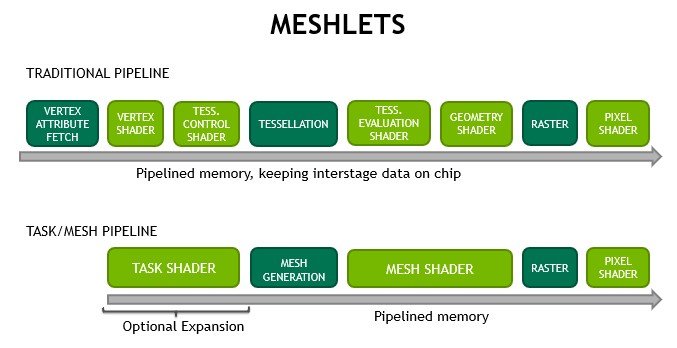
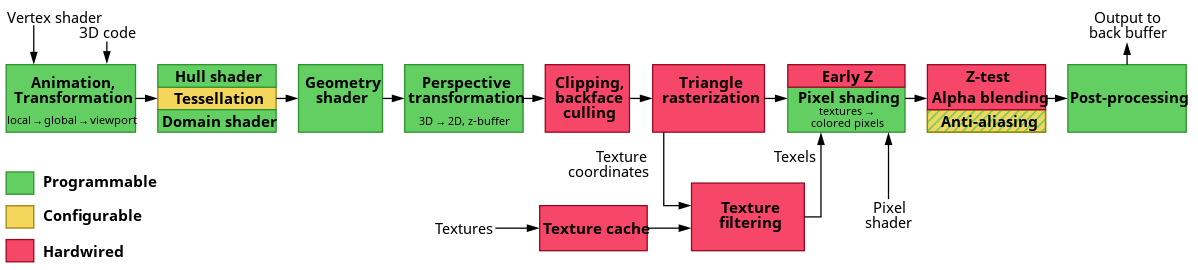
# Pipeline change





https://en.wikipedia.org/wiki/File:3D-Pipeline.svg

# vkCmdDrawMeshTasks\*EXT

**vkCmdDrawMeshTasksEXT** is the simplest as it functions the same as vkCmdDispatch, but dispatches the mesh or task shader in a graphics pipeline with the specified workgroup counts, rather than a compute shader.

Task or Mesh? It depends.

```

pipelineInfo.stageCount = 3;

stages[0].stage = VK\_SHADER\_STAGE\_TASK\_BIT\_NV;

stages[0].module = m\_shaderManager.get(shaders.task);

stages[0].pNext = rss\_info\_ptr;

stages[1].stage = VK\_SHADER\_STAGE\_MESH\_BIT\_NV;

stages[1].module = m\_shaderManager.get(shaders.task\_mesh);

stages[1].pNext = rss\_info\_ptr;

stages[2].stage = VK\_SHADER\_STAGE\_FRAGMENT\_BIT;

stages[2].module = m\_shaderManager.get(shaders.mesh\_fragment);

stages[2].pNext = nullptr;

```

# Meshlet

### What is a meshlet

Subdivide your meshes into small groups of vertices/primitives during the asset building phase.

* Up to 256 vertices/primitives (typical: 128/128)
* Typically, 1 MS workgroup processes 1 meshlet (AMD prefers large workgroups, Maybe not for AMD)
* 1 TS invocation ∼ 1 meshlet

