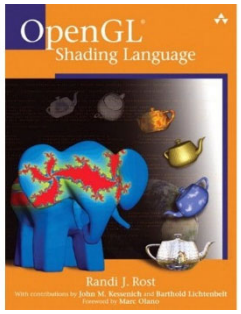
CS 380 - GPU and GPGPU Programming
Lecture 3: Introduction, Pt. 3

Markus Hadwiger, KAUST

Reading Assignment #2 (until Sep 11)

Read (required):

- Orange book (GLSL), Chapter 4
  (*The OpenGL Programmable Pipeline*)



- Nice brief overviews of GLSL and legacy assembly shading language

`https://en.wikipedia.org/wiki/OpenGL_Shading_Language`

`https://en.wikipedia.org/wiki/ARB_assembly_language`

- GPU Gems 2 book, Chapter 30
  (*The GeForce 6 Series GPU Architecture*)

`http://download.nvidia.com/developer/GPU_Gems_2/GPU_Gems2_ch30.pdf`

Programming Assignments: Schedule (tentative)

Assignment #1:

- Querying the GPU (OpenGL/GLSL and CUDA) — due Sep 4

Assignment #2:

- Phong shading and procedural texturing (GLSL) — due Sep 18

Assignment #3:

- Deferred Shading and Image Processing with GLSL — due Oct 2

Assignment #4:

- Image Processing with CUDA
- Convolutional layers with CUDA — due Oct 23

Assignment #5:

- Linear Algebra (CUDA) — due Nov 13

What is in a GPU?

Lots of floating point processing power

- Processors, different names:
  ALUs,
  stream processors (SP),
  CUDA cores,
  FP32 cores, FP64 cores, ...

- Was vector processing, now scalar cores!

Still lots of fixed graphics functionality

- Attribute interpolation (per-vertex → per-fragment)
- Rasterization (triangles → fragments/pixels)
- Texture sampling and filtering
- Depth buffering (per-pixel visibility)
- Blending/compositing (semi-transparent geometry, ...)
- Frame buffers (and implicit atomic operations in ROPs)

NVIDIA Volta SM

Multiprocessor: SM

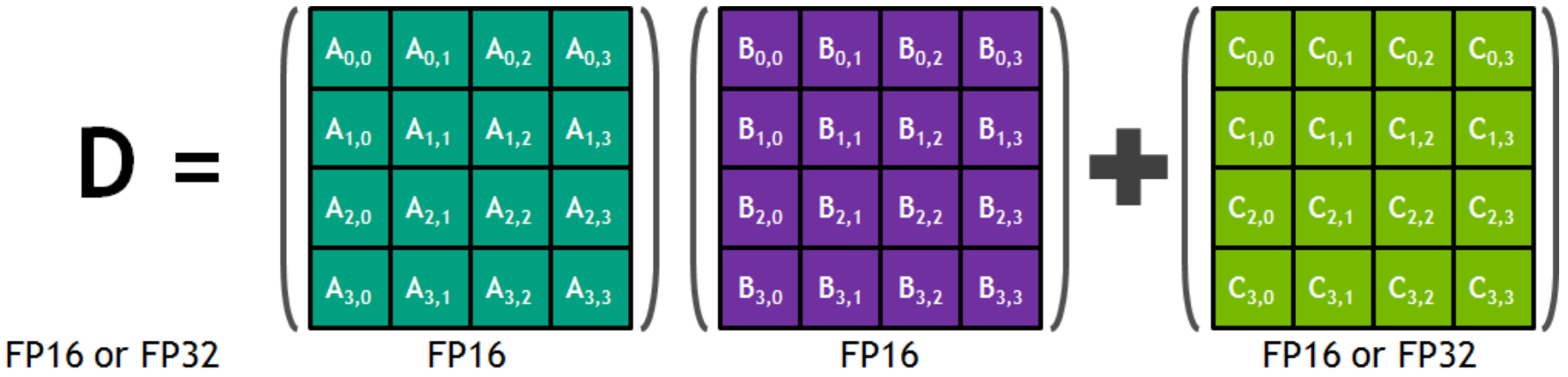
- 64 FP32 + INT32 cores
- 32 FP64 cores
- 8 tensor cores (FP16/FP32 mixed-precision)

4 partitions inside SM

- 16 FP32 + INT32 cores each
- 8 FP64 cores each
- 8 LD/ST units each
- 2 tensor cores each
- Each has: warp scheduler, dispatch unit, register file

Example for “Special Cores”: Tensor Cores

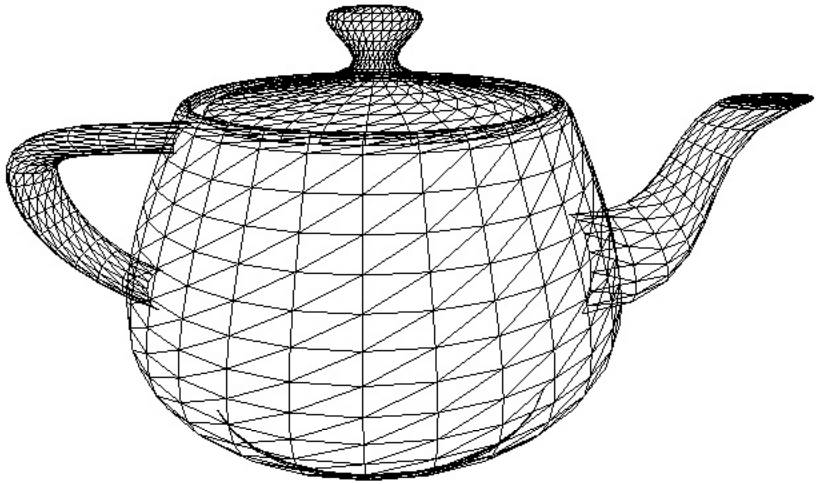
Mixed-precision, fast matrix-matrix multiply and accumulate



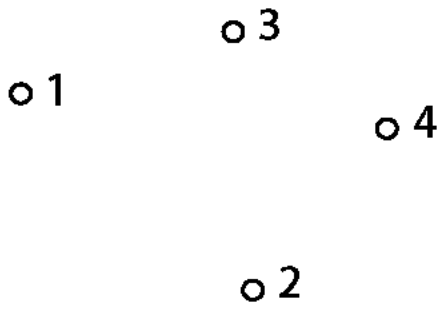
From this, build larger sizes, higher dimensionalities, ...

