



DirectX

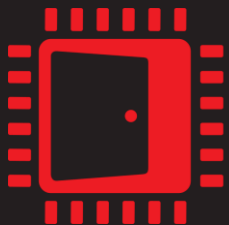


ULTIMATE

DIRECTX[®] 12 ULTIMATE

SAMPLER FEEDBACK & MESH SHADERS

COLIN RILEY



AMD
GPU Open

AMD
RDNA 2

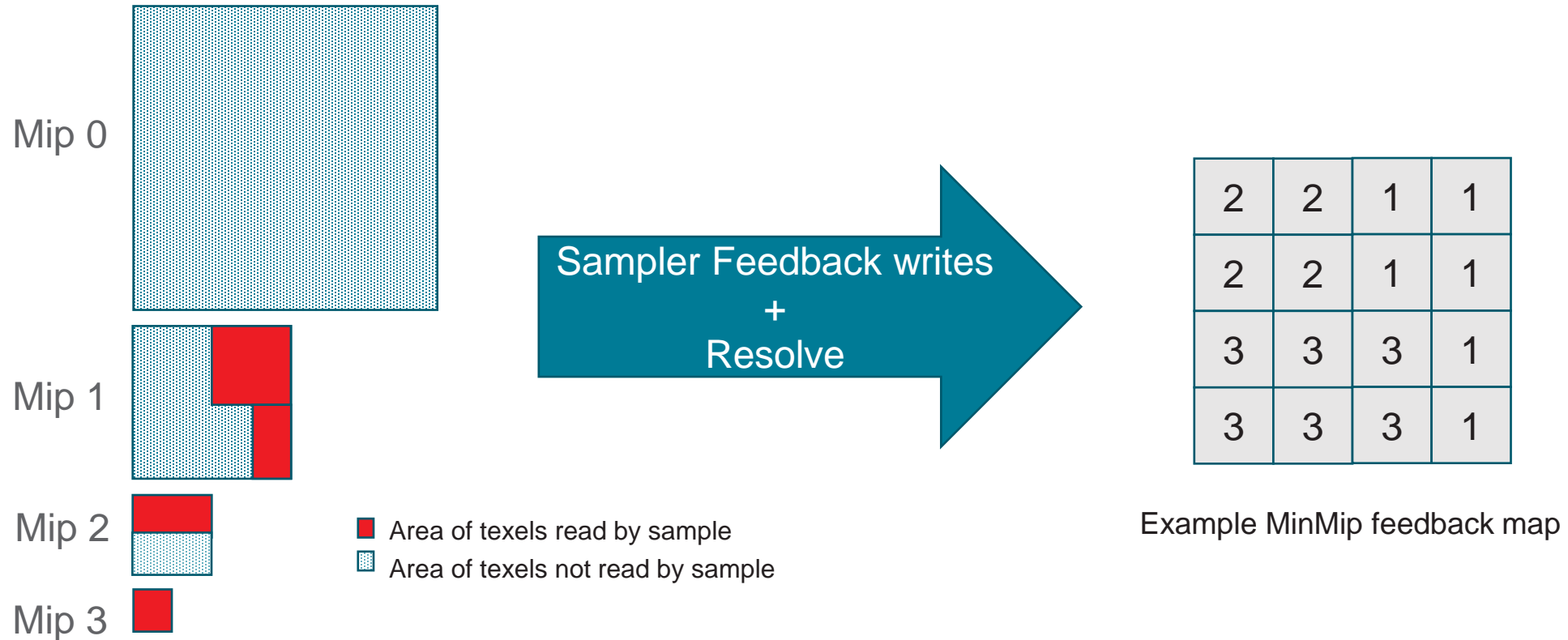
WHAT'S NEW?

- Description of DirectX® 12 Ultimate features
 - Sampler Feedback
 - Mesh Shaders
- AMD RDNA™ 2 PC performance recommendations

SAMPLER FEEDBACK

- Feature which is enabled by new hardware functionality.
- Provides information about what texture data is read by a sample operation.
- Two feedback map formats:
 - MinMip – Records the highest Mip level requested in the sample operation.
 - MipRegionUsed – Records all requested Mip levels used in the sample operation.
- Feedback is written in HLSL into a feedback resource, at a lower resolution than the texture.
- Feedback resources need decoded via a resolve operation into a usable form.