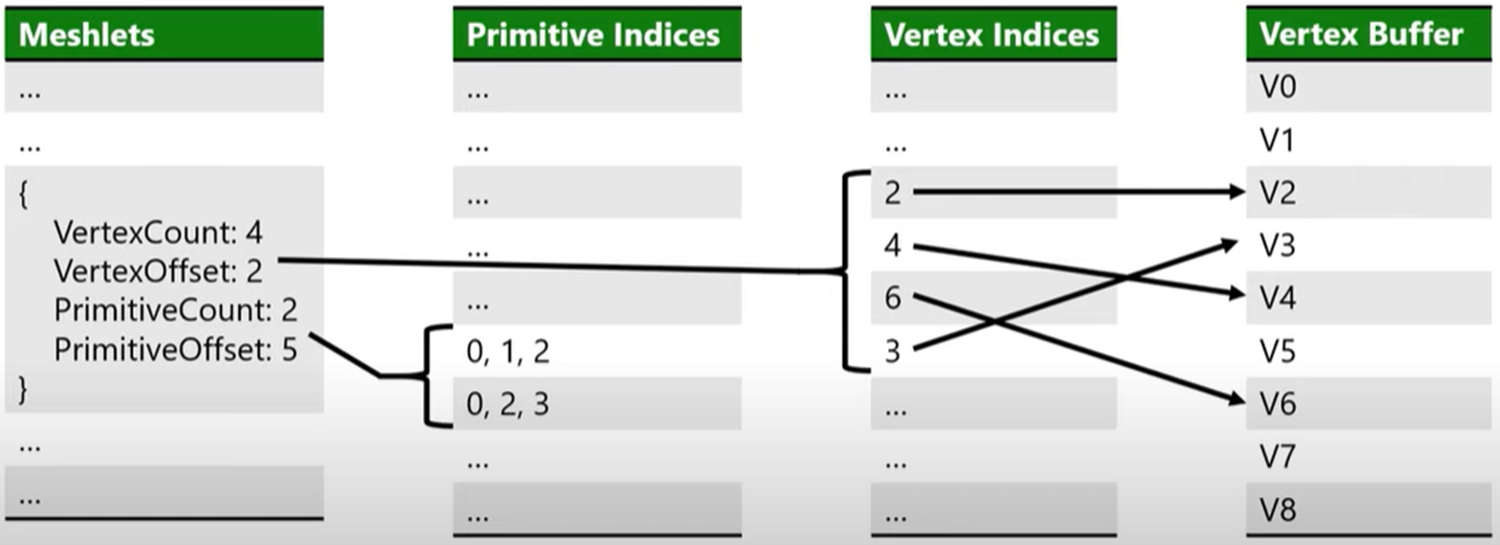


### Meshlet indexing

Vertex buffer: unchanged;

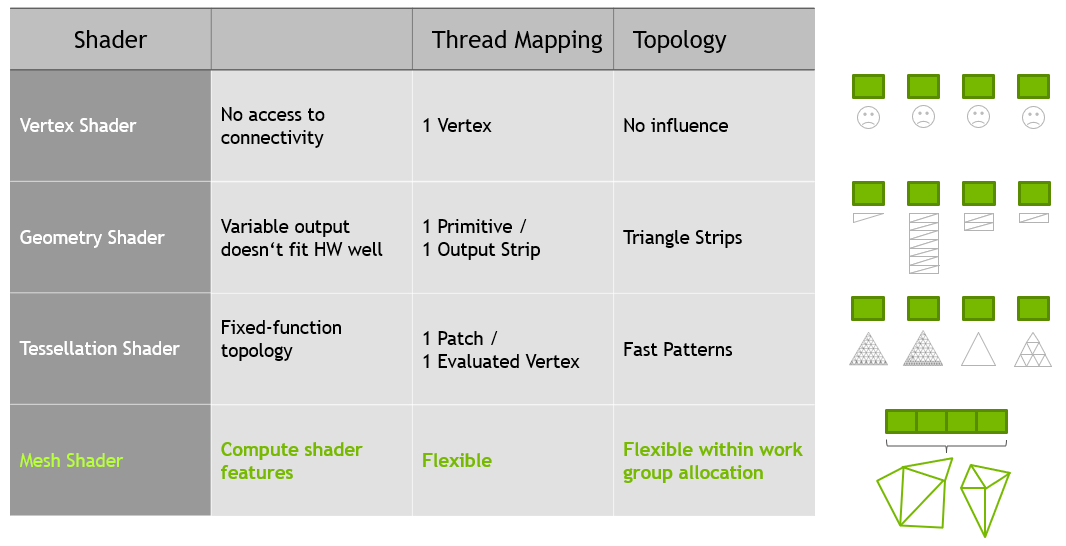
Three buffers replace index buffer:

* Meshlet buffer
* Vertex index list
* Primitive index list



From: Reinventing the Geometry Pipeline: Mesh Shaders in DirectX 12 | Shawn Hargreaves | DirectX Dev Day

# Mesh shader basic



https://developer.nvidia.com/blog/introduction-turing-mesh-shaders/

Task shader: How many mesh shader workgroups do you want(How many do you want for Task shader?)?