Newer versions have additional precisions/formats, …

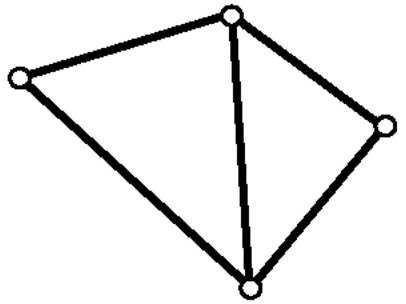
Real-time graphics primitives (entities)



Represent surface as a 3D triangle mesh



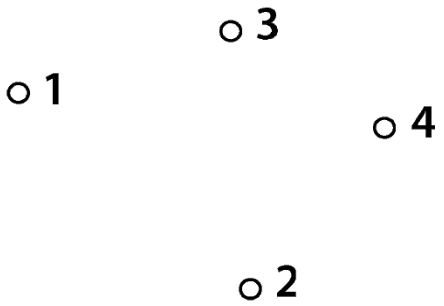
Vertices



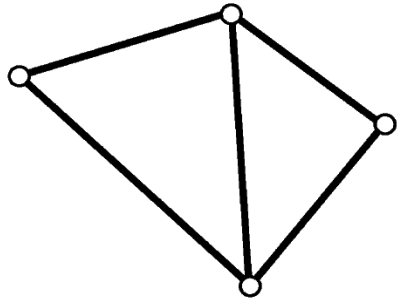
Primitives
(e.g., triangles, points, lines)

Courtesy Kayvon Fatahalian, CMU

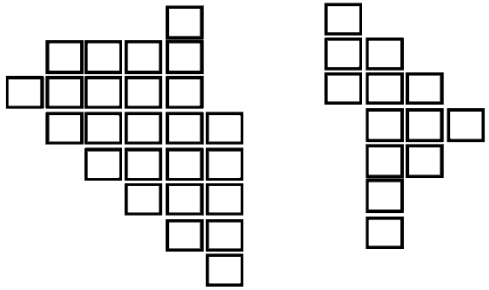
Real-time graphics primitives (entities)



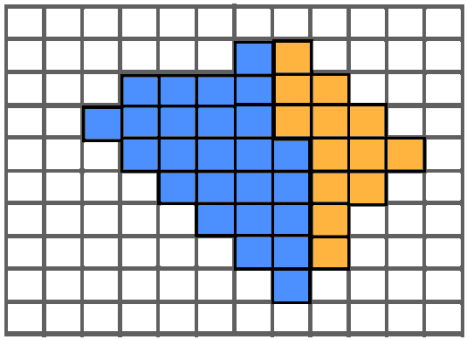
Vertices



Primitives
(e.g., triangles, points, lines)



Fragments

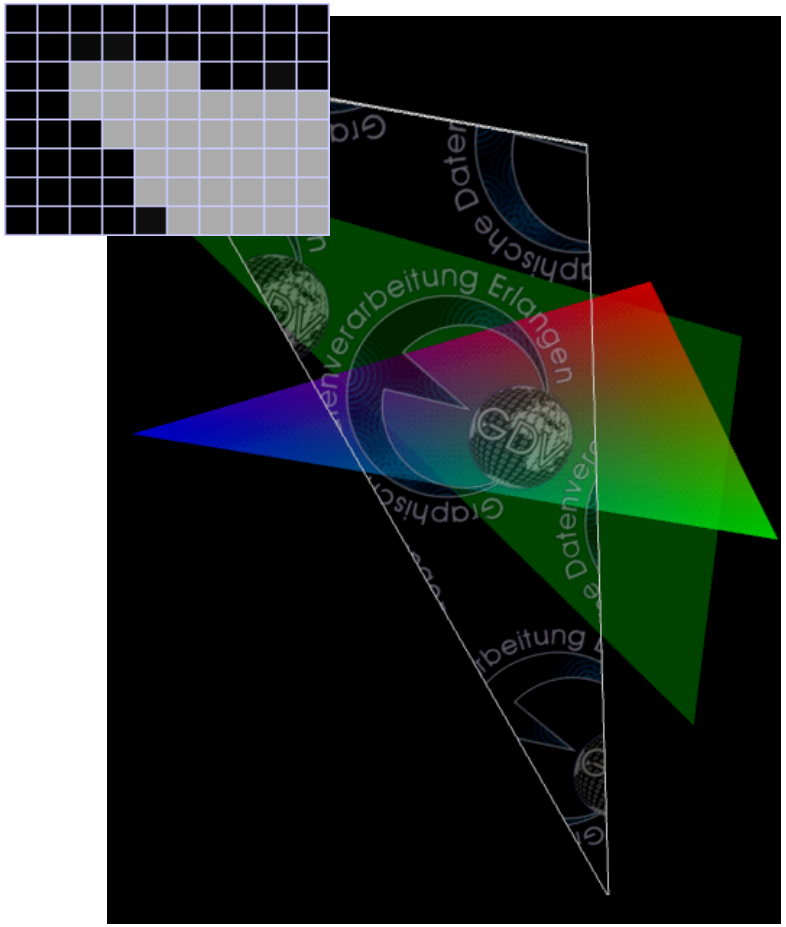


Pixels (in an image)

