NVIDIA VULKAN UPDATE

Christoph Kubisch, March 20 2019, GTC 2019

AGENDA

Turing Mesh Shaders

Turing Barycentrics

Buffer Reference

Turing Cooperative Matrix

Partitioned Subgroup

Turing Texture Access Footprint

Turing Derivatives in Compute Shader

Turing Corner Sampled Image

Turing Representative Fragment Test

Turing Exclusive Scissor Test

Cross API Interop

DEDICATED SESSIONS

GTC 2019

S9833 - NVIDIA VKRay - Ray Tracing in Vulkan

Hardware-Accelerated Real-time Raytracing

VK_NV_ray_tracing

S9891 - Updates on Professional VR and Turing VRWorks

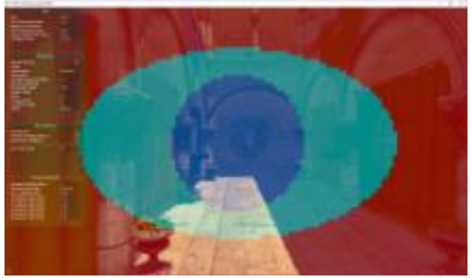
Variable rate shading, multi-view, multi-GPU

VK_NV_shading_rate_image,

KHR_multiview and KHR_device_group
(promoted in VK 1.1)

S9661 - NVIDIA Nsight Graphics: Getting The Most From Your Vulkan Applications

Profiling and Debugging



MESH SHADERS

MOTIVATION

Detail Geometry

- Vegetation, undergrowth, greebles
- Fine geometric detail at massive scale
- Pre-computed topologies for LODs
- Efficient submission of small objects
- Flexible instancing
- Custom precision for vertices

MOTIVATION

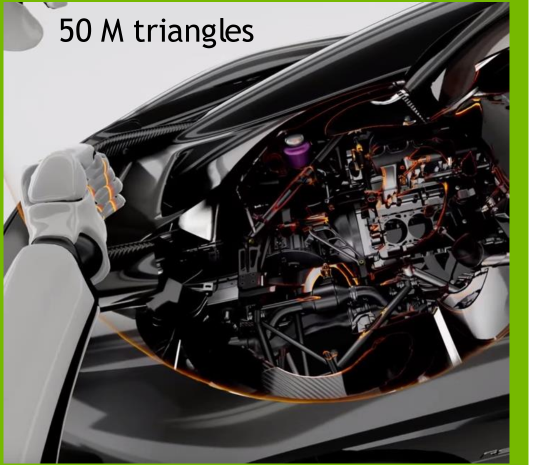
Auxiliary Meshes

- Proxy hull objects
- Iso-surface extraction
- Particles
- Text glyphs
- Lines/Stippling etc.
- Instancing of procedural shapes

MOTIVATION

CAD Models

- High geometric complexity
  (treat as many simple triangle clusters)
- Large assemblies can easily reach
  multiple 100 million triangles
- VR demands high framerates and detail
- Cannot always rely on static solutions
  (animations, clipping etc.)
- Allow compressed representations



MESH SHADING

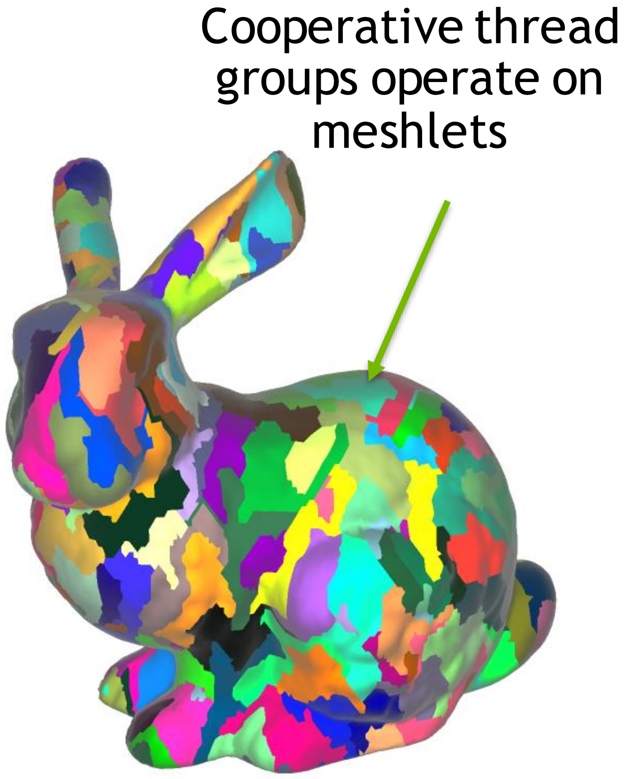
New programming model for geometry processing

Evolution from singleton shaders to cooperative groups

- Pixel lighting ➔ Tile-based lighting via compute
- Vertex processing ➔ Meshlet processing

Essential components

- Compute-like execution model – data sharing and sync
- No fixed-function fetch for index processing or vertices
- One level of expansion, flexible work creation/tessellation



EXECUTION

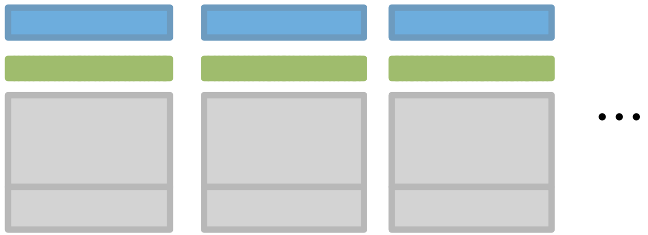
Compute Shader Model

SHADER INVOCATIONS

Input: uint WorkGroupID

Thread group

Dispatched as 1D grid



Output memory <=16 KB

Compile-time allocation size

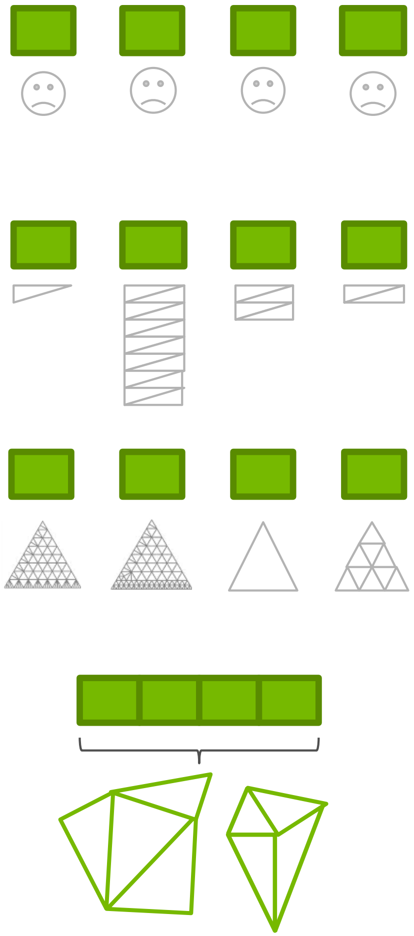
Generic Output or Vertices/Indices

Shared Memory

Cooperative access to per-workgroup memory

Manual synchronization required (barrier()...)

Shader		Thread Mapping	Topology
Vertex Shader	No access to connectivity	1 Vertex	No influence
Geometry Shader	Variable output doesn‘t fit HW well	1 Primitive / 1 Output Strip	Triangle Strips
Tessellation Shader	Fixed-function topology	1 Patch / 1 Evaluated Vertex	Fast Patterns
Mesh Shader	Compute shader features	Flexible	Flexible within work group allocation