Mesh shader: Use a compute-like programming model to feed the rasterizer directly(? No fix function between these two?)

Task shader execution

* 1 invocation ∼ 1 meshlet
* Use EmitMeshTasksEXT to launch mesh workgroups

What can you do in a task shader?

* Coarse per-meshlet culling
* LOD selection
* Geometry amplification
* Replacement for compute pre-pass

Mesh shader execution

* 1 invocation ∼ 1-2 vertex/primitive
* Use SetMeshOutputsEXT to allocate output arrays
* Write to output arrays
* Follow driver preferences!

What else can you do in a mesh shader?

* Per-triangle culling
* Procedural generation of vertices and primitives



Mesh shader driver preferences

* NVidia: small workgroups, more vertices/primitives per invocation
* AMD: large workgroups, 1 vertex/primitive per invocation
* Use a compile-time loop to match both!

On NV:

We recommend using up to 64 vertices and 126 primitives. The ‘6’ in 126 is not a typo. The first generation hardware allocates primitive indices in 128 byte granularity and needs to reserve 4 bytes for the primitive count. Therefore 3 \* 126 + 4 maximizes the fit into a 3 \* 128 = 384 bytes block. Going beyond 126 triangles would allocate the next 128 bytes. 84 and 40 are other maxima that work well for triangles.