SAMPLER FEEDBACK



SAMPLER FEEDBACK RECOMMENDATIONS ON AMD RDNA™ 2 ON PC

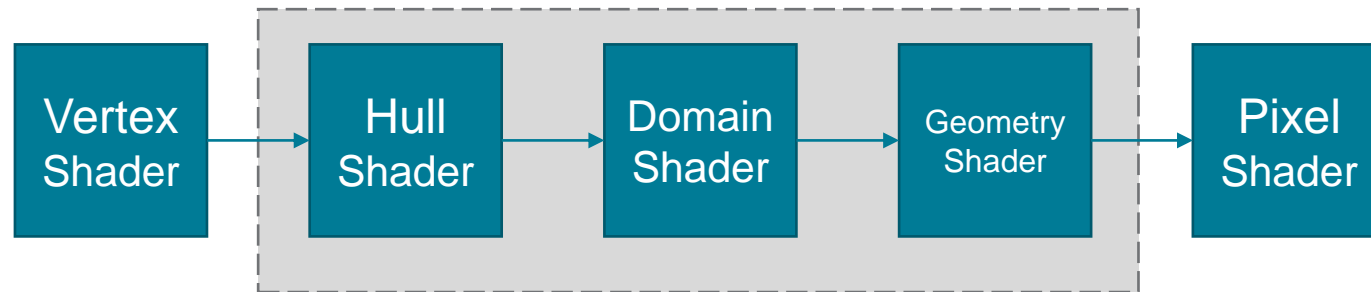
- Avoid writing feedback for every shaded pixel.
 - Stochastic discard on a pixel shader performing feedback writes can improve performance.
 - Little to no image quality difference on high percentage levels of discard.
 - Experiment to find a level which works well for you.

MESH SHADERS

- Feature which gives greater control of the geometry pipeline of GPUs.
- Provides new concepts and shader stages to achieve this.
- Processing smaller batches of primitives called “Meshlets” in parallel.

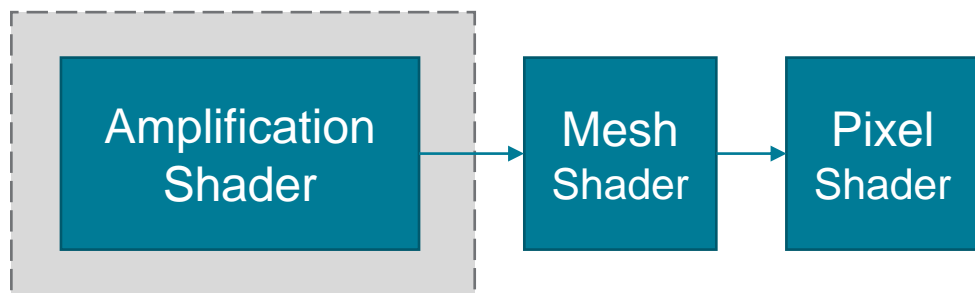
TRADITIONAL GEOMETRY PIPELINE

- Traditional Vertex -> Pixel shader pipelines process geometry whole and in-order using a linear index buffer.
- Data processing required before significant user-defined culling can occur.
- Fixed and rigid data types for input assembly.
- Many user specified stages depend on fixed function blocks.