MESH SHADING

New Geometric Pipeline

TRADITIONAL Vertex/Tessellation/Geometry (VTG) PIPELINE

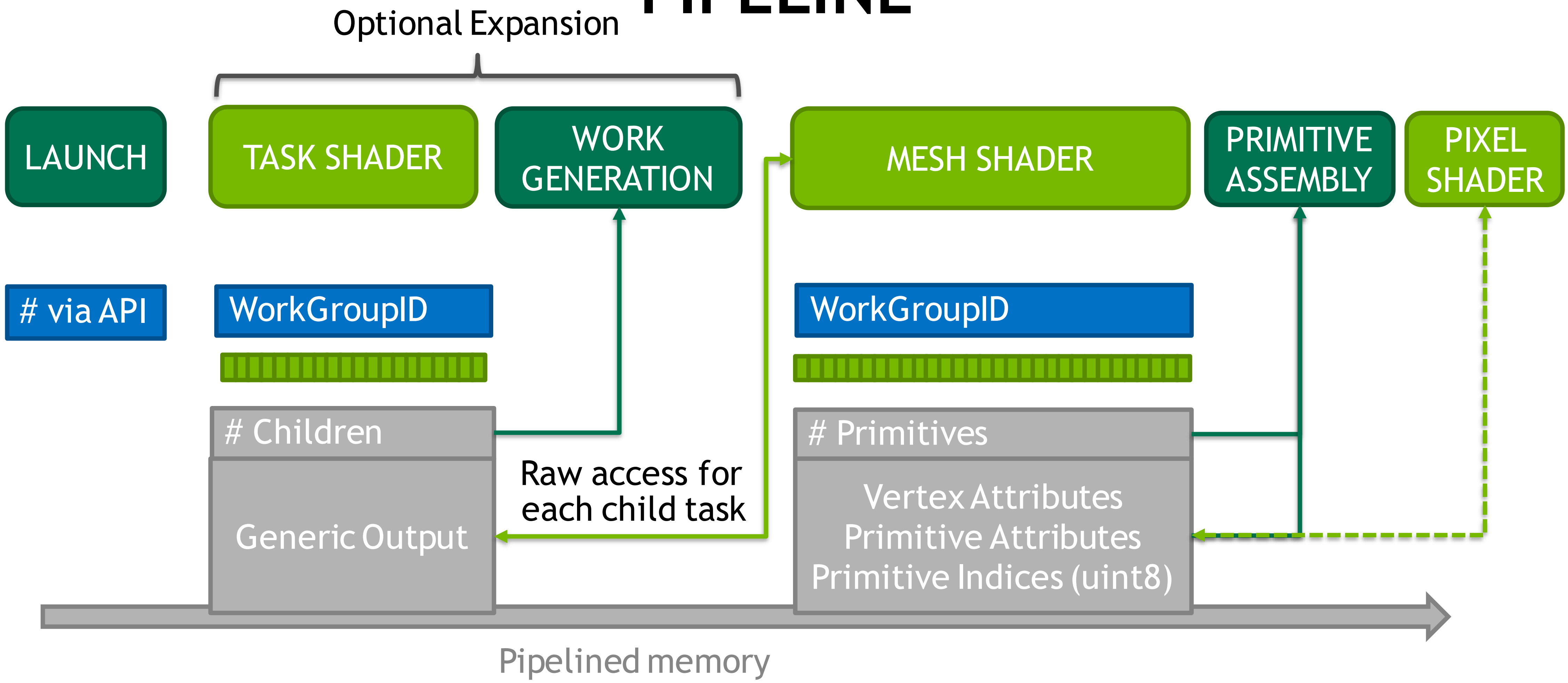
Pipelined memory, keeping interstage data on chip

TASK/MESH PIPELINE

Optional Expansion

Pipelined memory…

PIPELINE


Optional Expansion
LAUNCH
TASK SHADER
WORK GENERATION
MESH SHADER
PRIMITIVE ASSEMBLY
PIXEL SHADER
via API
WorkGroupID
WorkGroupID
Children
Generic Output
Raw access for each child task
Primitives
Vertex Attributes
Primitive Attributes
Primitive Indices (uint8)
Pipelined memory

TASK & MESH SHADING

Task shader allows culling (subpixel, frustum, back-face, occlusion...) or
lod picking to minimize mesh workgroups

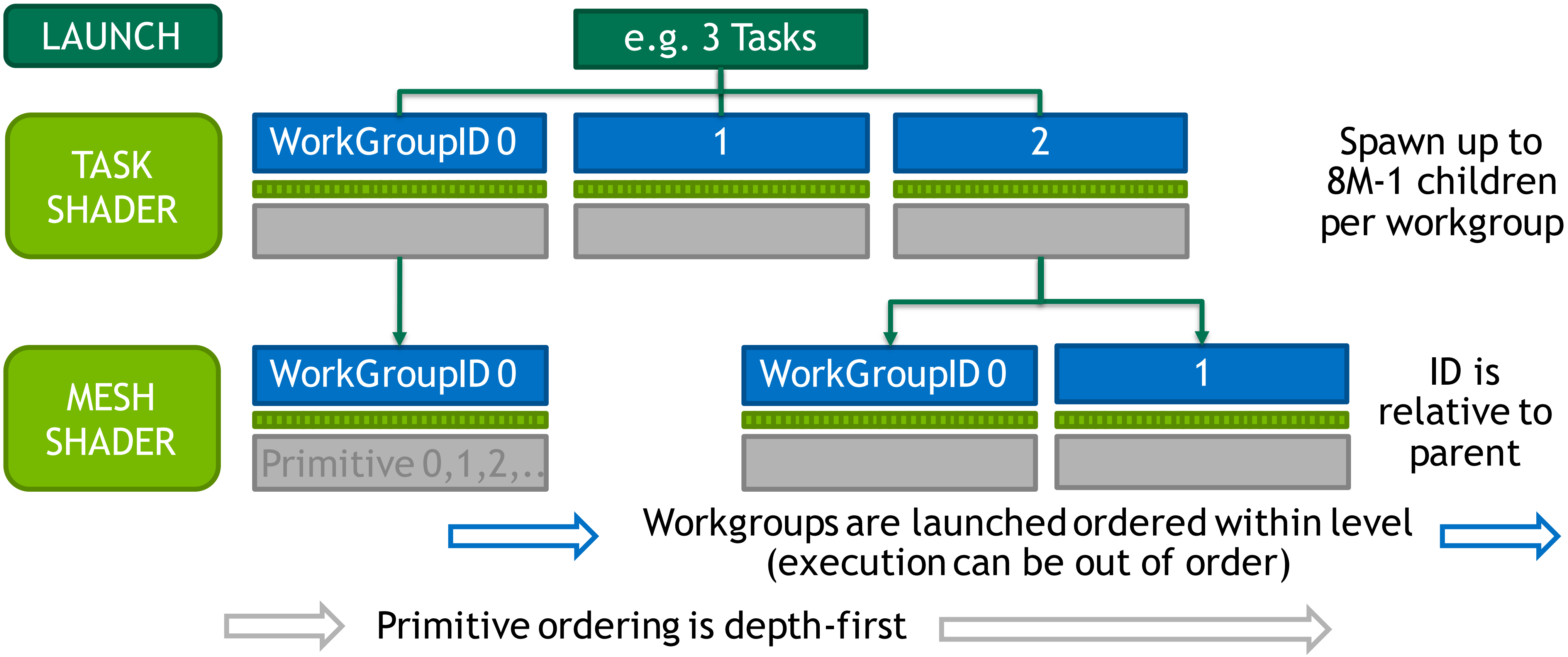
For generic use we recommend meshlets with 64 vertices, 84 or 124 triangles

Use your own encodings for geometry, all data fetched by shader (compression etc.)

Provides more efficient procedural geometry creation (points, lines, triangles)

With disabled rasterizer implement basic compute trees

TREE EXPANSION



API

GL & VK & SPIR-V EXTENSIONS

Introduces new graphics stages (TASK, MESH) that cannot be combined with VTG stages

New drawcalls operate only with appropriate pipeline (similar calls in GL)

```
void vkCmdDrawMeshTasksNV(VkCommandBuffer buffer, uint32_t taskCount, uint32_t taskFirst);
     vkCmdDrawMeshTasksIndirectNV
     vkCmdDrawMeshTasksIndirectCountNV
```

GLSL

```
// same as compute
layout(local_size_x=32) in;
in uvec3 gl_WorkGroupID;
in uvec3 gl_LocalInvocationID;
...
shared MyStruct s_shared;

// new for task shader
out uint gl_TaskCountNV;

// new for mesh shader
layout(max_vertices=64) out;
layout(max_primitives=84) out;
layout(triangles/lines/points)
out;

out uint gl_PrimitivesCountNV;
out uint gl_PrimitiveIndicesNV[];
```

```
out gl_MeshPerVertex {
 vec4  gl_Position;
 float gl_PointSize;
 float gl_ClipDistance[];
 float gl_CullDistance[];
} gl_MeshVerticesNV[]; // [max_vertices]

perprimitiveNV out gl_MeshPerPrimitive {
  int gl_PrimitiveID;
  int gl_Layer;
  int gl_ViewportIndex
  int gl_ViewportMask;
} gl_MeshPrimitivesNV[]; // [max_primitives]

taskNV in/out MyCustomTaskData {
 ...
} blah;
```