# Vertex shader vs mesh shader

### D3D vertex shader

*VS\_OUTPUT main(VS\_INPUT input)*

*{*

***VS\_OUTPUT Output****;*

*float4 pos = float4(input.vPos, 1.0f);*

*// object space to homogeneous projection space*

*pos = mul(pos, mWorld);*

*pos = mul(pos, View);*

*pos = mul(pos, Projection);*

*Output.Position = pos;*

*// Just pass through the color data*

*Output.Color = float4(input.vColor, 1.0f);*

*return Output;*

*}*

### Vulkan vertex shader

*layout (location = 0) in vec3 inPos;*

*layout (location = 1) in vec3 inColor;*

*layout (binding = 0) uniform UBO*

*{*

*mat4 projectionMatrix;*

*mat4 modelMatrix;*

*mat4 viewMatrix;*

*} ubo;*

*layout (location = 0) out vec3 outColor;*

*out gl\_PerVertex*

*{*

*vec4 gl\_Position;*

*};*

*void main()*

*{*

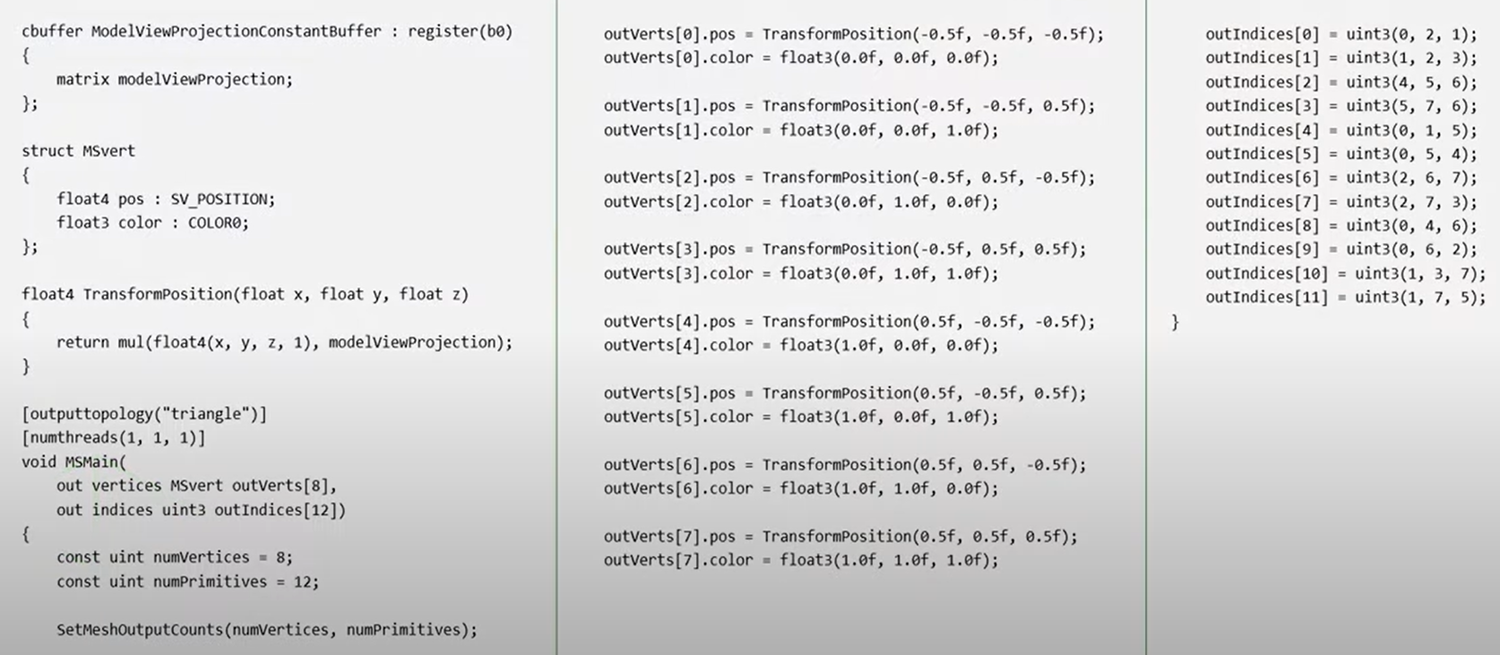
*outColor = inColor;*

*gl\_Position = projectionMatrix \* viewMatrix \* modelMatrix \* vec4(inPos.xyz, 1.0);*

