Replacing the geometry pipeline with mesh shaders

Ricardo Garcia

October 4
XDC 2022, Minneapolis

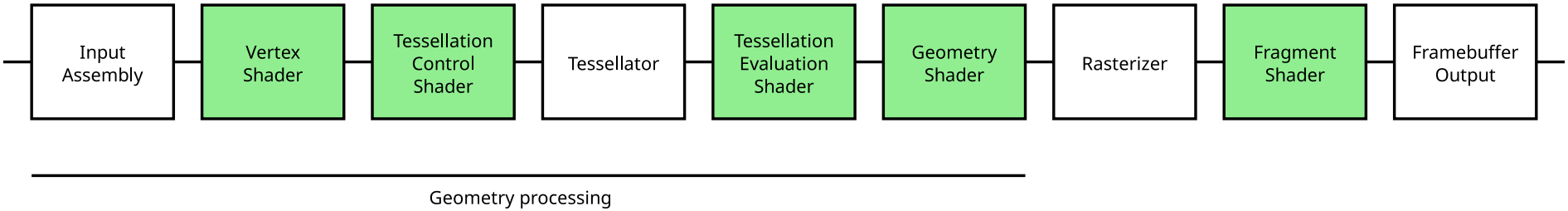
Main Points

- What are mesh shaders?
- Comparison between classic and mesh shading pipelines
- Problems mesh shaders try to solve
- What a mesh shader looks like
- Problems introduced by mesh shaders

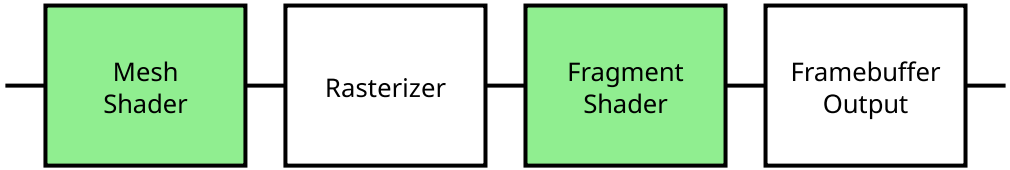
What is mesh shading?

- New type of graphics pipeline with simplified stages
- Tries to address shortcoming with classic graphics pipelines on modern HW
- Brings the geometry part of the pipeline closer to the compute model

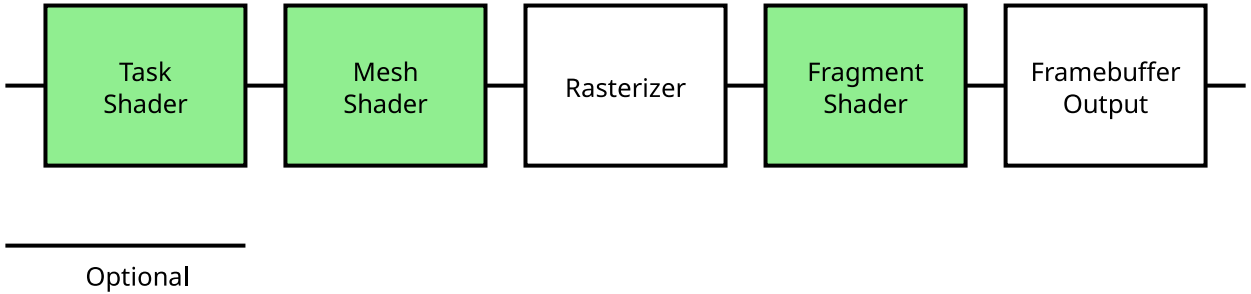
Classic Graphics Pipeline



Mesh Shading Pipeline



Mesh Shading Pipeline (Full)



Classic Pipeline Problems

- Vertex inputs are annoying
- No control over how geometry is arranged
- Waste of compute resources
- Waste of memory bandwidth
- Geometry amplification and modification should be simpler and more flexible
- Maybe we can use a model that’s closer to compute shaders

How do they look?

```
#version 450
#extension GL_EXT_mesh_shader : enable




void main ()
{




}
```