Courtesy Kayvon Fatahalian, CMU

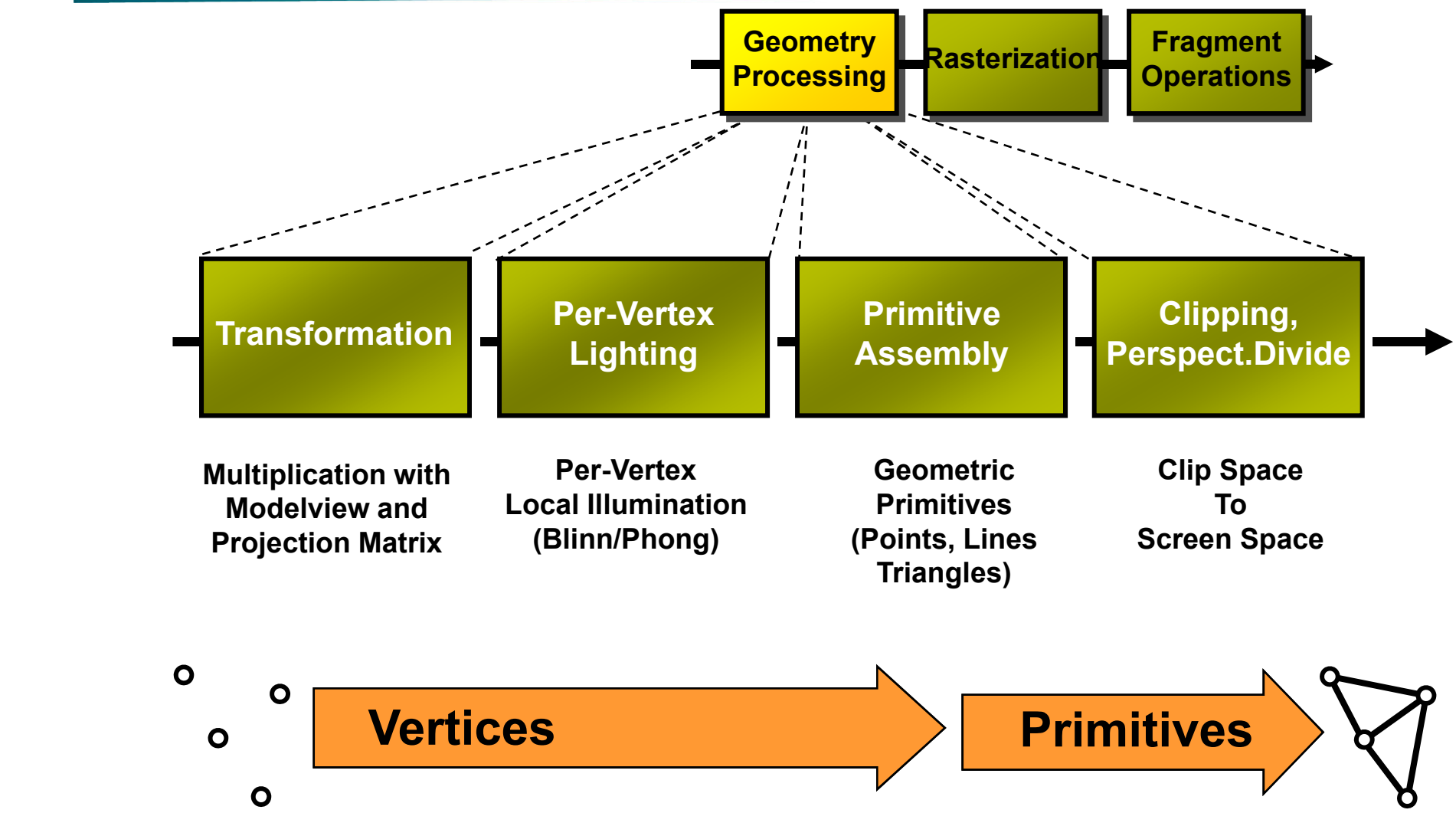
What can the hardware do?

- **Rasterization**
  - Decomposition into fragments
  - Interpolation of color
  - Texturing
    - Interpolation/filtering
    - Fragment shading
- **Fragment operations (or: raster operations)**
  - Depth test (Z-test)
  - Alpha blending (compositing)
  - ...



Graphics Pipeline

Geometry Processing



Rasterization

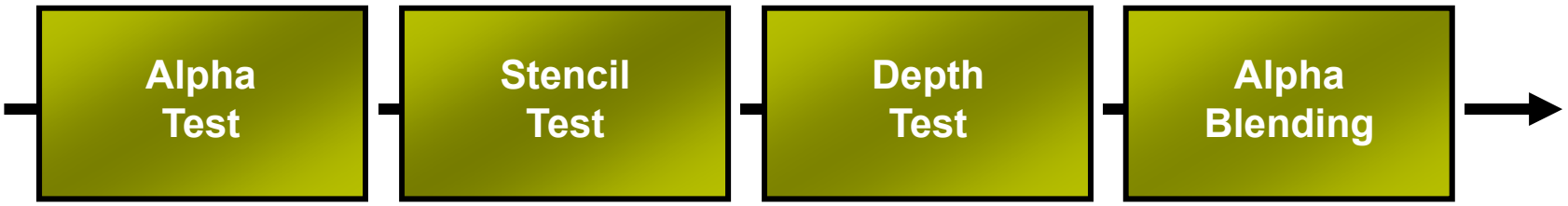
Decomposition of primitives into fragments

Interpolation of texture *coordinates*

Filtering of texture color

Combination of primary color with texture color

Fragment (Raster) Operations

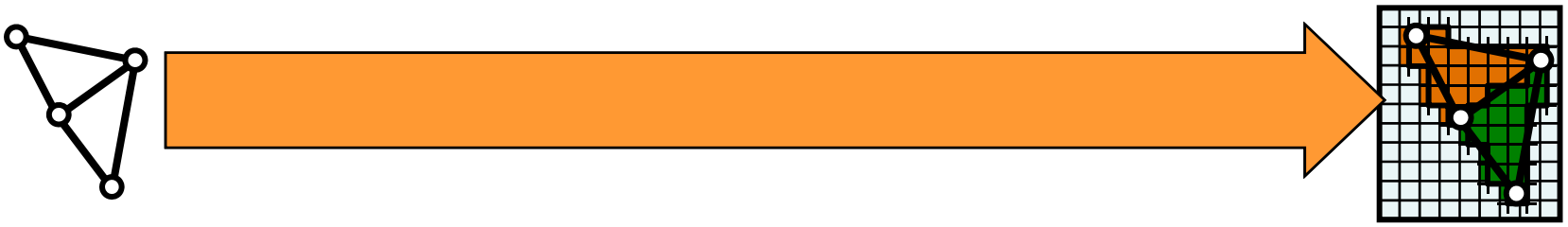


Discard all fragments within a certain alpha range

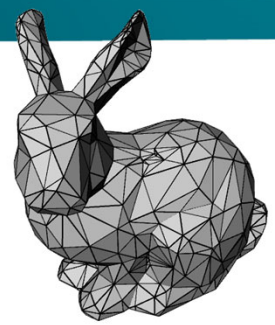
Discard a fragment if the stencil buffer is set