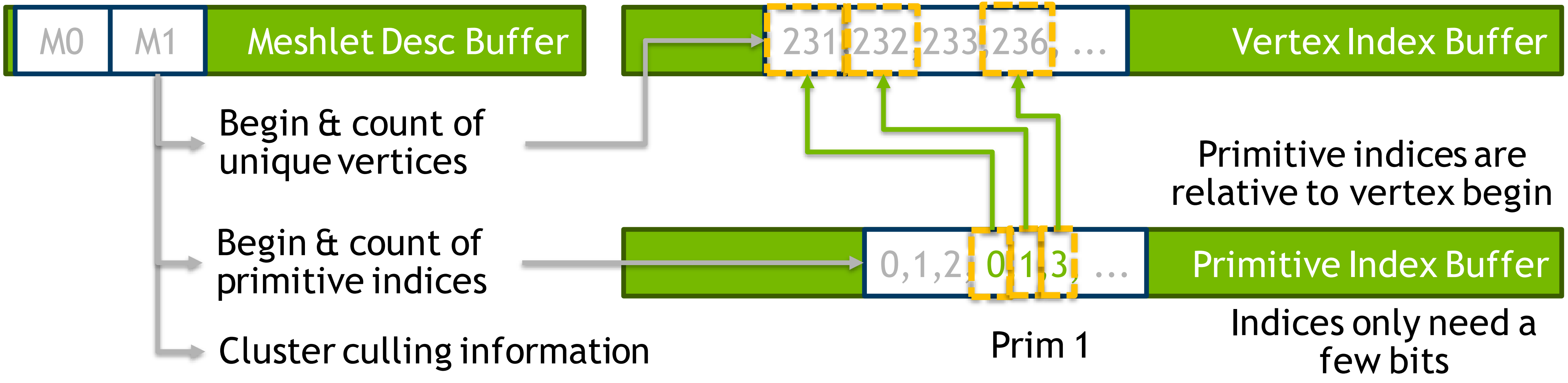
```
layout(local_size_x=32) in;
layout(max_vertices=32, max_primitives=32, triangles) out;
out MyVertex {      // define custom per-vertex as usual
  vec3 normal;      // interfaces with fragment shader
} myout[];

void main() {
  uint invocation = gl_LocalInvocationID.x;
  uvec4 info = meshinfos[gl_WorkGroupID.x]; // #verts,vertoffset,#prims,primoffset

  uint vertex = min(invocation, info.x - 1);
  gl_MeshVerticesNV[invocation].gl_Position = texelFetch(texVbo, info.y + vertex);
  myout[invocation].normal = texelFetch(texNormal, info.y + vertex).xyz;

  uint prim = min(invocation, info.z - 1);
  uint topology = texelFetch(texTopology, info.w + prim);
  // alternative utility function exists to write packed 4x8
  gl_PrimitiveIndicesNV[invocation * 3 + 0] = (topology<<0) & 0xFF;
  gl_PrimitiveIndicesNV[invocation * 3 + 1] = (topology<<8) & 0xFF;
  gl_PrimitiveIndicesNV[invocation * 3 + 2] = (topology<<16) & 0xFF;
  gl_PrimitiveCountNV = info.z;              // (actually one thread enough)
}
```

MESHLET EXAMPLE

Data Structure

Replace traditional indexbuffer with pre-computed custom packing

Pack meshlets against a fixed vertex/primitive limit

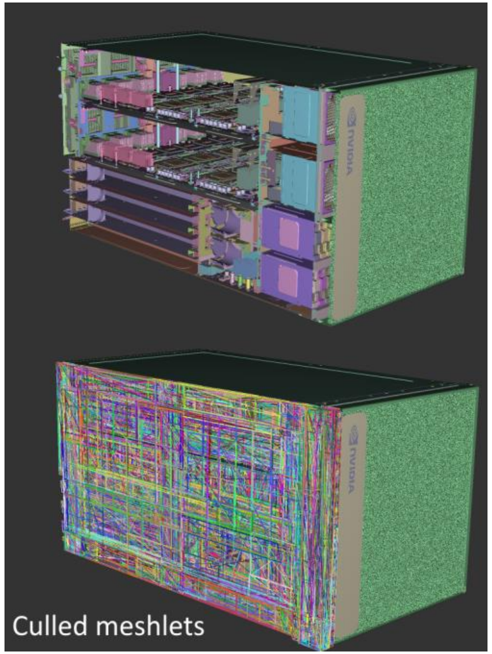


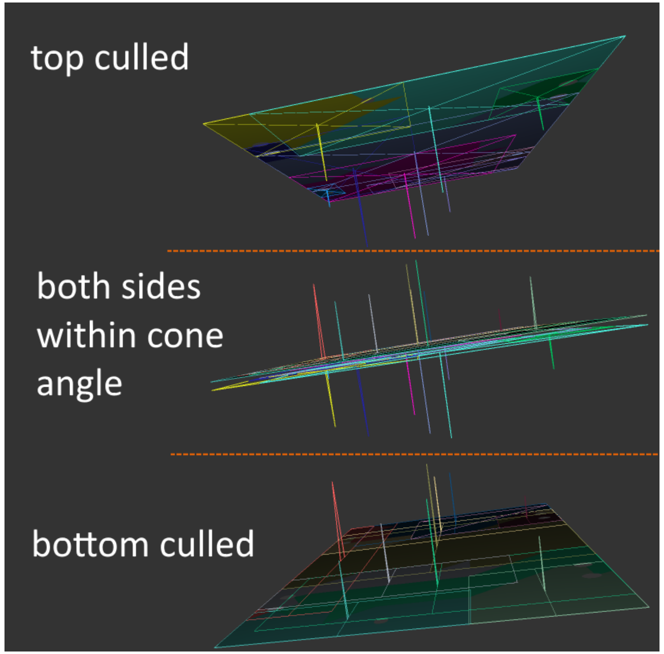
MESHLET EXAMPLE

Cluster Culling

Task shader handles cluster culling:

- Outside frustum
- User clipping plane
- Back-face cluster
- Below custom pixel size



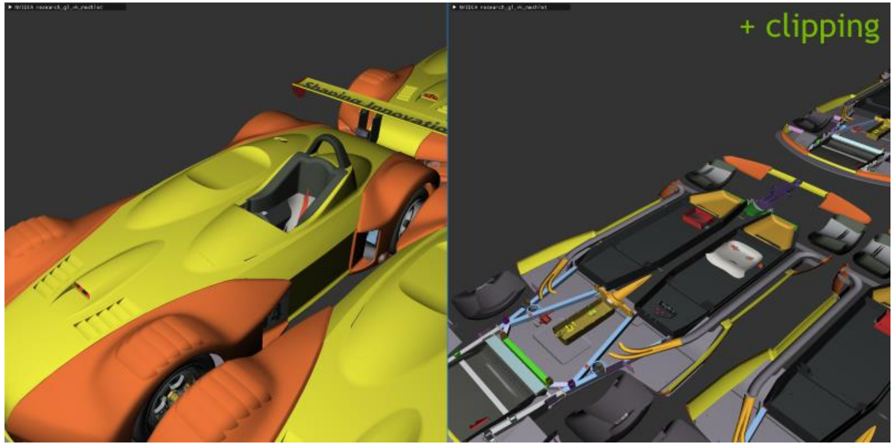


MESHLET EXAMPLE

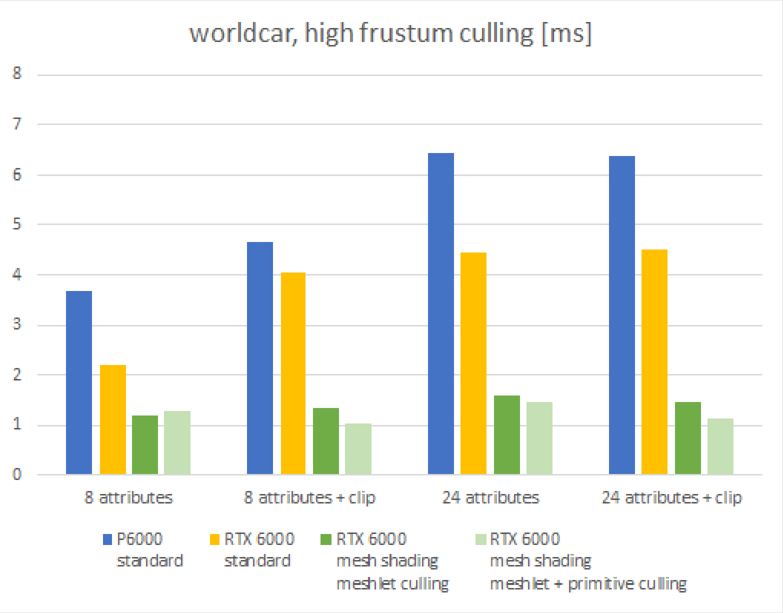
Open-Source Sample

Sample that replaces indexbuffer with meshlet data structure and uses task shader to perform cluster culling. It also saves 25-50% of memory compared to indexbuffer.

