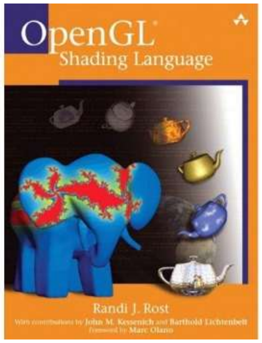
CS 380 - GPU and GPGPU Programming
Lecture 3: GPU Architecture, Pt. 1

Markus Hadwiger, KAUST

King Abdullah University of Science and Technology
KAUST

Reading Assignment #2 (until Sep 15)



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`
- Read:
  `https://en.wikipedia.org/wiki/Instruction_pipelining`
  `https://en.wikipedia.org/wiki/Classic_RISC_pipeline`
- Get an overview of NVIDIA Hopper and Blackwell GPU architectures:
  `https://resources.nvidia.com/en-us-hopper-architecture/nvidia-h100-tensor-c`
  `https://images.nvidia.com/aem-dam/Solutions/geforce/blackwell/`
  `nvidia-rtx-blackwell-gpu-architecture.pdf`

Read (optional):

- GPU Gems 2 book, Chapter 30
  (*The GeForce 6 Series GPU Architecture*)
  `http://download.nvidia.com/developer/GPU_Gems_2/GPU_Gems2_ch30.pdf`

NVIDIA Architectures (since first CUDA GPU)

Tesla [CC 1.x]: 2007-2009

- G80, G9x: 2007 (Geforce 8800, ...)
  GT200: 2008/2009 (GTX 280, ...)

Fermi [CC 2.x]: 2010 (2011, 2012, 2013, …)

- GF100, ... (GTX 480, ...)
  GF104, ... (GTX 460, ...)
  GF110, ... (GTX 580, ...)

Kepler [CC 3.x]: 2012 (2013, 2014, 2016, …)

- GK104, ... (GTX 680, ...)
  GK110, ... (GTX 780, GTX Titan, ...)

Maxwell [CC 5.x]: 2015

- GM107, ... (GTX 750Ti, ...); [Nintendo Switch]
  GM204, ... (GTX 980, Titan X, ...)

Pascal [CC 6.x]: 2016 (2017, 2018, 2021, 2022, …)

- GP100 (Tesla P100, ...)
- GP10x: x=2,4,6,7,8, ...
  (GTX 1060, 1070, 1080, Titan X *Pascal*, Titan Xp, ...)

Volta [CC 7.0, 7.2]: 2017/2018

- GV100, ...
  (Tesla V100, Titan V, Quadro GV100, ...)

Turing [CC 7.5]: 2018/2019

- TU102, TU104, TU106, TU116, TU117, ...
  (Titan RTX, RTX 2070, 2080 (Ti), GTX 1650, 1660, ...)

Ampere [CC 8.0, 8.6, 8.7]: 2020

- GA100, GA102, GA104, GA106, ...; [Nintendo Switch 2]
  (A100, RTX 3070, 3080, 3090 (Ti), RTX A6000, ...)

Hopper [CC 9.0], Ada Lovelace [CC 8.9]: 2022/23

- GH100, AD102, AD103, AD104, ...
  (H100, L40, RTX 4080 (12/16 GB), 4090, RTX 6000, ...)

Blackwell [CC 10.0, 10.1, 10.3, 12.0, 12.1] : 2024/2025

- GB100/102, GB200/202/203/205/206/207, ...
  (RTX 5080/5090, GB200 NVL72, HGX B100/200, ...)

see `https://en.wikipedia.org/wiki/List_of_Nvidia_graphics_processing_units`
and `https://en.wikipedia.org/wiki/CUDA`

Comprehensive Overviews and Specs

Wikipedia has many comprehensive lists of architectures and specs:

https://en.wikipedia.org/wiki/List_of_Nvidia_graphics_processing_units

https://en.wikipedia.org/wiki/List_of_AMD_graphics_processing_units

Overview of compute capability (CC) vs. GPU; features (precision, etc.):