Discard all occluded fragments

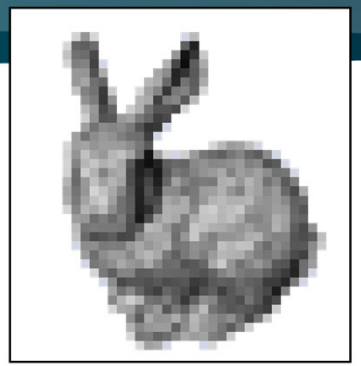
Combination of primary color with texture color



Graphics Pipeline



Scene Description



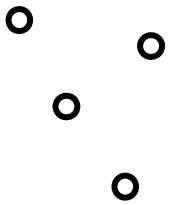
Raster Image

Programmable Pipeline

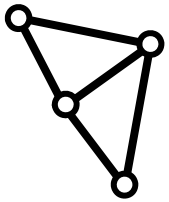
Vertex Shader

Fragment Shader

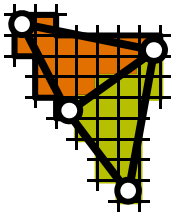
Fragment Operations



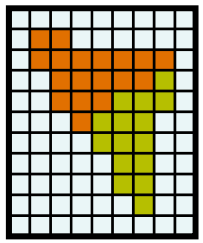
Vertices



Primitives

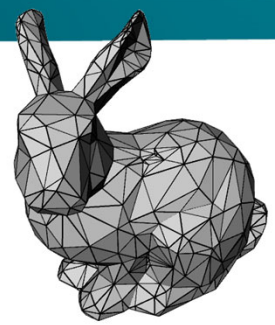


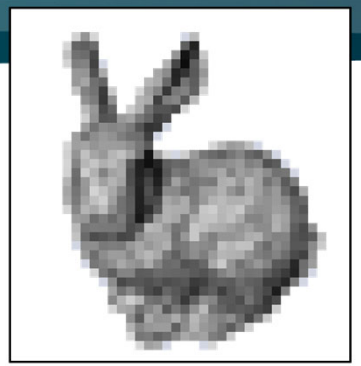
Fragments



Pixels

Graphics Pipeline





Scene Description