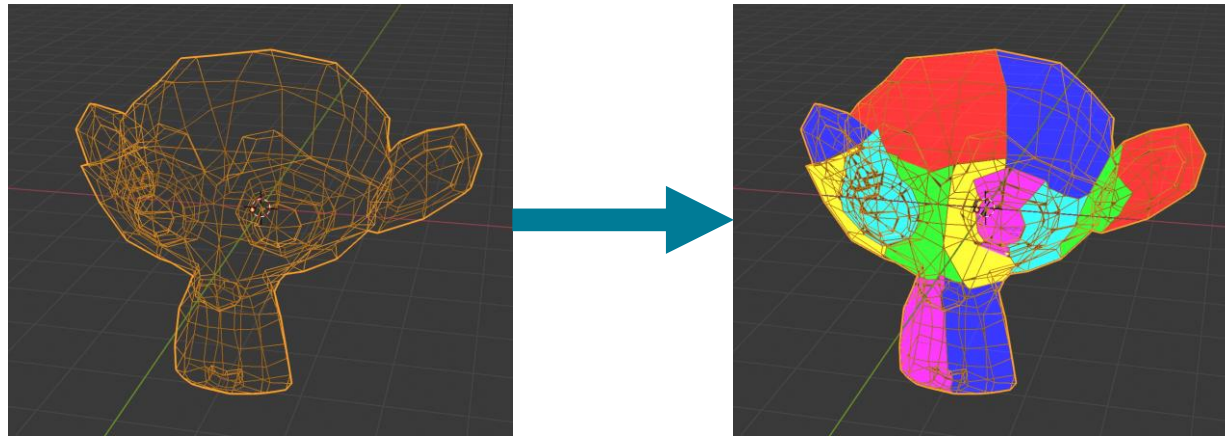
MESH SHADER GEOMETRY PIPELINE

- Compute based programming model.
- Input data is user defined via the graphics root signature.
- Threads can use groupshared memory.
- Optional Amplification shader stage dispatch mesh threadgroups.
- Mesh shader threadgroups write small indexed primitive lists, known as Meshlets, into shared output arrays.
 - Meshlets then proceed to rasterization and Pixels Shader stages.



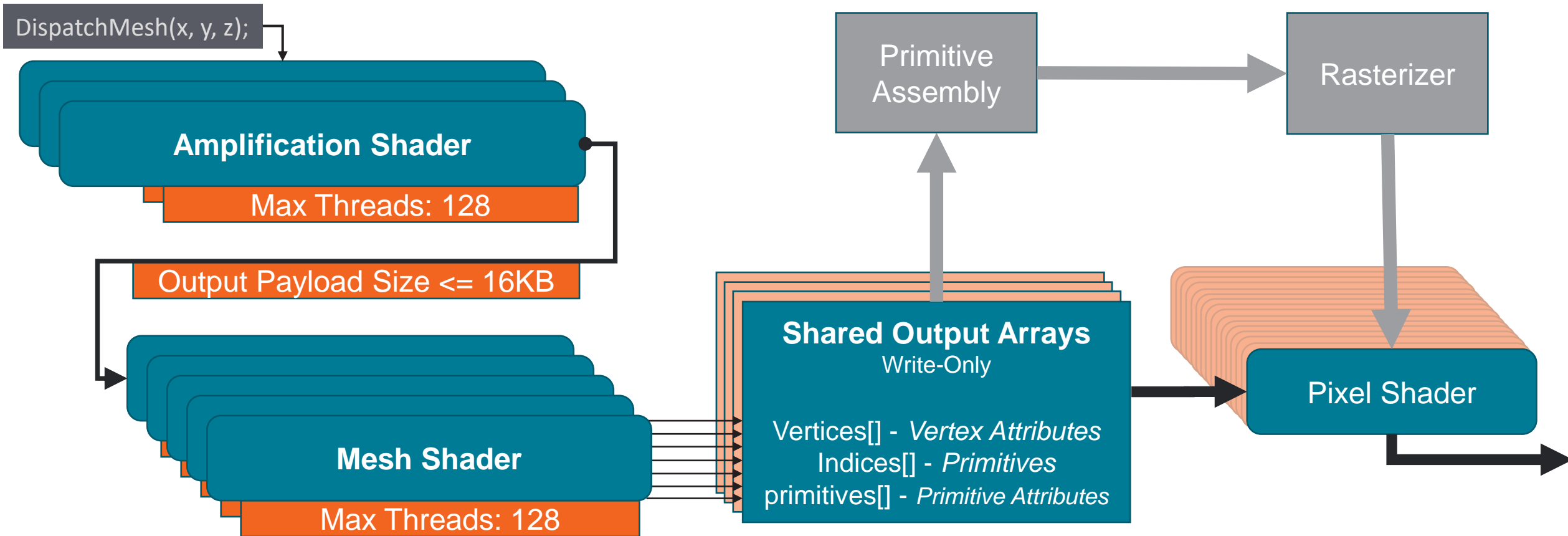
WHAT'S A MESHLET?

- A small submesh which is optimized and precalculated for vertex reuse.
- They have fixed bounds in terms of vertices and primitives such that a single mesh shader threadgroup outputs a single meshlet.
- Associated with additional data, like bounding boxes, which can be used to cull meshlets.
- Requires changes to geometry asset building so there can be considerable upfront cost.



13 Meshlets

MESH SHADER PIPELINE DATAFLOW



WHAT DOES MESH SHADER PROVIDE

- Data laid out in ways which make sense for your engine architecture, be it compressed geometry or hierarchical levels of detail, which can now be accessed directly in the shader.
- The possibility of custom culling, without the additional compute geometry pre-passes which are common now.
- Level of Detail and Tessellation decisions made in the Amplification Shader can be provided to Mesh shaders through the payload, impacting what meshlets are rendered and the number of them.
- Much better method for implementing Geometry-shader style pipelines.
- Requires investment in asset pipelines to output good meshlets.