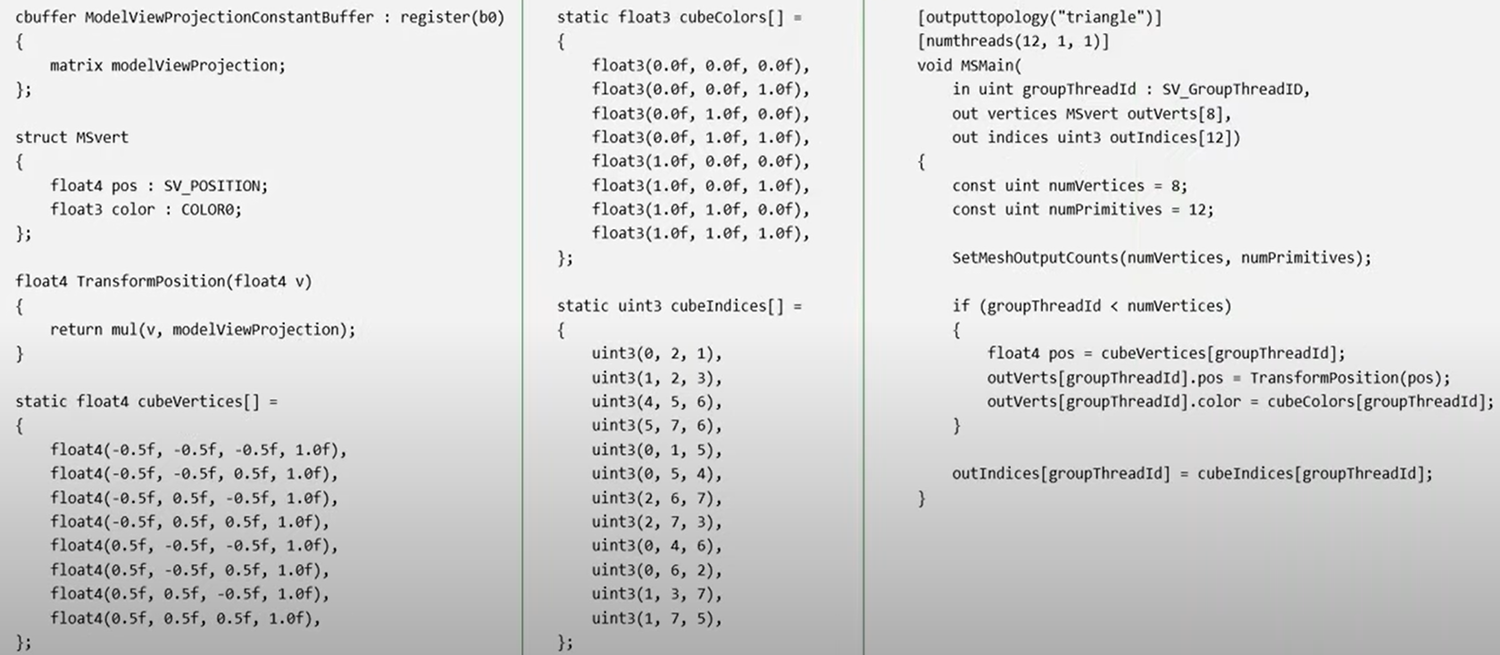
*}*

### First mesh shader(hlsl): cube

(HLSL sample from: https://www.youtube.com/watch?v=CFXKTXtil34)



### Better mesh shader(hlsl): cube



### Mesh shader with meshlet(hlsl)



# Task shader and mesh shader 1

### Only Mesh shader

Case 1: One meshlet per one mesh thread work group (One meshlet per one thread):

```

vkCmdDrawMeshTasksNV(commandBuffer, uint32\_t(mesh.meshlets.size()), 0);

layout(local\_size\_x = 1, local\_size\_y = 1, local\_size\_z = 1) in;

<https://github.com/zeux/niagara/blob/7ad941833f5bd23f19ea667a3b2cc3911520d20b/src/shaders/meshlet.mesh.glsl>

```

Case 2: One meshlet per mesh thread work group (One meshlet per 32 threads):

```

vkCmdDrawMeshTasksNV(commandBuffer, uint32\_t(mesh.meshlets.size()), 0);

layout(local\_size\_x = 32, local\_size\_y = 1, local\_size\_z = 1) in;

<https://github.com/zeux/niagara/blob/2898caaaf14f1c2c96f31eef8409f2d83adff53d/src/shaders/meshlet.mesh.glsl>

```

### Task shader and mesh shader

**Task shader**: One meshlet per task thread

1), Task

* Number of task shader Workgroups:
  + vkCmdDrawMeshTasksNV(commandBuffer, uint32\_t(mesh.meshlets.size()) / 32, 0);
* Work group size:
  + layout(local\_size\_x = 32, local\_size\_y = 1, local\_size\_z = 1) in;

2), Meshlet addressing:

uint ti = gl\_LocalInvocationID.x;

uint mgi = gl\_WorkGroupID.x;

uint mi = mgi \* 32 + ti;

3), Output:

gl\_TaskCountNV (How many meshlet are generated by this task workgroup, number of workgroups for mesh shader.) gl\_TaskCountNV is a work group variable.

<https://github.com/zeux/niagara/blob/fa3d81dfda345520f48accf70da7cb4bcc1b1583/src/shaders/meshlet.task.glsl>

**Mesh shader**: One meshlet per one mesh thread work group

* Number of mesh shader Workgroups:
  + sum of all(gl\_TaskCountNV)?
* Work group size:
  + layout(local\_size\_x = 32, local\_size\_y = 1, local\_size\_z = 1) in;

<https://github.com/zeux/niagara/blob/fa3d81dfda345520f48accf70da7cb4bcc1b1583/src/shaders/meshlet.mesh.glsl>

# Task shader and mesh shader 2

nvmath::uvec4 drawRange;

drawRange.x = di.meshlet.offset;

drawRange.y = di.meshlet.offset + di.meshlet.count - 1;

drawRange.z = 0;

drawRange.w = 0;

# Culling

Conventional culling (based on vertex + fragment) can be performed by:

* software at the level of an entire draw call
* or by hardware on a per-primitive basis only after all the vertices of a primitive have been shaded(after vertex shader, before rasterization),
* but there are no in-between options.