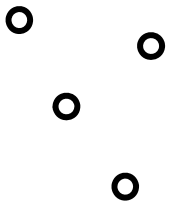
Programmable Pipeline

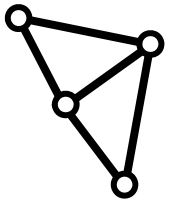
Raster Image

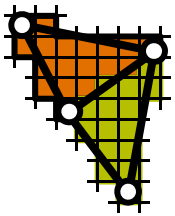
Vertex Shader

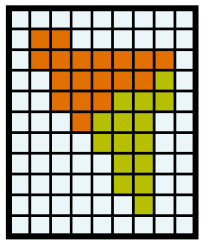
Fragment Shader

Fragment Operations









Vertices

Primitives

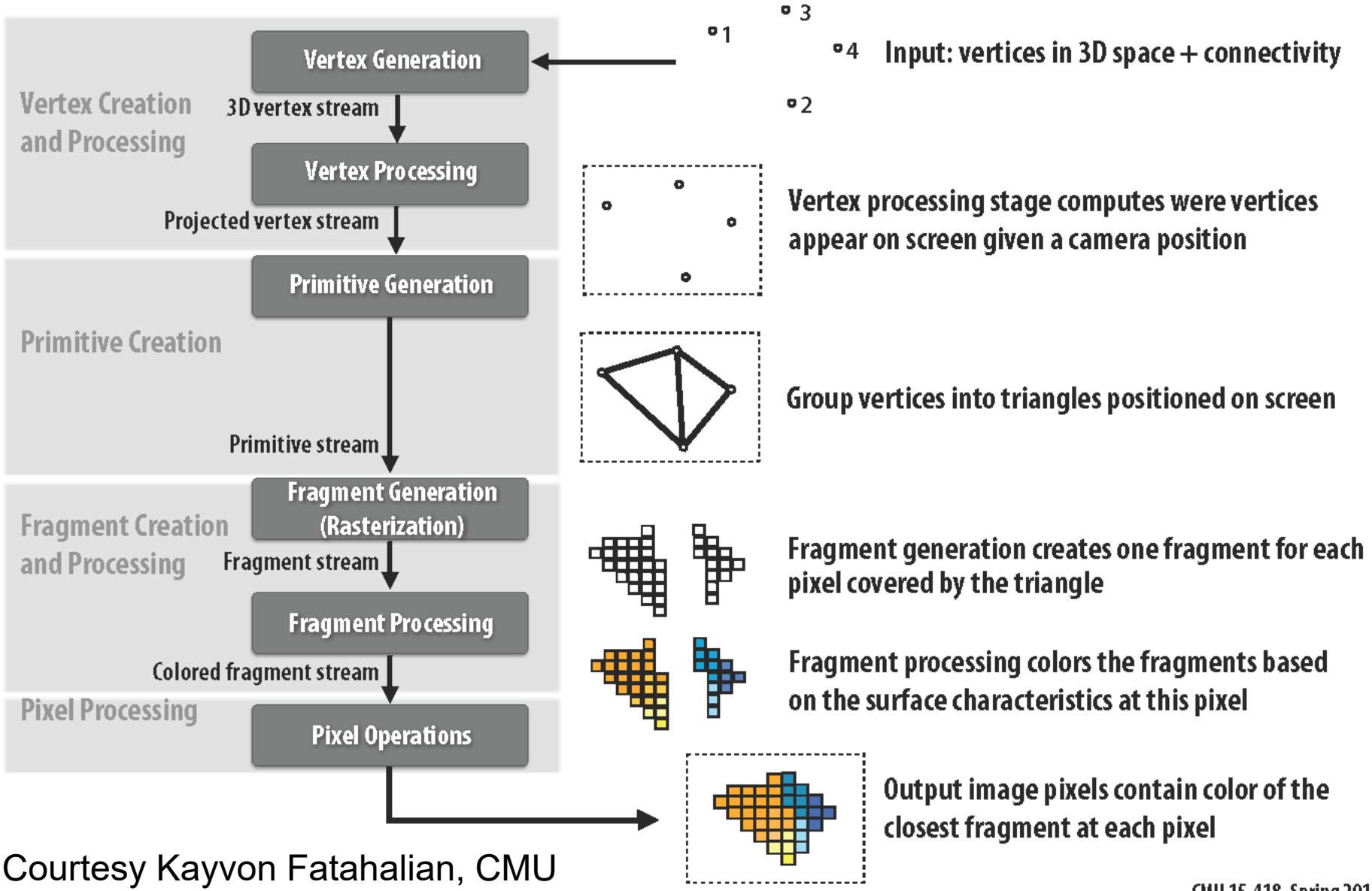
Fragments

Pixels

ROPs = raster operations
(render output units)

Graphics pipeline architecture

Performs operations on vertices, triangles, fragments, and pixels

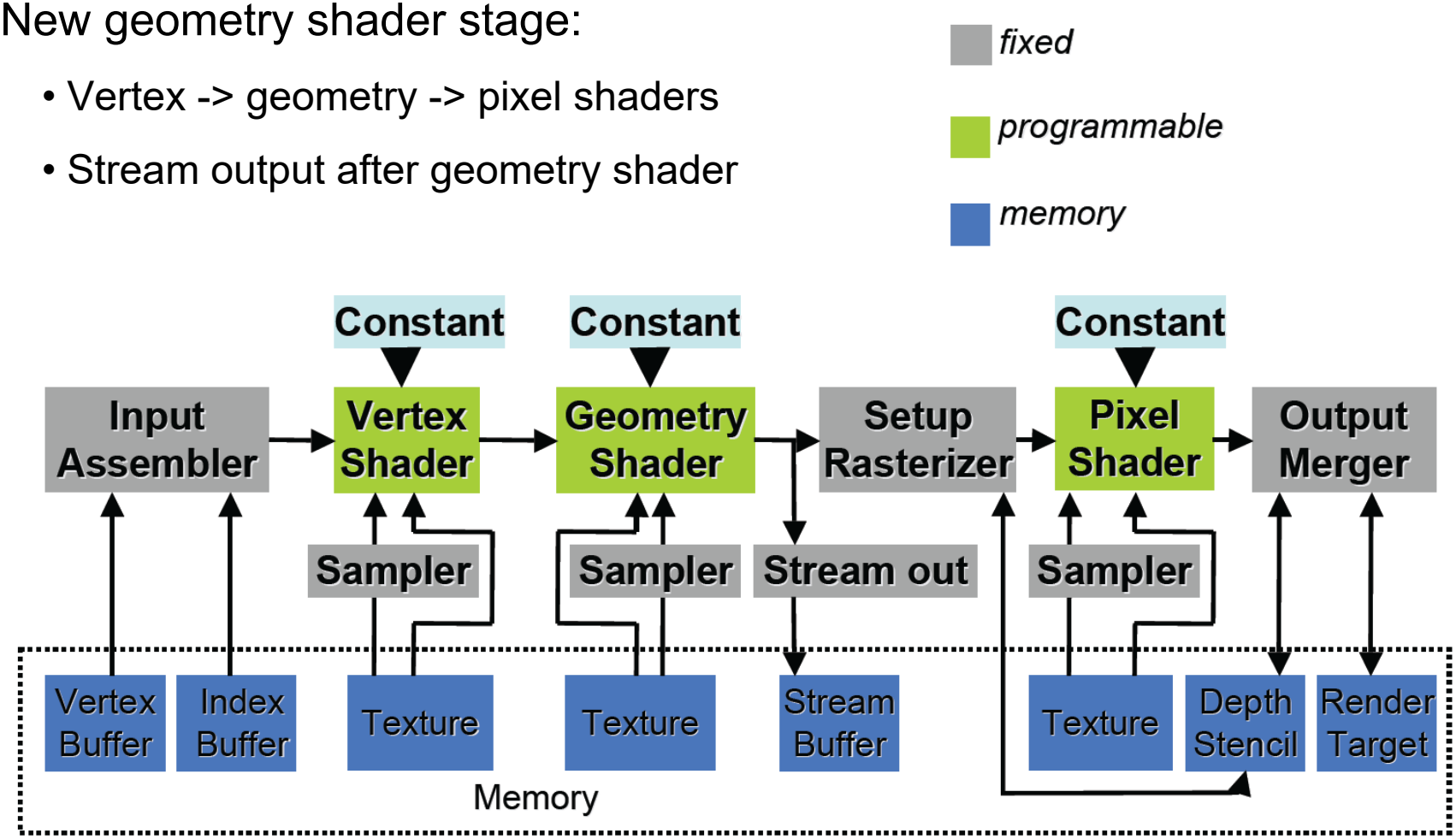


Courtesy Kayvon Fatahalian, CMU

Direct3D 10 Pipeline (~OpenGL 3.2)

New geometry shader stage:

- Vertex -> geometry -> pixel shaders
- Stream output after geometry shader



Courtesy David Blythe, Microsoft

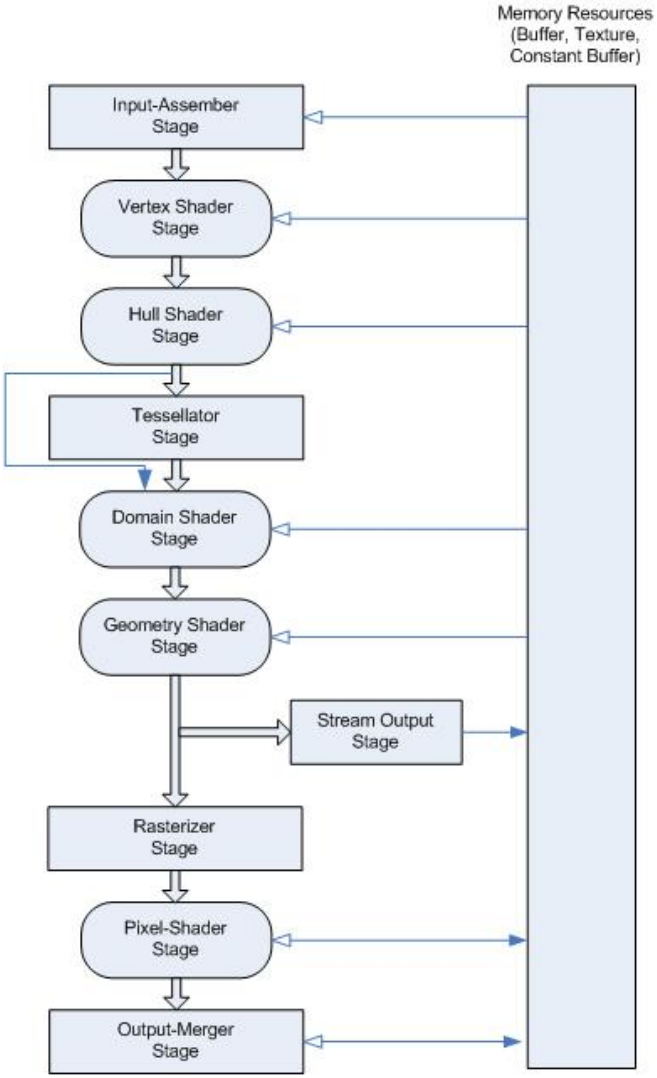
Direct3D 11 Pipeline (~OpenGL 4.x)

New tessellation stages

- Hull shader
  (OpenGL: *tessellation control*)
- Tessellator
  (OpenGL: *tessellation primitive generator*)
- Domain shader
  (OpenGL: *tessellation evaluation*)

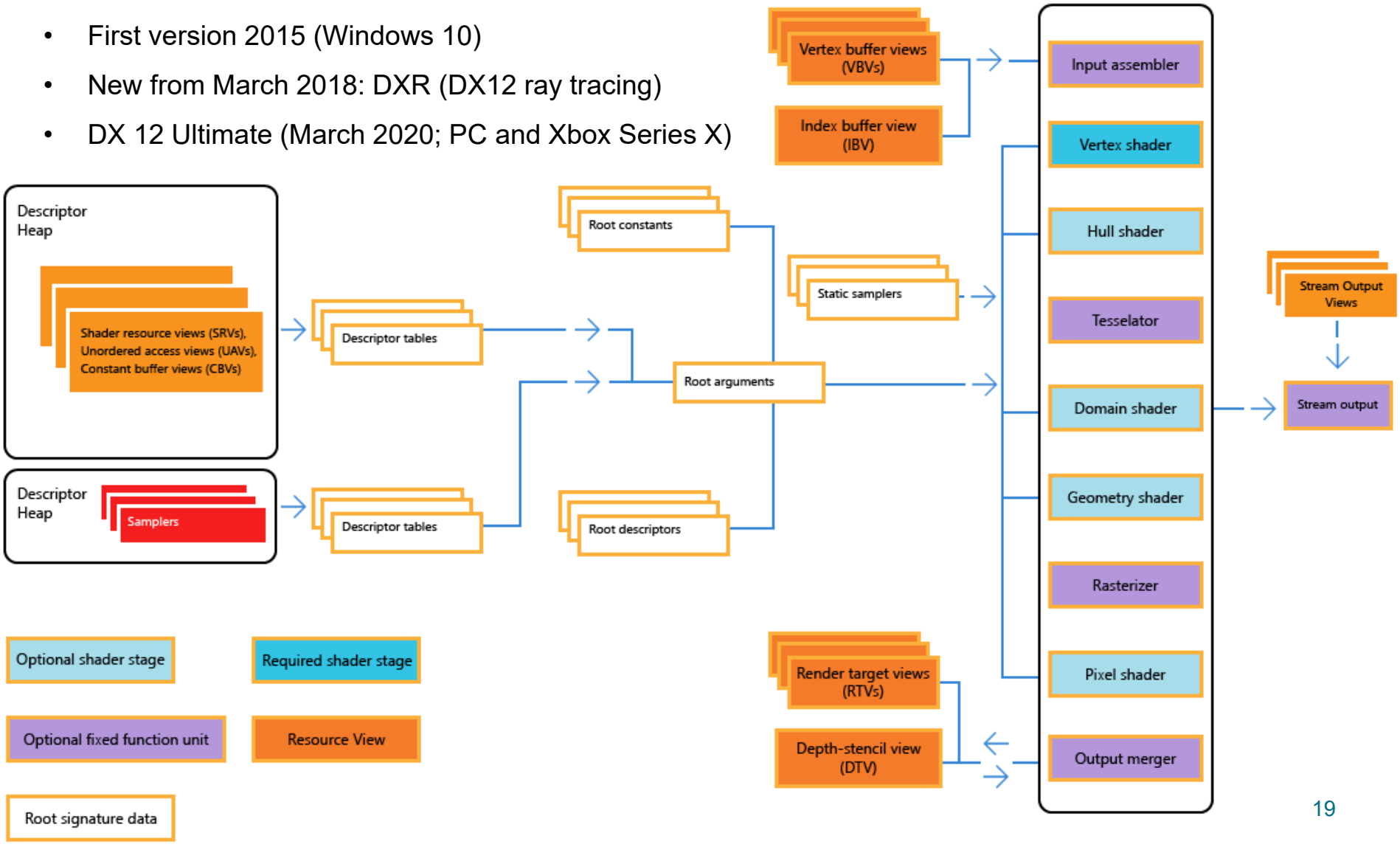
Outside this pipeline

- Compute shader
- (Ray tracing cores, D3D 12)
- (Mesh shader pipeline, D3D 12.2)