https://github.com/nvpro-samples/gl_vk_meshlet_cadscene



model courtesy of PTC

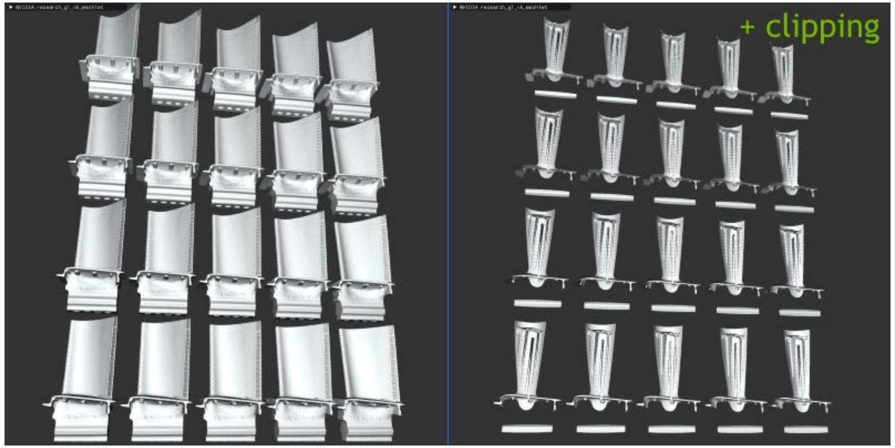


MESHLET EXAMPLE

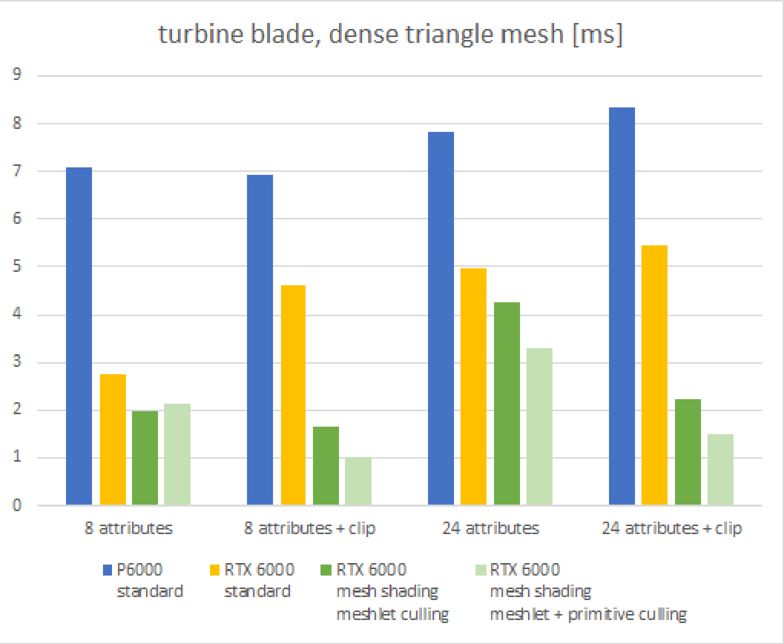
Open-Source Sample

Sample that replaces indexbuffer with meshlet data structure and uses task shader to perform cluster culling. It also saves 25-50% of memory compared to indexbuffer.

https://github.com/nvpro-samples/gl_vk_meshlet_cadscene



model courtesy of Georgia Institute of Technology



TINY DRAW CALLS

Some scenes suffer from low-complexity drawcalls (< 512 triangles)

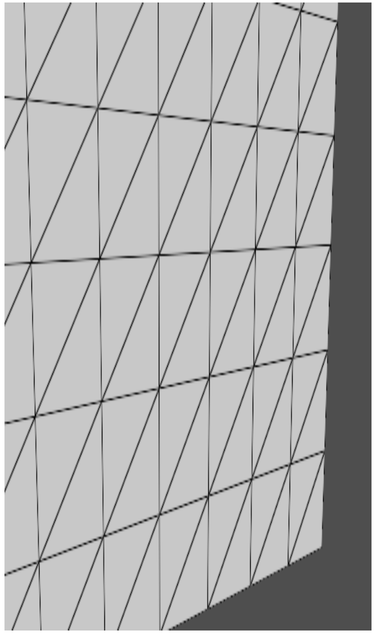
Task shaders can serve as faster alternative to Multi Draw Indirect (MDI)

- MDI or instanced drawing can still be bottlenecked by GPU
- Task shaders provide distributed draw call generation across chip
- Also more flexible than classic instancing (change LOD etc.)

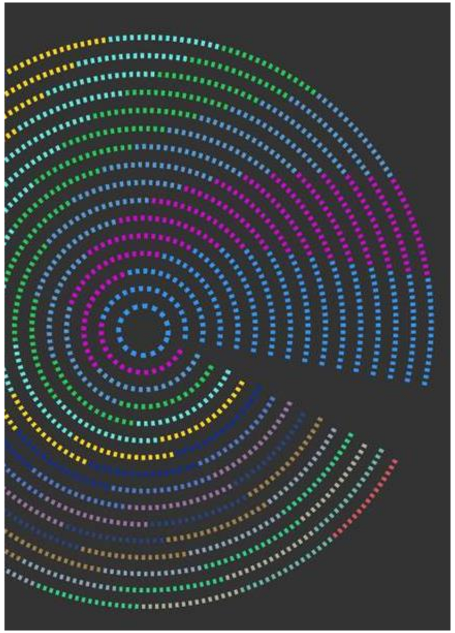
PROCEDURAL MESHES

Task shader can compute how much work needs to be generated per input primitive (line strips [4], grids, shapes etc.).

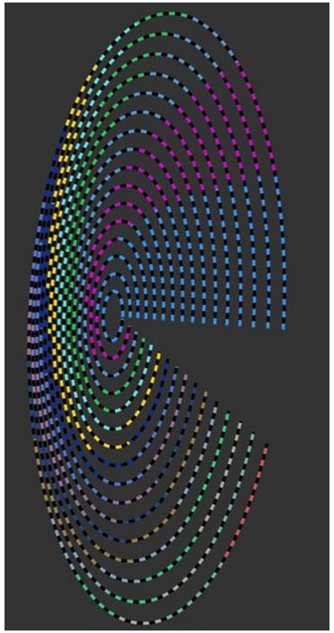
Can also skip invisible portions entirely.



Procedural Grid



Geometry stipples



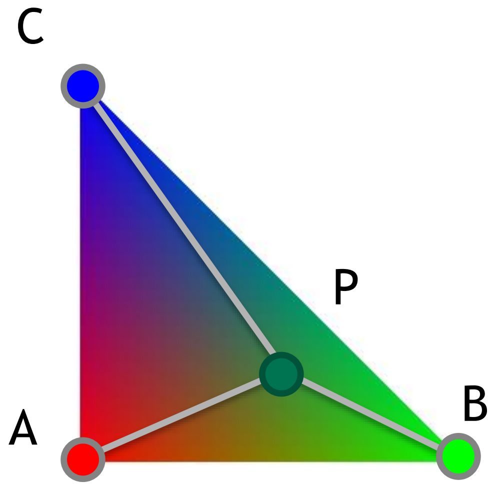
Texture space stipples

BARYCENTRICS

BARYCENTRIC COORDINATES

VK/SPV_NV_fragment_shader_barycentric

Custom interpolation of fragment shader inputs



P = A*bx + B*by + C*bz

```
// new built-ins
in vec3 gl_BaryCoordNV;
in vec3 gl_BaryCoordNoPerspNV;

// new keyword to get un-interpolated inputs
pervertexNV in Inputs {
  uint  packed;
} inputs[];

// manual interpolation, also allows using smaller datatypes

vec2 tc = unpackHalf2x16(inputs[0].packed) * gl_BaryCoordNV.x +
          unpackHalf2x16(inputs[1].packed) * gl_BaryCoordNV.y +
          unpackHalf2x16(inputs[2].packed) * gl_BaryCoordNV.z;
```

BUFFER REFERENCE

BUFFER REFERENCE

GLSL_EXT_buffer_reference

Greater flexibility in custom data structures stored within SSBOs

„pointer“-like workflow

Developer responsible to manage alignment

```
// declare a reference data type
layout(buffer_reference, buffer_reference_align=16) buffer MyType {
  uvec2 blah;
   vec2 blubb;
};
uniform Ubo {
  MyType ref; // buffer references are 64-bit sized, address via API
};

// behaves similar to struct, can also be passed to functions
... ref.blah ... or ... doSomething(ref);

// flexible casting, and constructing from other references/uint64
... MyType(uint64_t(ref) + 128).foo ... MyOtherType(ref).foo

// UPCOMING EXTENSION: array/arithmetic usage
... (ref+1).blah ... or ... doSomething(ref + idx);
...  ref[1].blah ... or ... doSomething(ref[idx]);
```

BUFFER REFERENCE

VK_EXT_buffer_device_address

Ability to get the physical address of buffers

The extension was also designed to be debug tool friendly (nsight, renderdoc etc.) to allow trace replay with old address values