Mesh shader culling can be performed by:

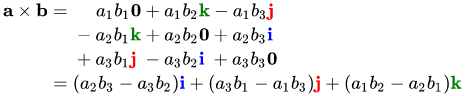
* software at the level of an entire draw call
* a per-primitive basis: back face, frustum, subpixel
* Cluster culling

### Backface culling





https://stackoverflow.com/questions/243945/calculating-a-2d-vectors-cross-product



*// back face culling*

*vec2 ab = b.xy - a.xy;*

*vec2 ac = c.xy - a.xy;*

*float cross\_product = ab.x \* ac.y - ab.y \* ac.x;*

*if (cross\_product \* winding < 0) return false;*

#if USE\_BACKFACECULL

// back face culling

vec2 ab = b.xy - a.xy;

vec2 ac = c.xy - a.xy;

float cross\_product = ab.x \* ac.y - ab.y \* ac.x;

#if IS\_VULKAN

// Vulkan's upper-left window origin means that screen coordinates

// are reversed relative to OpenGL's. Reverse the sign of the

// cross-product to compensate.

cross\_product = -cross\_product;

#endif

if (cross\_product \* winding < 0) return false;

#endif

#if USE\_VIEWPORTCULL || USE\_SUBPIXELCULL

// compute the min and max in each X and Y direction

vec2 pixelMin = min(a,min(b,c));

vec2 pixelMax = max(a,max(b,c));

#endif

#if USE\_VIEWPORTCULL

// viewport culling

if (frustum && ((pixelMax.x < 0) || (pixelMin.x >= scene.viewportf.x) || (pixelMax.y < 0) || (pixelMin.y >= scene.viewportf.y))) return false;

#endif

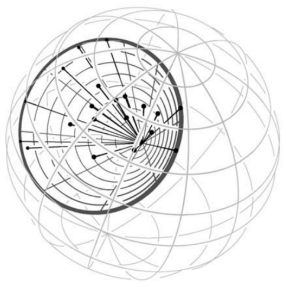
#if USE\_SUBPIXELCULL

if (pixelBboxCull(pixelMin, pixelMax)) return false;

#endif

### Cluster culling

The idea is “Cull entire clusters which are completely back-facing”.

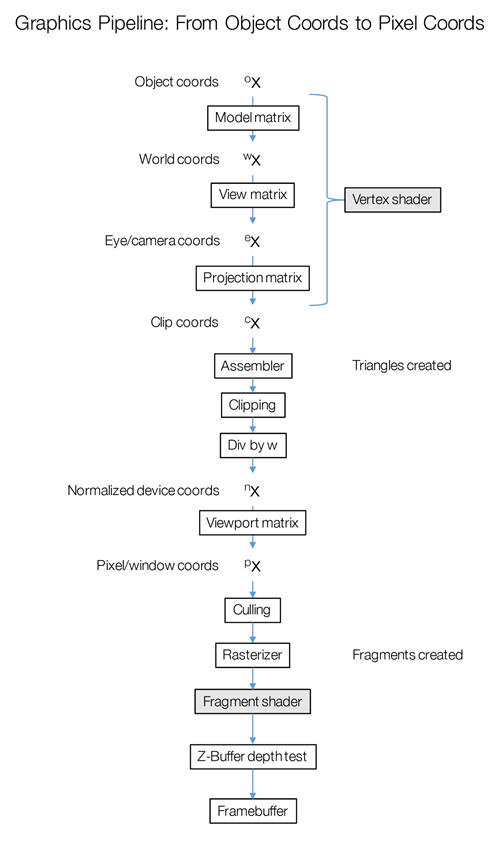


<https://gpuopen.com/learn/geometryfx-1-2-cluster-culling/>

<http://advances.realtimerendering.com/s2015/aaltonenhaar_siggraph2015_combined_final_footer_220dpi.pdf>

https://cgvr.cs.uni-bremen.de/teaching/cg\_literatur/clustered\_backface\_culling.pdf

https://gitea.yiem.net/QianMo/Real-Time-Rendering-4th-Bibliography-Collection/raw/branch/main/Chapter%201-24/[1630]%C2%A0[Computer%20Graphics%20Forum%201993]%20The%20Cone%20of%20Normals%20Technique%20for%20Fast%20Processing%20of%20Curved%20Patches.pdf