How do they look?

```
#version 450
#extension GL_EXT_mesh_shader : enable

layout (local_size_x=X, local_size_y=Y, local_size_z=Z) in; // Typical limit: 128 invocations.




void main ()
{
    // Typical compute built-ins: gl_NumWorkGroups, gl_WorkGroupID, gl_LocalInvocationID, etc.
    // Typical subgroup functionality: gl_NumSubgroups, gl_SubgroupID, subgroupElect(), etc.




}
```


How do they look?

```
#version 450
#extension GL_EXT_mesh_shader : enable

layout (local_size_x=X, local_size_y=Y, local_size_z=Z) in; // Typical limit: 128 invocations.
layout (triangles) out;                                     // May also be points or lines.




void main ()
{
    // Typical compute built-ins: gl_NumWorkGroups, gl_WorkGroupID, gl_LocalInvocationID, etc.
    // Typical subgroup functionality: gl_NumSubgroups, gl_SubgroupID, subgroupElect(), etc.




}
```


How do they look?

```
#version 450
#extension GL_EXT_mesh_shader : enable

layout (local_size_x=X, local_size_y=Y, local_size_z=Z) in; // Typical limit: 128 invocations.
layout (triangles) out;                                     // May also be points or lines.
layout (max_vertices=V, max_primitives=P) out;              // Typical limit: 256 vert/prim.



void main ()
{
    // Typical compute built-ins: gl_NumWorkGroups, gl_WorkGroupID, gl_LocalInvocationID, etc.
    // Typical subgroup functionality: gl_NumSubgroups, gl_SubgroupID, subgroupElect(), etc.




}
```


How do they look?

```
#version 450
#extension GL_EXT_mesh_shader : enable

layout (local_size_x=X, local_size_y=Y, local_size_z=Z) in; // Typical limit: 128 invocations.
layout (triangles) out;                                     // May also be points or lines.
layout (max_vertices=V, max_primitives=P) out;              // Typical limit: 256 vert/prim.



void main ()
{
    // Typical compute built-ins: gl_NumWorkGroups, gl_WorkGroupID, gl_LocalInvocationID, etc.
    // Typical subgroup functionality: gl_NumSubgroups, gl_SubgroupID, subgroupElect(), etc.

    SetMeshOutputsEXT(ACTUAL_V, ACTUAL_P);

    gl_MeshVerticesEXT[FOO].gl_Position = vec4(…);
    gl_PrimitiveTriangleIndicesEXT[BAR] = uvec3(…);
}
```


How do they look?

```
#version 450
#extension GL_EXT_mesh_shader : enable

layout (local_size_x=X, local_size_y=Y, local_size_z=Z) in; // Typical limit: 128 invocations.
layout (triangles) out;                                     // May also be points or lines.
layout (max_vertices=V, max_primitives=P) out;              // Typical limit: 256 vert/prim.

layout (location=0) out vec4 out0[];                        // Per-vertex.
layout (location=1) perprimitiveEXT out vec4 out1[];        // Per-primitive.

void main ()
{
    // Typical compute built-ins: gl_NumWorkGroups, gl_WorkGroupID, gl_LocalInvocationID, etc.
    // Typical subgroup functionality: gl_NumSubgroups, gl_SubgroupID, subgroupElect(), etc.

    SetMeshOutputsEXT(ACTUAL_V, ACTUAL_P);

    gl_MeshVerticesEXT[FOO].gl_Position = vec4(…);
    gl_PrimitiveTriangleIndicesEXT[BAR] = uvec3(…);
}
```


Some built-in arrays

```
// write only access
out uint  gl_PrimitivePointIndicesEXT[];
out uvec2 gl_PrimitiveLineIndicesEXT[];
out uvec3 gl_PrimitiveTriangleIndicesEXT[];

// write only access
out gl_MeshPerVertexEXT {
  vec4  gl_Position;
  float gl_PointSize;
  float gl_ClipDistance[];
  float gl_CullDistance[];
} gl_MeshVerticesEXT[];

// write only access
perprimitiveEXT out gl_MeshPerPrimitiveEXT {
  int  gl_PrimitiveID;
  int  gl_Layer;
  int  gl_ViewportIndex;
  bool gl_CullPrimitiveEXT;
  int  gl_PrimitiveShadingRateEXT;
} gl_MeshPrimitivesEXT[];
```