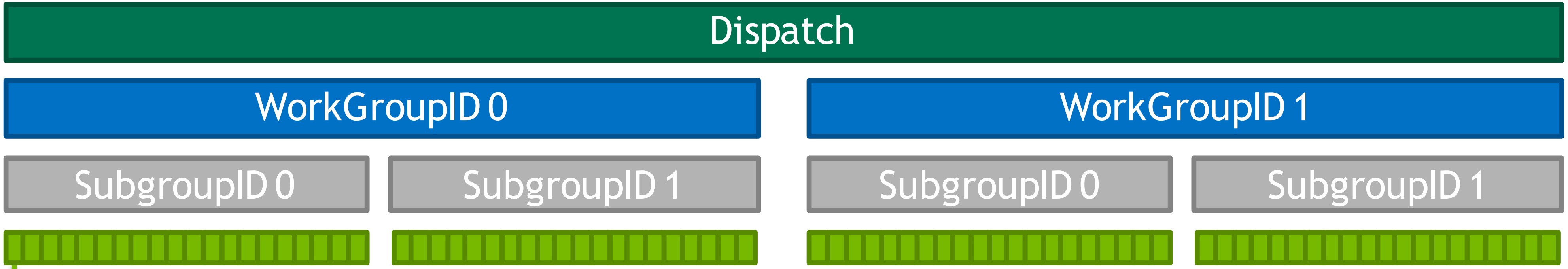
```
// supported on all NVIDIA Vulkan devices

// at creation time enable the new usage
VkBufferCreateInfo info =  {...};
info.usage |= VK_BUFFER_USAGE_SHADER_DEVICE_ADDRESS_BIT_EXT;

// later query the address and use it as value
// within buffers or pushconstants

VkDeviceAddress addr = vkGetBufferDeviceAddressEXT(
   device, {... buffer ...});

// put addr into buffer/image etc. as seen in UBO variable before
```

SUBGROUP REFRESHER

SUBGROUPS

VK_KHR_shader_subgroup_*

Invocations within a subgroup can synchronize and share data with each other efficiently. For NVIDIA 32 invocations form one subgroup (“warp”).




```
// Single Invocation : „Shader Thread“
gl_LocalInvocationID (1D) == gl_SubGroupInvocationID + gl_SubgroupID * gl_SubgroupSize;
```

SUBGROUPS

Task Shader Example

Variable	Invoc. 0	1	2
render	true	false	true
vote	101 (binary)	101	101

```
// meshletID is different per invocation
bool render = valid && !(earlyCull(meshletID, object));

// The ballot functions can be used to easily count across
// a subgroup and create prefixsums
uvec4 vote     = subgroupBallot(render);
uint  tasks    = subgroupBallotBitCount(vote);
// exclusive means the value of current invocation is excluded
uint  outIndex = subgroupBallotExclusiveBitCount(vote);

if (render) {
  OUT.meshletIDs[outIndex] = meshletID;
}
if (gl_SubgroupInvocationID == 0) {
  gl_TaskCountNV = tasks;
}
```

A task shader culls 32 meshlets within a subgroup and outputs surviving meshletIDs

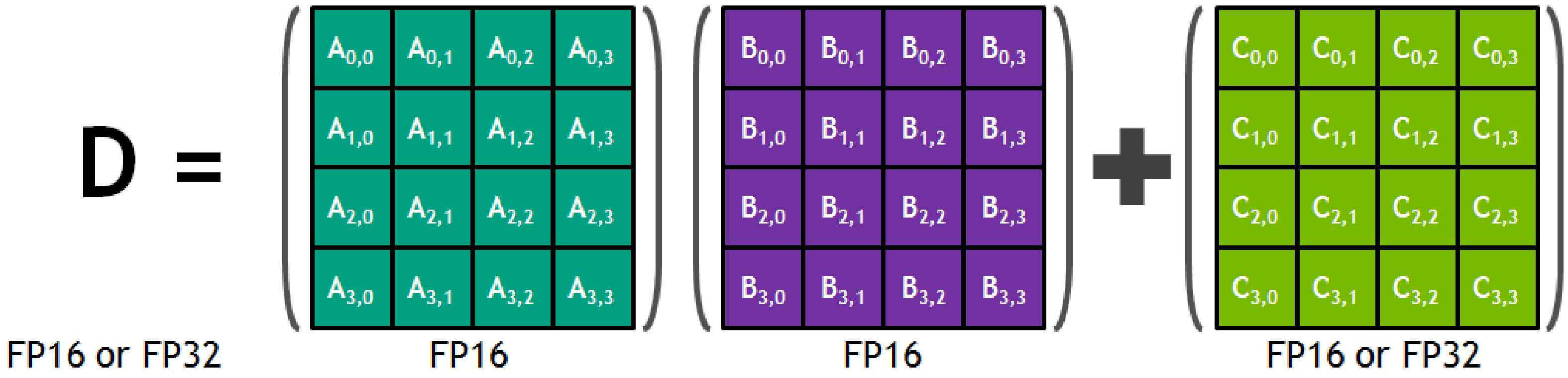
COOPERATIVE MATRIX

COOPERATIVE MATRIX

Tensor Core Access

VK_NV_cooperative_matrix brings very fast large matrix multiply-add to Vulkan

Supported for Turing RTX (NOT Volta)



$$D = A \times B + C$$

COOPERATIVE MATRIX

GLSL cooperative operations

```
// Classic datatype variables exist per invocation (thread) or are in shared memory.
// New datatype introduced that exists within a pre-defined scope.

fcoopmatNV<PRECISION_BITS, gl_ScopeSubgroup, ROWS, COLS> variable;

// new functions handle load/store (one example shown)
void coopMatLoadNV(out fcoopmatNV m,
  volatile coherent float16_t[] buf,      // ssbo or shared memory array variable
                           uint element,  // starting index into buf to load from
                           uint stride,   // element stride for one column or row
                           bool colMajor) // compile-time constant
// if colMajor == true, load COLS many values from buf[element + column_idx * stride];

// perform the actual multiply within the scope (here subgroup)
fcoopmatNV coopMatMulAddNV(fcoopmatNV A, fcoopmatNV B, fcoopmatNV C)
```

COOPERATIVE MATRIX

Integration

Query support from device

Optionally use specialization constants to quickly build multiple kernels

Example here https://github.com/jeffbolznv/vk_cooperative_matrix_perf

71 TFLOPS on Titan RTX

```
typedef struct VkCooperativeMatrixPropertiesNV {
    VkStructureType      sType;
    void*                pNext;
    uint32_t             MSize;
    uint32_t             NSize;
    uint32_t             KSize;
    VkComponentTypeNV    AType;
    VkComponentTypeNV    BType;
    VkComponentTypeNV    CType;
    VkComponentTypeNV    DType;
    VkScopeNV            scope;
} VkCooperativeMatrixPropertiesNV;

// Multiple configurations may be supported

vkGetPhysicalDeviceCooperativeMatrixPropertiesNV
  (VkPhysicalDevice, uint32_t* propCount, ...props)
```

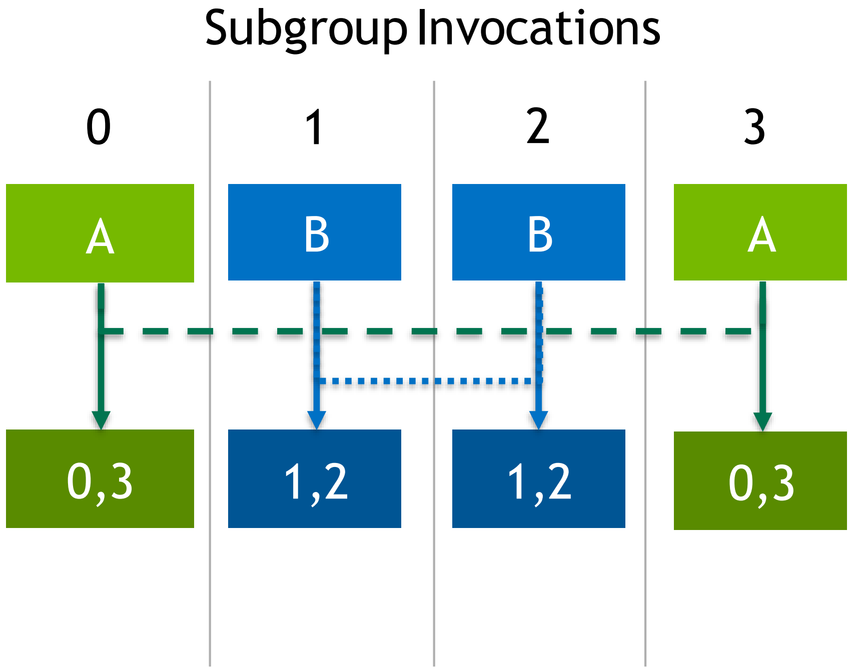
PARTITIONED SUBGROUP

PARTITIONED SUBGROUP

VK_NV_shader_subgroup_partitioned

Identify invocations with the same variable value

Use bitfield masks to operate across subset of threads



PARTITIONED SUBGROUP

VK_NV_shader_subgroup_partitioned

```
// Find invocations with identical key values within a subgroup
uvec4 identicalBitMask = subgroupPartitionNV(key);
```