Direct3D 12 Traditional Geometry Pipeline

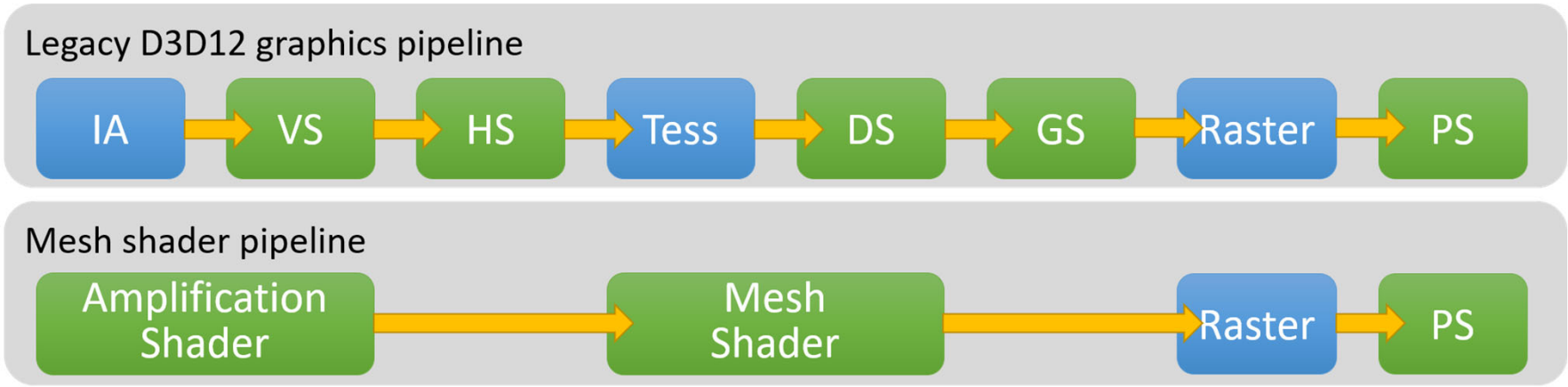
- First version 2015 (Windows 10)
- New from March 2018: DXR (DX12 ray tracing)
- DX 12 Ultimate (March 2020; PC and Xbox Series X)



Direct3D 12 Mesh Shader Pipeline

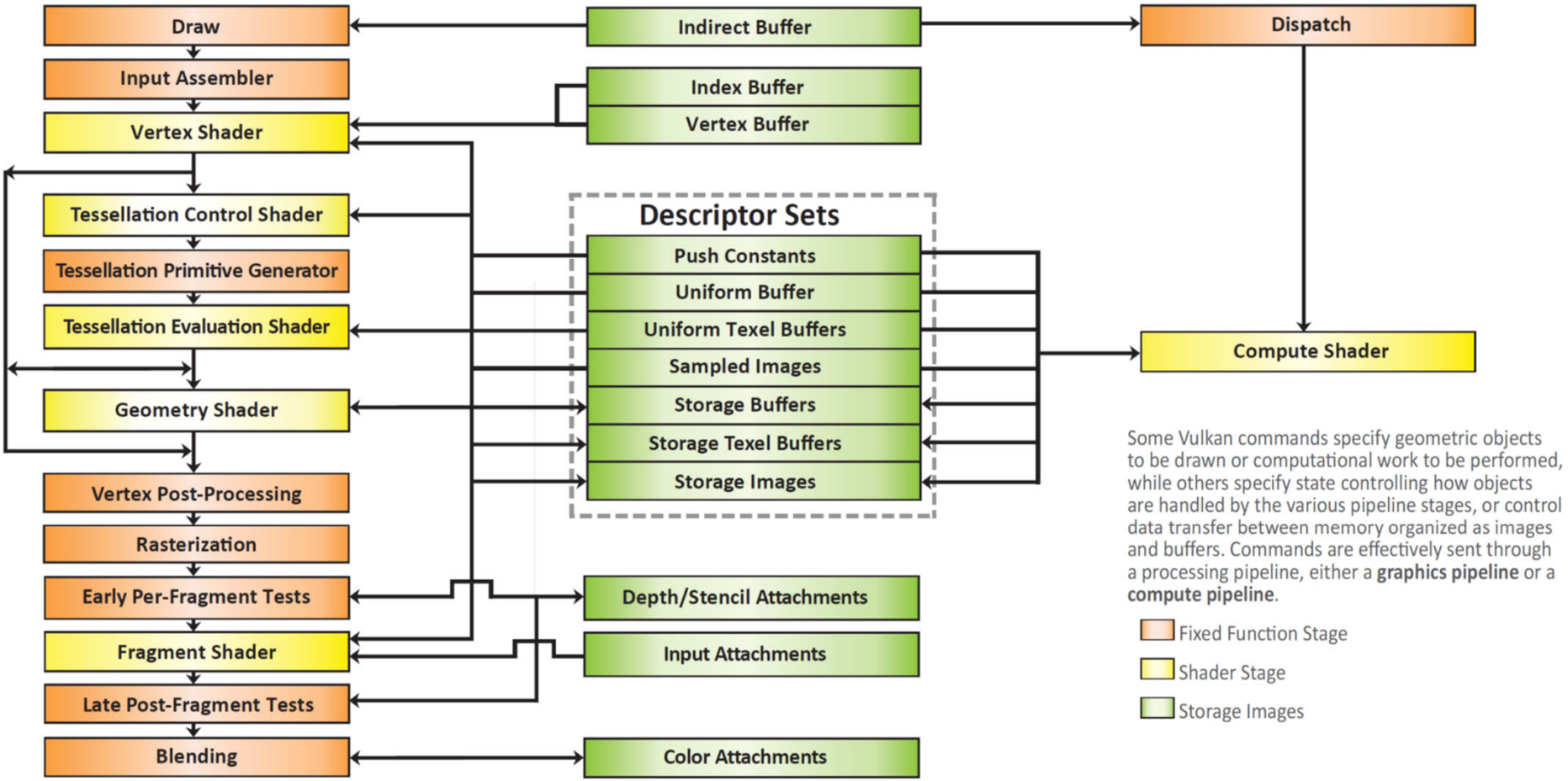
Reinventing the Geometry Pipeline

- Mesh and amplification shaders: new high-performance geometry pipeline based on compute shaders (DX 12 Ultimate / feature level 12.2)
- Compute shader-style replacement of IA/VS/HS/Tess/DS/GS