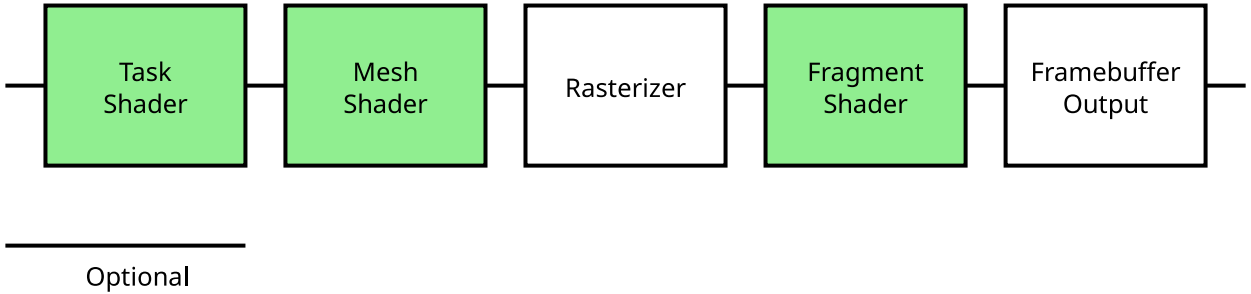

Meshlets

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

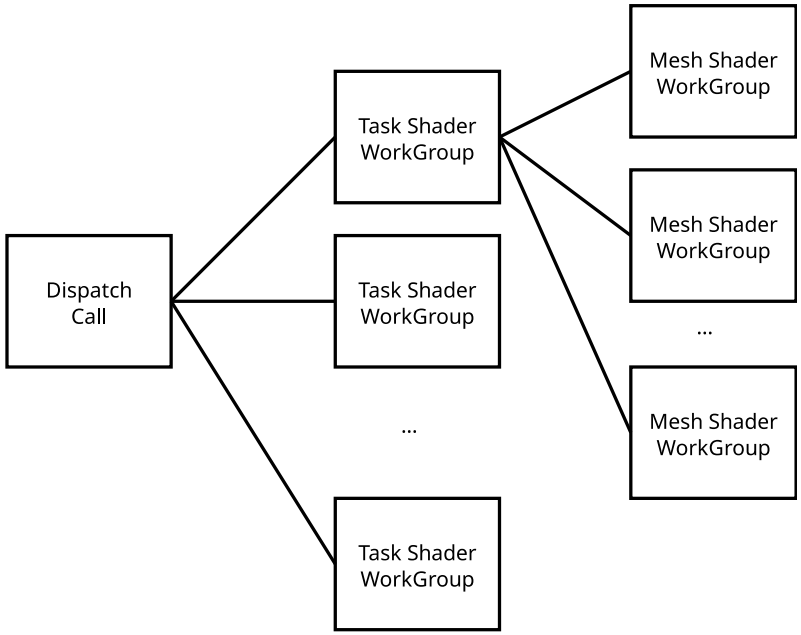
Dispatching Work Groups

- vkCmdDrawMeshTasksEXT(cmdBuffer, X, Y, Z);
- vkCmdDrawMeshTasksIndirectEXT(cmdBuffer, ...)
- vkCmdDrawMeshTasksIndirectCountEXT(cmdBuffer, ...);

Mesh Shading Pipeline (Full)