Value	Invocation 0	1	2	3	4
key	17	35	17	9	35
identicalBitMask	00101 (binary)	10010	00101	01000	10010

```
bool     isFirstUnique = gl_SubgroupInvocationID == subgroupBallotFindLSB(identicalBitMask);
```

isFirstUnique	true	true	false	true	false

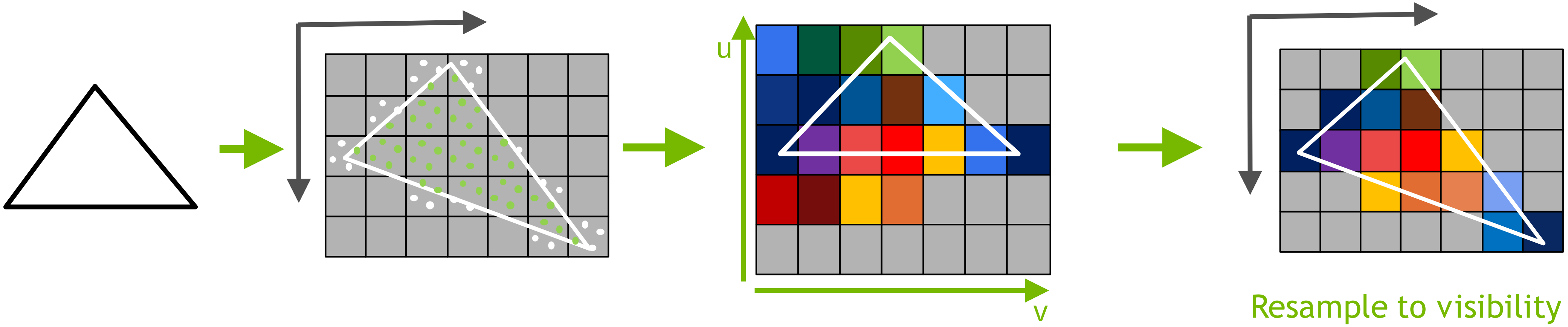
```
// use mask for aggregate operations, for example
uint sum = subgroupPartitionedAddNV(value, identicalBitMask);
```

value	7	3	13	1	2
sum	20	5	20	1	5

TEXTURE ACCESS FOOTPRINT

TEXTURE SPACE SHADING

Aka Decoupled Shading

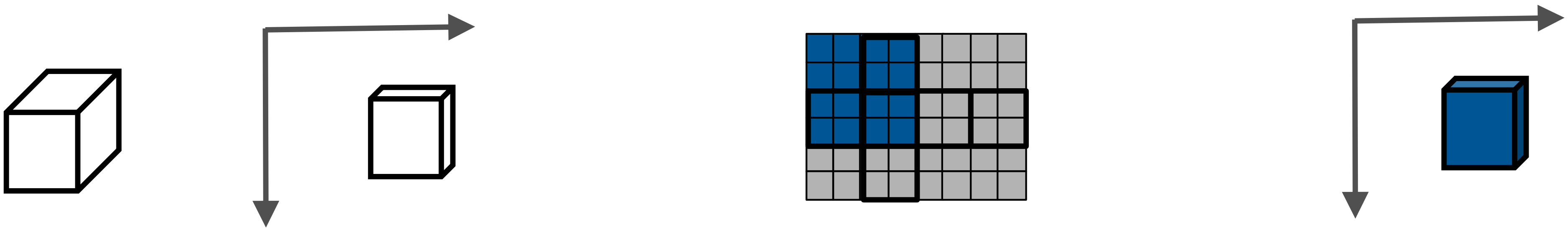


Geometry

Visibility sampling
(find texels that are needed)

Shading
(shade texels in uv space)

Resample to visibility
(regular texture fetch, anti-aliasing via texture filtering)



TEXTURE SPACE SHADING

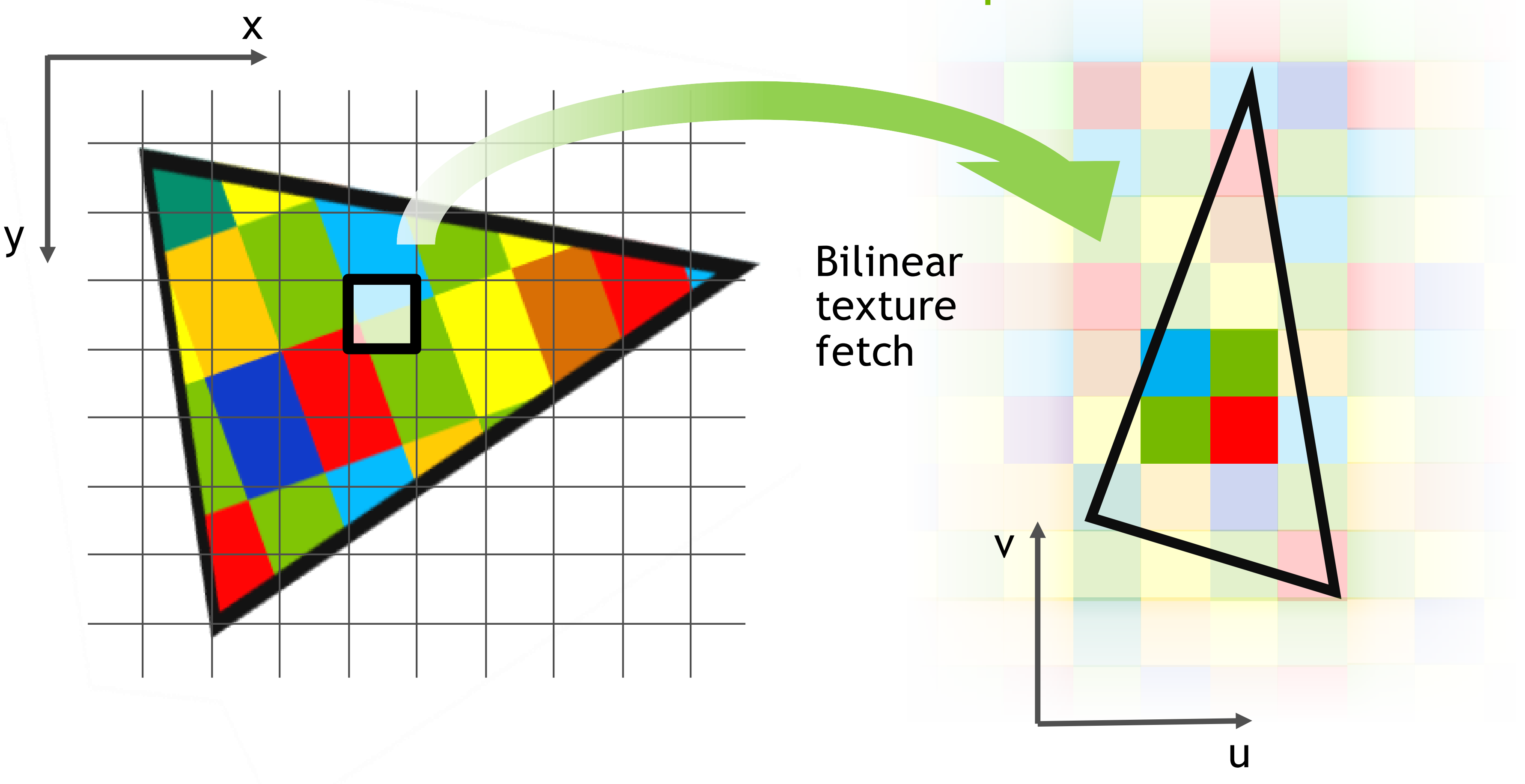
https://devblogs.nvidia.com/texture-space-shading/