See talk by Shawn Hargreaves: `https://www.youtube.com/watch?v=CFXKTXtil34`

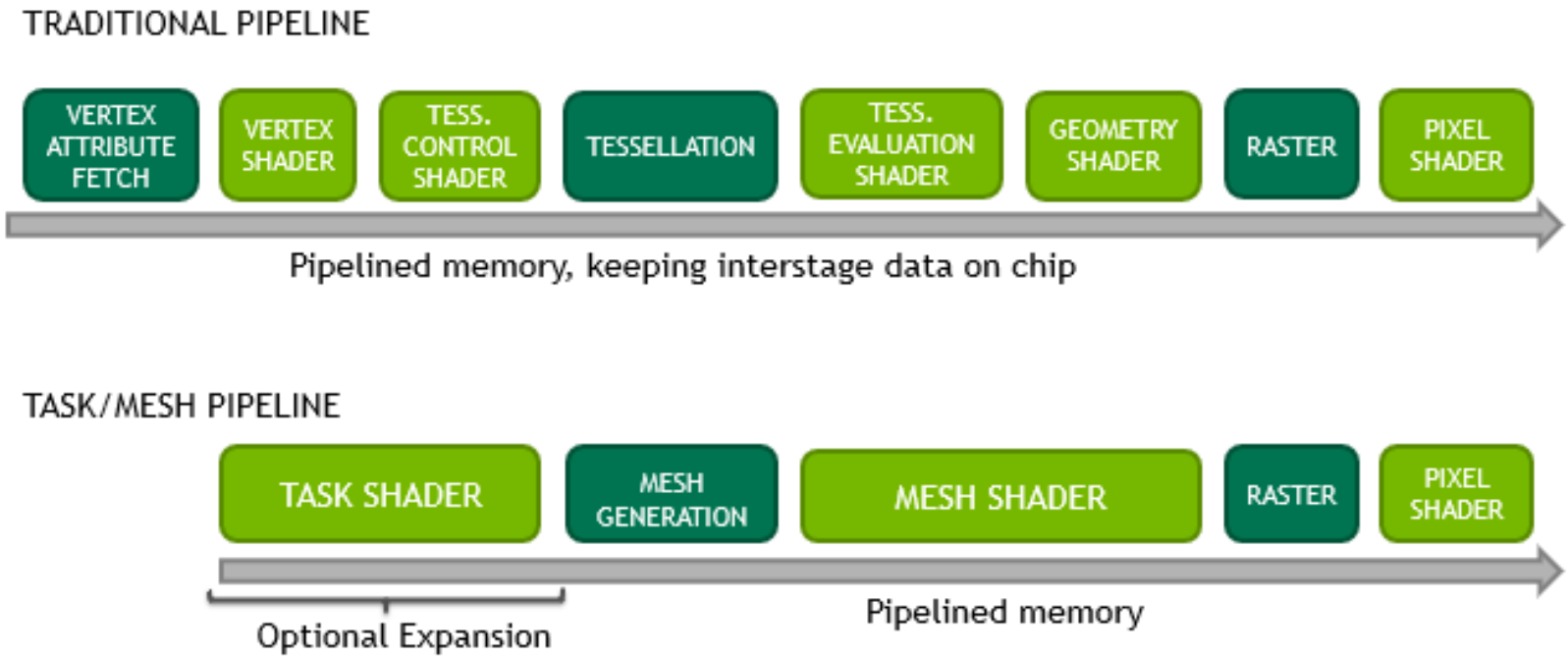
Vulkan (1.3)


Draw
Input Assembler
Vertex Shader
Tessellation Control Shader
Tessellation Primitive Generator
Tessellation Evaluation Shader
Geometry Shader
Vertex Post-Processing
Rasterization
Early Per-Fragment Tests
Fragment Shader
Late Post-Fragment Tests
Blending
Indirect Buffer
Index Buffer
Vertex Buffer
Descriptor Sets
Push Constants
Uniform Buffer
Uniform Texel Buffers
Sampled Images
Storage Buffers
Storage Texel Buffers
Storage Images
Depth/Stencil Attachments
Input Attachments
Color Attachments
Dispatch
Compute Shader
Some Vulkan commands specify geometric objects to be drawn or computational work to be performed, while others specify state controlling how objects are handled by the various pipeline stages, or control data transfer between memory organized as images and buffers. Commands are effectively sent through a processing pipeline, either a graphics pipeline or a compute pipeline.
Fixed Function Stage
Shader Stage
Storage Images

Vulkan (1.3)

- Mesh and task shaders: new high-performance geometry pipeline based on compute shaders

  (Mesh and task shaders also available as OpenGL 4.5/4.6 extension: `GL_NV_mesh_shader`)




```
vulkan.org
github.com/KhronosGroup/Vulkan-Guide
https://www.khronos.org/blog/mesh-shading-for-vulkan
```

Motivational Examples

Doom (2016)

http://www.adriancourreges.com/blog/2016/09/09/doom-2016-graphics-study/

Doom Eternal

https://simoncoenen.com/blog/programming/graphics/DoomEternalStudy.html

Unreal Engine 5

https://www.unrealengine.com/en-US/unreal-engine-5
https://www.unrealengine.com/en-US/blog/a-first-look-at-unreal-engine-5

Thank you.