Meshlet culling:

https://www.youtube.com/watch?v=KckRq7Rm3Mw

Cluster culling:

http://advances.realtimerendering.com/s2015/aaltonenhaar\_siggraph2015\_combined\_final\_footer\_220dpi.pdf

occlusion culling

Cone culling:

https://github.com/zeux/niagara/blob/fa3d81dfda345520f48accf70da7cb4bcc1b1583/src/shaders/meshlet.task.glsl

Meshlet culling

https://computergraphics.stackexchange.com/questions/11060/what-are-mesh-clusters-hierarchical-cluster-culling-with-lod-triangle-c

### Stats

if (laneID == 0) {

gl\_PrimitiveCountNV = primCount;

#if USE\_STATS

atomicAdd(stats.meshletsOutput, 1);

atomicAdd(stats.trisOutput, primCount);

atomicAdd(stats.attrInput, vertCount);

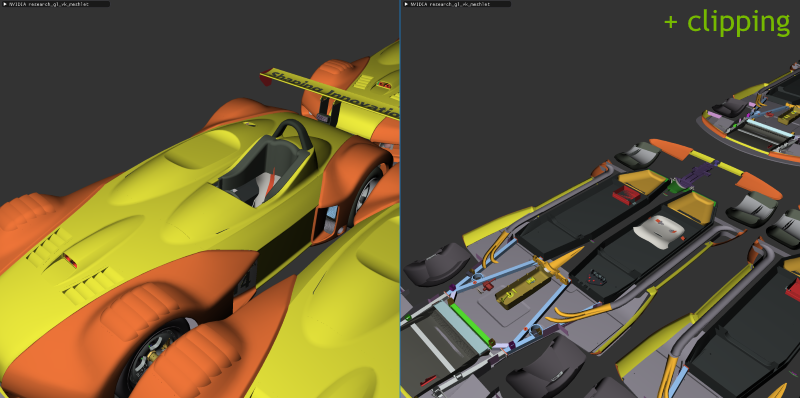
atomicAdd(stats.attrOutput, vertCount);

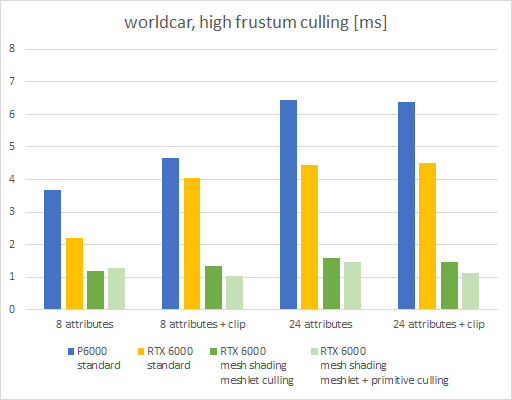
#endif

}

### Perf data(2018)

9 cars, 32 M triangles and 16 K drawcalls.





| **timing in [ms]** | **P6000 std** | **RTX 6000 std** | **RTX 6000 mesh T** | **RTX 6000 mesh TM** |
| --- | --- | --- | --- | --- |
| 8 attributes | 3.68 | 2.19 | 1.20 | 1.29 |
| 8 attributes + clipping | 4.65 | 4.04 | 1.34 | 1.02 |
| 24 attributes | 6.46 | 4.44 | 1.60 | 1.47 |
| 24 attributes + clipping | 6.38 | 4.50 | 1.46 | 1.12 |
| **triangle output** |  |  |  |  |
| regular | 100 % | 100 % | 31 % | 5 % |
| clipping | 100 % | 100 % | 20 % | 2 % |

From: https://github.com/nvpro-samples/gl\_vk\_meshlet\_cadscene

# Mesh shaing Possiblity

<https://www.reedbeta.com/blog/mesh-shader-possibilities/>

# Loop unrolling(Compile time loop)

CPU: http://spraetor.github.io/2015/12/26/compile-time-loops.html

GPU:

https://etd.ohiolink.edu/apexprod/rws\_etd/send\_file/send?accession=osu1253131903&disposition=inline

### 