https://www.youtube.com/watch?v=Rpy0-q0TyB0

Visit these links for more details

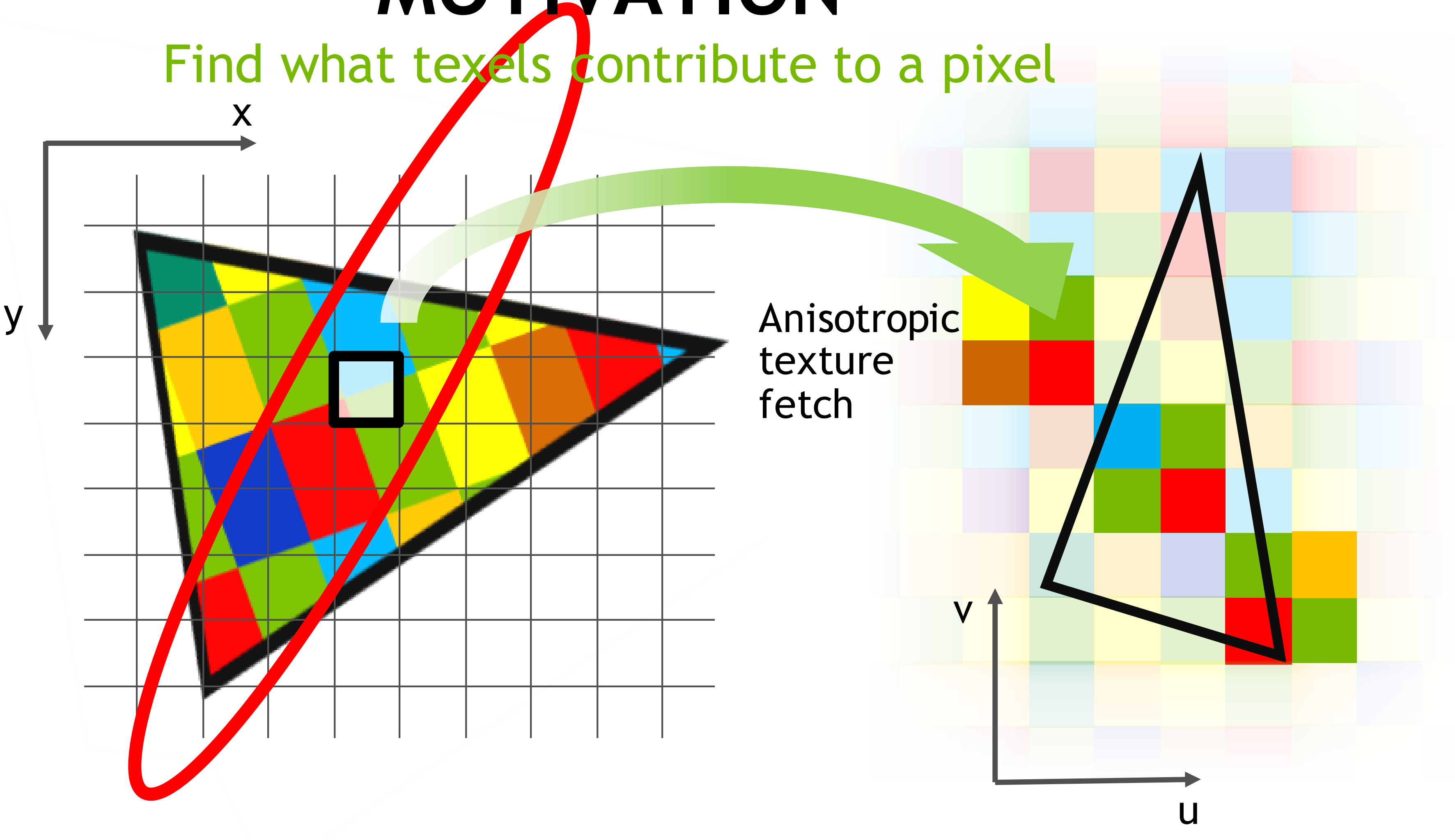
MOTIVATION

Find what texels contribute to a pixel



MOTIVATION

Find what texels contribute to a pixel



TEXTURE ACCESS FOOTPRINT

VK_NV_shader_image_footprint / GLSL_NV_shader_texture_footprint

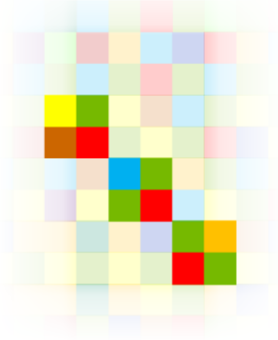
New query functions in GLSL/SPIR-V

Returned footprint helps to identify which mips and which texel tiles within them would be touched

```
gl_TextureFootprint2DNV {
  uvec2 anchor;
  uvec2 offset;
  uvec2 mask;
  uint  lod;
  uint  granularity;
} footprint;

bool singleMipOnly =
  textureFootprintNV(
    tex, uv,
    granularity,
    bCoarseMipLevel,
    footprint);
```

Each bit in mask represents tiles: e.g. 2x2 texels

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0



DERIVATIVES IN COMPUTE SHADERS

DERIVATIVES IN COMPUTE

VK_NV_compute_shader_derivatives

Previously only fragment shader texture lookups allowed the use of derivatives in texture lookups (implicit mip-mapping etc.)

Now compute shaders supports:

- All texture functions
- Derivative functions
- subgroup_quads functions

Local invocations as 2x2x1 (quads)

2x+0, 2y+0, z	2x+1, 2y+0, z
2x+0, 2y+1, z	2x+1, 2y+1, z

as linear threads

4n+0	4n+1
4n+2	4n+3

```
// enable the layout
layout(derivative_group_quadsNV) in;
// or
layout(derivative_group_linearNV) in;

// you can use all texture functions now
... texture(tex, uv);
// or derivatives
... dFdx(variable);
```

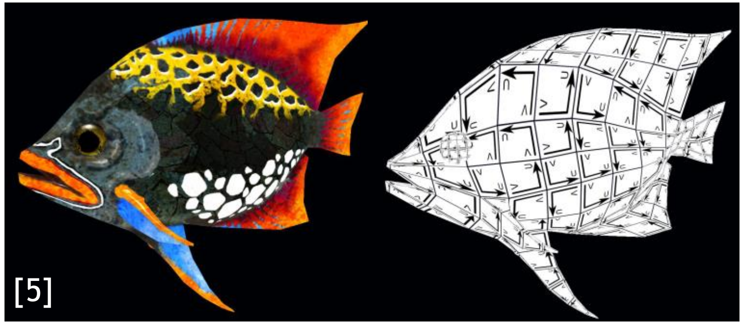
CORNER SAMPLED IMAGE

CORNER SAMPLED IMAGES

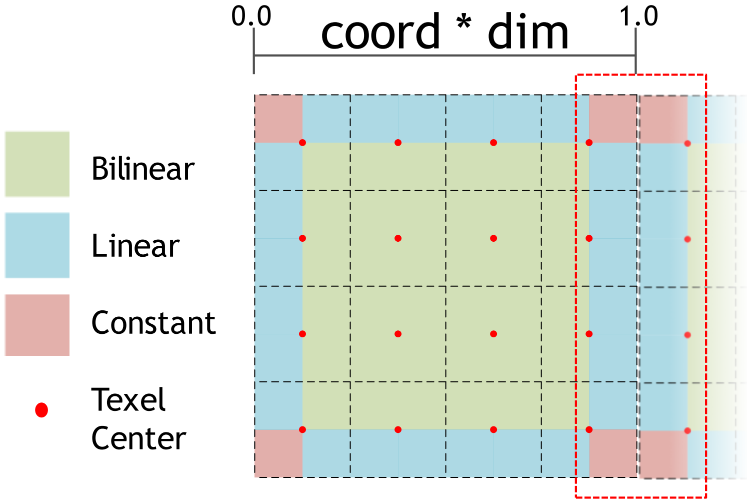
VK_NV_corner_sampled_image

A new extension that eases hardware-accellerated PTEX

No seams at borders

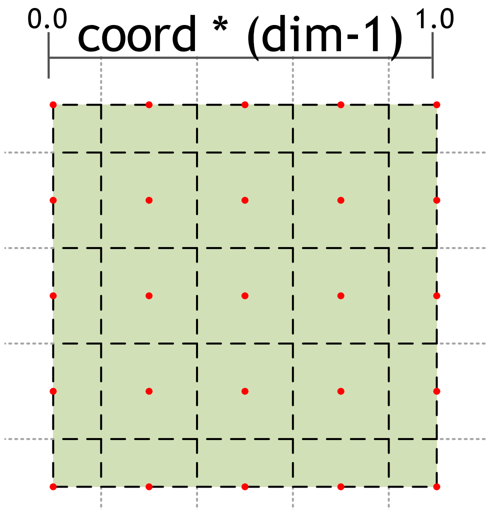


Traditional clamped texture



Visible seams due to interpolation

Corner sampled texture



All samples interpolated equally

```
VkImageCreateInfo info = {...};
info.flags |= VK_IMAGE_CREATE_CORNER_SAMPLED_BIT_NV;
```