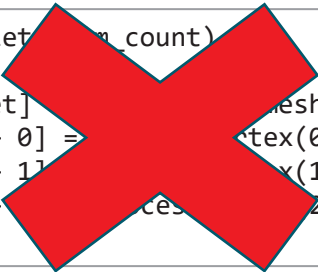
MESH SHADERS

RECOMMENDATIONS ON AMD RDNA™ 2 ON PC

- Ideal primitive and vertex count for meshlets is between 64 and 128.
- Restrict writes to the shared output arrays such that each thread only writes to its own threadID location within the arrays, with threadgroup size set to the maximum of vertex or primitive count.

```
if (threadid < meshlet.prim_count)
{
    prims[threadid] = ProcessPrim(meshlet, threadid);
}
if (threadid < meshlet.vert_count)
{
    verts[threadid] = ProcessVertex(meshlet, threadid);
}
```

```
if (threadid < meshlet.prim_count)
{
    prims[prim_offset + threadid] = ProcessPrim(meshlet, threadid);
    verts[v_offset + 0] = ProcessVertex(meshlet, threadid);
    verts[v_offset + 1] = ProcessVertex(meshlet, threadid);
    verts[v_offset + 2] = ProcessVertex(meshlet, threadid);
}
```



- Try to dispatch at least 4 mesh threadgroups from the Amplification stage, each writing at least 64 primitives to hide latency.

MESH SHADERS

RECOMMENDATIONS ON AMD RDNA™ 2 ON PC

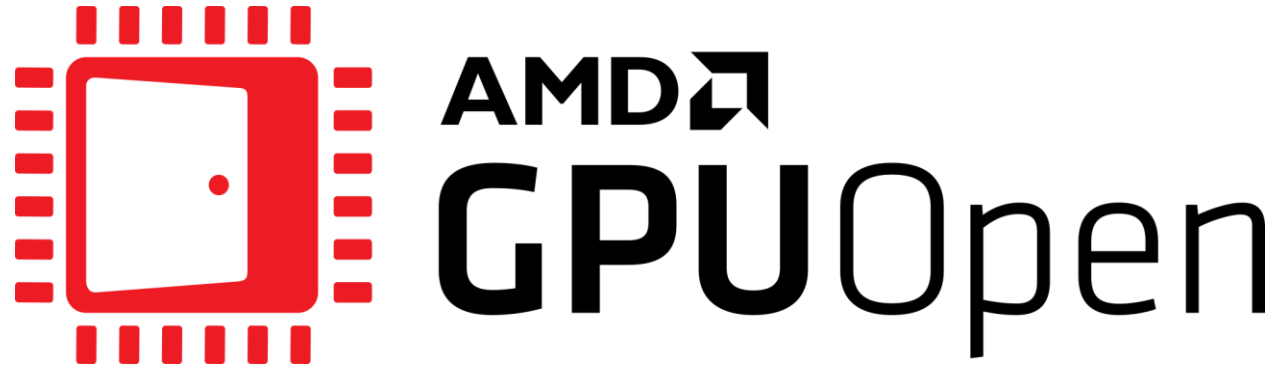
- Do coarse culling, instancing and level of detail calculations in the Amplification stage.
- Don't perform per-triangle culling in the mesh shader, however, if some culling is essential, use the `SV_CullPrimitive` attribute instead of writing degenerate triangles.
- Be wary of using bitmasks on groupshared variables for visibility testing and culling in the amplification shader.

MESH SHADERS

RECOMMENDATIONS ON AMD RDNA™ 2 ON PC

- Whilst the Pixel Shader stage of a mesh shader pipeline is optional, be aware that omitting it and using Mesh shaders as a means for GPU work creation is not likely the most efficient use of the hardware.
- If you have Geometry Shader passes in your engine, try porting them to Mesh Shaders.
 - This is a low-hanging fruit introduction to mesh shaders which can provide better performance.
- Profile your mesh shader implementations to ensure they perform better than any legacy stages they may be replacing. There will still be areas where traditional pipelines will win in performance

LEARN MORE AT GPUOPEN.COM



RADEON



Contact Information:

Colin.Riley@amd.com

<http://www.twitter.com/GPUOpen>

DISCLAIMER

The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale.

AMD, the AMD Arrow logo, Epyc, Radeon, RDNA, Ryzen, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

DirectX is a registered trademark of Microsoft Corporation in the US and other jurisdictions.

© 2020 Advanced Micro Devices, Inc. All rights reserved.