VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_BUILD_SIZES_INFO;

vkGetAccelerationStructureBuildSizes( LogicalDevice, VK_ACCELERATION_STRUCTURE_BUILD_TYPE_DEVICE,
            IN &vasbgi, IN &vasbri.primitiveCount, OUT &vasbsi );

<< vasbsi.accelerationStructureSize is how big the buffer should be >>
<< use VK_BUFFER_USAGE_{ACCELERATION_STRUCTURE_STORAGE | SHADER_DEVICE_ADDRESS |
STORAGE_BUFFER} _BIT when creating that buffer >>

VkAccelerationStructureCreateInfo               vasci;
        vasci.sType = VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_CREATE_INFO;
        vasci.pNext = nullptr;
        vasci.createFlags = 0;
        vasci.buffer = MyBottomLevelBuffer.buffer;              // where BLAS will be stored
        vasci.offset = 0;
        vasci.size = vasbsi.accelerationStructureSize;
        vasci.type = VK_ACCELERATION_STRUCTURE_TYPE_BOTTOM_LEVEL;
        vasci.deviceAddress = nullptr;

VkAccelerationStructure         BottomLevelAccelerationStructure;
result = vkCreateAccelerationStructure( LogicalDevice, IN &vasci, PALLOCATOR, OUT &BottomLevelAccelerationStructure );
vasbgi.dstAccelerationStructure = BottomLevelAccelerationStructure;
vasbgi.scratchData.deviceAddress = MyBottomLevelBuffer.vdm;
```

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

5

Building the *Bottom Level* Acceleration Structures

6

```
VkAccelerationStructureBuildGeometryInfo        vasbgi;         // already created…
. . .
VkAccelerationStructureBuildRangeInfo           vasbri;         // already created…
. . .
vkCmdBuildAccelerationStructure( CommandBuffer, 1, IN &vasbgi, IN &vasbri );
```

The BLAS vkCmdBuildAccelerationStructure command must be submitted right away. It must complete before attempting to build a TLAS.

6

Submitting the BLAS vkCmdBuildAccelerationStructure Command

7

```
VkCommandBufferBeginInfo                vcbbi;
    vcbbi.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO;
    vcbbi.pNext = nullptr;
    vcbbi.flags = VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT;
    vcbbi.pInheritanceInfo = (VkCommandBufferInheritanceInfo *)nullptr;

result = vkBeginCommandBuffer( AccelerationStructureCommandBuffer, IN &vcbbi);

vkCmdBuildAccelerationStructure( AccelerationStructureCommandBuffer, 1, IN &vasbgi, IN &vasbri );

result = vkEndCommandBuffer( AccelerationStructureCommandBuffer );

VkSubmitInfo                    vsi;
    vsi.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
    vsi.pNext = nullptr;
    vsi.commandBufferCount = 1;
    vsi.pCommandBuffers = &AccelerationStructureCommandBuffer;
    vsi.waitSemaphoreCount = 0;
    vsi.pWaitSemaphores = (VkSemaphore *)nullptr;
    vsi.signalSemaphoreCount = 0;
    vsi.pSignalSemaphores = (VkSemaphore *)nullptr;
    vsi.pWaitDstStageMask = (VkPipelineStageFlags *)nullptr;

result = vkQueueSubmit(Queue, 1, IN & vsi, 0 );

result = vkQueueWaitIdle( Queue );
```

7

Creating the *Top Level* Acceleration Structure

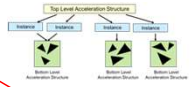
8

```
VkAccelerationStructureDeviceAddressInfo vasdai;
    vasdai.sType = VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_DEVICE_ADDRESS_INFO;
    vasdai.accelerationStructure = BottomLevelAccelerationStructure;

glm::mat4 instanceRotation = glm::rotate( glm::mat4(1.), rotAngle, axis );
VkTransformMatrix                       vtm;
    vtm.matrix = glm::mat3x4( instanceRotation );

VkAccelerationStructureInstance         vasi;
    vasi.transform.matrix = vtm.matrix;
    vasi.mask = 0xff;
    vasi.instanceShaderBindingTableRecordOffset = 0;
    vasi.flags = VK_GEOMETRY_INSTANCE_TRIANGLE_FACING_CULL_DISABLE_BIT;
    vasi.accelerationStructureReference = vkGetAccelerationStructureDeviceAddress( LogicalDevice, &vasdai );

VkAccelerationStructureBuildRangeInfo   vasbri;
    vasbri.primitiveCount = 1                       // 1 instance
    vasbri.primitiveOffset = 0;
    vasbri.firstVertex = 0;
    vasbri.transformOffset = 0;
```



8

Creating the *Top Level* Acceleration Structure

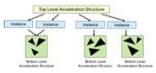
9

```
VkAccelerationStructureGeometryInstancesData    vasgid;
    vasgid.sType = VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_GEOMETRY_INSTANCES_DATA;
    vasgid.pNext = nullptr;
    vasgid.instances.arrayOfPointers = VK_FALSE;
    vasgid.instances.data.deviceAddress = << VkDeviceOrHostAddress of TopLevelAccelerationStructure >>;