REPRESENTATIVE FRAGMENT TEST

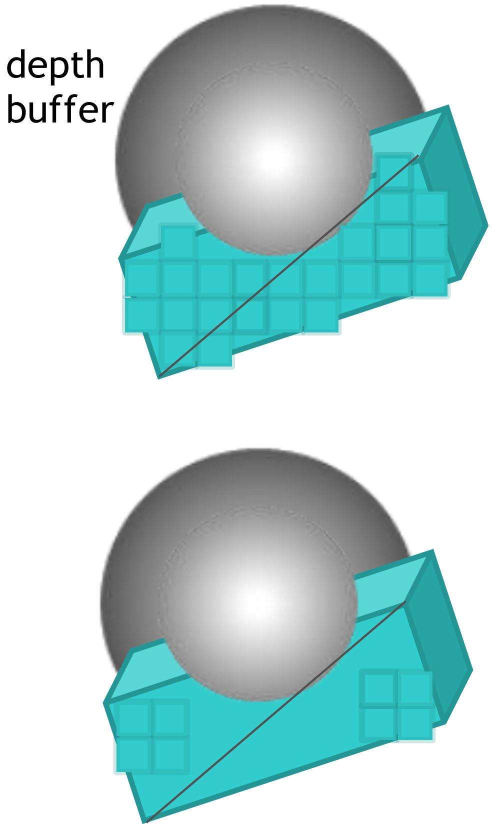
FASTER OCCLUSION TESTS

VK_NV_representative_fragment_test

This extension can help shader-based occlusion queries that draw many object proxies at once.

Enabling can reduce fragment-shader invocations when proxy primitives take up larger portions of the screen.

https://github.com/nvpro-samples/gl_occlusion_culling




```
// depth-test passing
// fragments tag objects
// as visible
layout(early_fragment_tests) in;
...
  visibility[objectID] = 1;
```

Representative test OFF: primitives are rastered completely

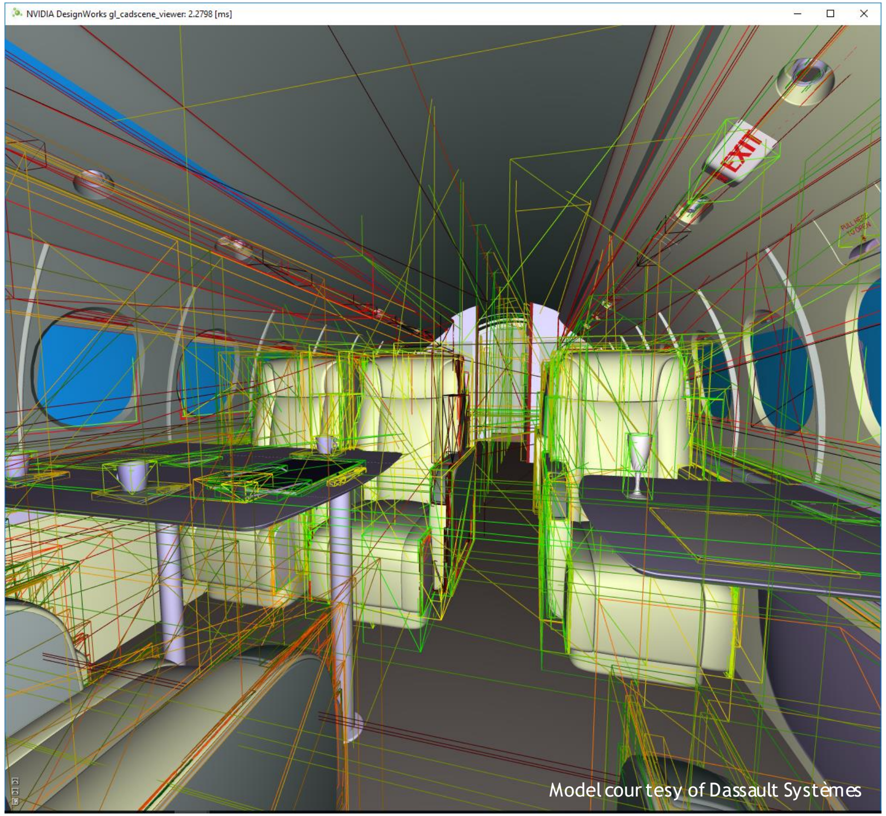
Representative test ON: primitives can be rastered partially

FASTER OCCLUSION TESTS

VR-like scenario, occlusion test for ~9k bboxes at 2048 x 2048 x 2x msaa

Representative test OFF: 0.5 ms

Representative test ON: 0.15 ms

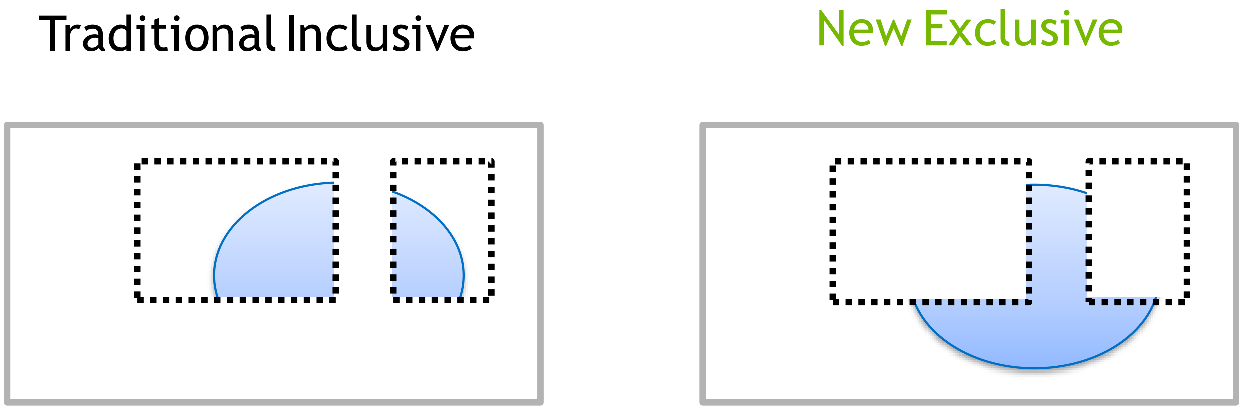


EXCLUSIVE SCISSOR TEST

EXCLUSIVE SCISSOR

VK_NV_scissor_exclusive

Can reverse the scissor-test to „stamp out“ areas




```
// specify at PSO create-time
VkPipelineViewportExclusiveScissorStateCreateInfoNV info;
psoInfo.next = &info;
info.pExclusiveScissors = {{offset,extent},..};
..
// or use dynamic state
vkCmdSetExclusiveScissorNV(cmd, first, count, rectangles);
```

CROSS API INTEROP

CROSS API INTEROP

Vulkan or DX12 as exporters

Vulkan® Microsoft® DirectX® 12

OpenGL®

NVIDIA® CUDA® 10.0

Allocated Memory

Buffer | Image

Export Allocation

VK 1.1 or VK_KHR_external_memory
ID3D12Heap, ID3D12Resource

Imported Memory

Buffer | Texture

Directly Reference Allocation

GL_EXT_memory_object
GL_EXT_memory_attachment

Imported Memory

device memory | cuArray

Directly Reference Allocation

cudaExternalMemory_t

Synchronization Object

Import Semaphore

GL_EXT_semaphore

Synchronization Object

Export Semaphore

VK_KHR_external_semaphore
ID3D12Fence

Synchronization Object

Import Semaphore

cudaExternalSemaphore_t

```
nv_spec_contributors
{
  Jeff Bolz
  Pierre Boudier
  Pat Brown
  Chao Chen
  Piers Daniell
  Mark Kilgard
  Pyarelal Knowles
  Daniel Koch
  Christoph Kubisch
  Chris Lentini
  Sahil Parmar
  Tyson Smith
  Markus Tavenrath
  Kedarnath Thangudu
  Yury Uralsky
  Eric Werness
};
```

ckubisch@nvidia.com (professional vis, GL/VK) @pixeljetstream

THANK YOU

[1] www.facebook.com/artbyrens

[2] https://www.flickr.com/photos/14136614@N03/6209344182

[3] k-DOPs as Tighter Bounding Volumes for Better Occlusion Performance - Bartz, Klosowski & Staneker https://pdfs.semanticscholar.org/bf4e/7c405d0f2a259f78e91ce1eb68a5d794c99b.pdf

[4] GTC 2016 - OpenGL Blueprint Rendering - Christoph Kubisch http://on-demand.gputechconf.com/gtc/2016/presentation/s6143-christoph-kubisch-blueprint-rendering.pdf

[5] https://developer.nvidia.com/sites/default/files/akamai/gamedev/docs/Borderless%20Ptex.pdf