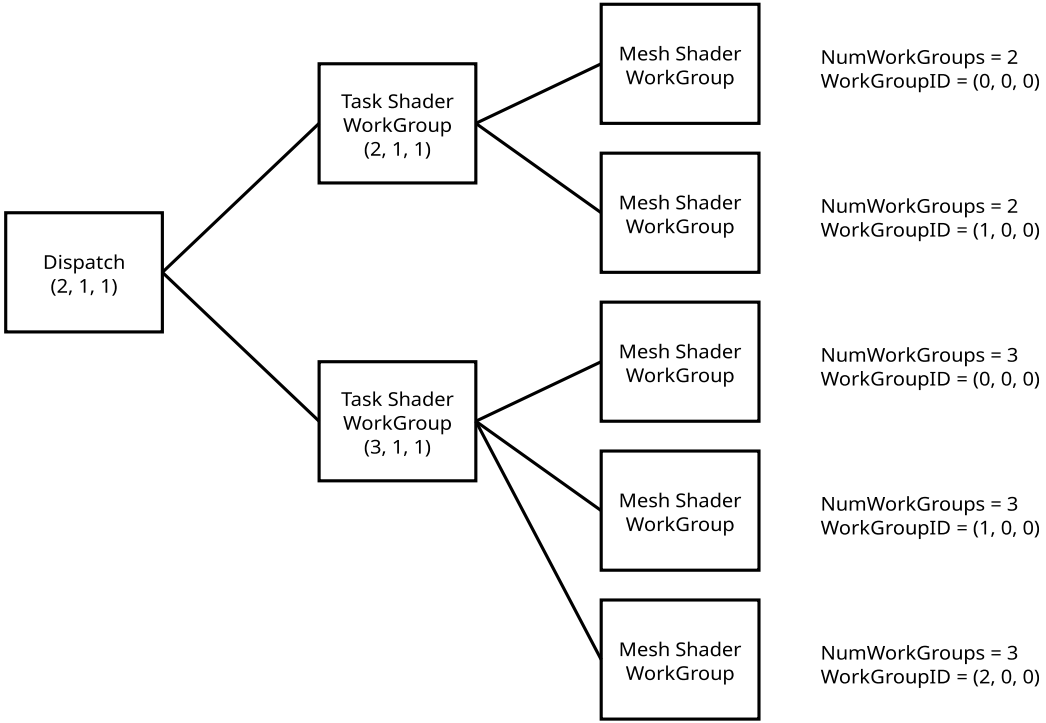


Task (Amplification) Shader



Similar to a two-level compute dispatch

Task (Amplification) Shader



To uniquely identify the work that needs to be done in each mesh shader work group, info needs to be passed from the task shader to the mesh shader.

Payload

```
#version 450
#extension GL_EXT_mesh_shader : enable

// Typical limit: 128 invocations.
layout (local_size_x=X, local_size_y=Y, local_size_z=Z) in;

struct TaskData {
    …
};
taskPayloadSharedEXT TaskData td;

void main ()
{
    // Prepare payload for children.
    td.SOMETHING = SOMETHING_ELSE;

    // Example: (2, 1, 1)
    EmitMeshTasksEXT(MeshX, MeshY, MeshZ);

}
```

```
#version 450
#extension GL_EXT_mesh_shader : enable

layout (local_size_x=X, local_size_y=Y, local_size_z=Z) in;
layout (triangles) out;
layout (max_vertices=V, max_primitives=P);

struct TaskData {
    …
};
taskPayloadSharedEXT TaskData td;

void main ()
{
    // Use data from td.
    SetMeshOutputsEXT(ACTUAL_V, ACTUAL_P);

    gl_MeshVerticesEXT[FOO].gl_Position = …;
    gl_MeshTriangleIndicesEXT[BAR] = uvec3(…);
}
```


Mesh Shading Pros

- Avoids input assembly bottlenecks.
- Can pre-compute data and discard geometry in advance.
- Can save memory bandwidth by not pulling in unneeded data.
- Flexible geometry amplification.
- Non-sequential programming model.
- Can be used to streamline compute pre-passes into pipeline.
- Can be used as a two-level compute pipeline.

Mesh Shading Cons

- Problematic for tilers.
- Leaving choices to the user increases room for errors.
- Increases coupling of in-memory vertex data with shaders.
- Different vendors recommend different things for performance.
  - Exposed as properties in the extension.
  - Lower number of invocations, loop.
  - Higher number of invocations, access arrays with gl_LocalInvocationIndex.

Q&A


igalia