VkAccelerationStructureBuildSizesInfo           vasbsi;
    vasbsi.sType = VK_STRUCTURE_TYPE_ACCEERATION_STRUCTURE_BUILD_SIZES_INFO;
VkGetAccelerationStructreBuildSizes(LogicalDevice, VK_ACCELERATION_STRUCTURE_BUILD_TYPE_DEVICE, IN &vasbgi, IN &vasbri.pritmitiveCount, OUT &vasbsi );

VkAccelerationStructureGeometry                 vasg;
    vasg.sType = VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_GEOMETRY;
    vasg.geometryType = VK_GEOMETRY_TYPE_INSTANCES;
    vasg.geometry.instances = vasgid;

VkAccelerationStructureInfo                     vasi;
    vasi.type = VK_ACCELERATION_STRUCTURE_TYPE_TOP_LEVEL;
    vasi.flags = 0;
    vasi.pNext = nullptr;
    vasi.instanceCount = 0;
    vasi.geometryCount = 1;
    vasi.pGeometries   = &vasg;
```



9

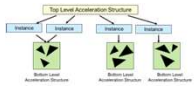
Creating the *Top Level* Acceleration Structure

10

```
VkAccelerationStructureCreateInfo               vascl;
    vasci.sType = VK_STRUCTURE_TYPE_ACCELERATION_STRUCTURE_CREATE_INFO;
    vasci.pNext = nullptr;
    vasci.createFlags = ???;
    vasci.buffer = << where TLAS will be stored >>;
    vasci.offset = 0;
    vasci.size = vasbsi.accelerationStructureSize;
    vasci.type = VK_ACCELERATION_STRUCTURE_TYPE_TOP_LEVEL;
    vasci.deviceAddress = nullptr;

VkAccelerationStructure         TopLevelAccelerationStructure;
result = vkCreateAccelerationStructure( LogicalDevice, IN &vasci, PALLOCATOR, OUT &TopLevelAccelerationStructure );

vasi.dstAccelerationStructure = TopLevelAccelerationStructure;
Create scratch buffer: buildsizes.buildScratchSize, VK_BUFFER_USAGE_STORAGE_BUFFER_BIT
vasi.scratchData.deviceAddress << device address of tlas scratch buffer handle >>
```



At this point, TopLevelAccelerationStructure Is just a handle We need to call vkCmdBuildAccelerationStructure( ) to populate it

10

Building the *Top Level* Acceleration Structure

11

```
. . .

VkAccelerationStructureBuildGeometryInfo        vasbgi;

. . .

VkAccelerationStructureBuildRangeInfo           vasbri;

. . .

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

11

Submitting the vkCmdBuildAccelerationStructure

12

```
VkCommandBufferBeginInfo                vcbbi;
    vcbbi.sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO;
    vcbbi.pNext = nullptr;
    vcbbi.flags = VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT;
    vcbbi.pInheritanceInfo = (VkCommandBufferInheritanceInfo *)nullptr;

result = vkBeginCommandBuffer( AccelerationStructureCommandBuffer, IN &vcbbi);

vkCmdBuildAccelerationStructure( AccelerationStructureCommandBuffer, 1, IN &vasbgi, IN &vasbri );

result = vkEndCommandBuffer( AccelerationStructureCommandBuffer );

VkSubmitInfo                    vsi;
    vsi.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
    vsi.pNext = nullptr;
    vsi.commandBufferCount = 1;
    vsi.pCommandBuffers = &AccelerationStructureCommandBuffer;
    vsi.waitSemaphoreCount = 0;
    vsi.pWaitSemaphores = (VkSemaphore *)nullptr;
    vsi.signalSemaphoreCount = 0;
    vsi.pSignalSemaphores = (VkSemaphore *)nullptr;
    vsi.pWaitDstStageMask = (VkPipelineStageFlags *)nullptr;

result = vkQueueSubmit(Queue, 1, IN & vsi, 0 );

result = vkQueueWaitIdle( Queue );
```

12

Other Information for Creating the *Top* Level Acceleration Structure

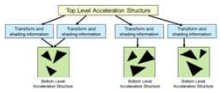
13

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

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

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


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



Oregon State University Computer Graphics

mjb – March 3, 2023

13

Why a 3x4 Matrix?

14

glm::mat4

$$\begin{Bmatrix} x' \\ y' \\ z' \\ 1 \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{Bmatrix} x \\ y \\ z \\ 1 \end{Bmatrix}$$

glm::mat3x4

$$\begin{Bmatrix} x' \\ y' \\ z' \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ & & & \end{bmatrix} \cdot \begin{Bmatrix} x \\ y \\ z \\ 1 \end{Bmatrix}$$

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  );
```

Oregon State University Computer Graphics

mjb – March 3, 2023

14