https://en.wikipedia.org/wiki/CUDA

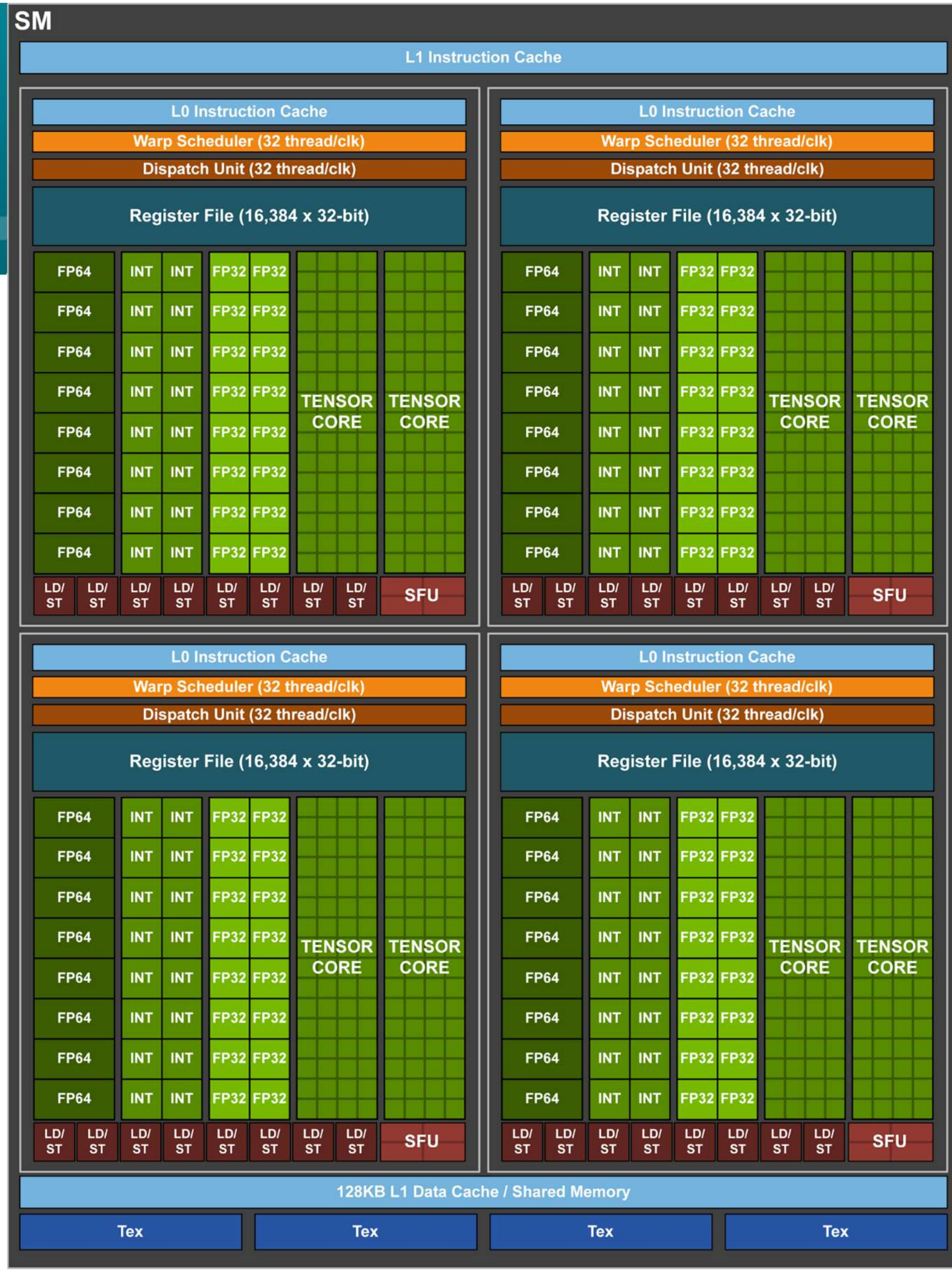
NVIDIA Volta SM

CC 7.0 SM (Multiprocessor)

- 64 FP32 + 64 INT32 cores
- 32 FP64 cores
- 32 LD/ST units; 16 SFUs
- 8 tensor cores
  (FP16/FP32 mixed-precision)

4 partitions inside SM

- 16 FP32 + 16 INT32 cores each
- 8 FP64 cores each
- 8 LD/ST units; 4 SFUs each
- 2x 1st gen tensor cores;
  16x (64b) dot product units / core
- Each has: warp scheduler,
  dispatch unit, register file



Example for “Special Cores”: Tensor Cores

Mixed-precision, fast matrix-matrix multiply and accumulate

$$\underbrace{D}_{\text{FP16 or FP32}} = \underbrace{\begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix}}_{\text{FP16}} \underbrace{\begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix}}_{\text{FP16}} + \underbrace{\begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix}}_{\text{FP16 or FP32}}$$

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

NVIDIA Volta Architecture (2017/2018)

Total chip capacity on Tesla V100 (GV100 architecture)

- 80 SMs
  - 64 FP32 cores / SM = 5,120 FP32 cores in total
  - 64 INT32 cores / SM = 5,120 INT32 cores in total
  - 32 FP64 cores / SM = 2,560 FP64 cores in total
  - 4 FP16/FP32 mixed-prec. tensor cores = 650 tensor cores in total
- 40 TPCs (2 SMs per TPC)
- 6 GPCs

Maximum capacity would be 84 SMs and 42 TPCs

NVIDIA Hopper SM

CC 9.0 SM (GH100 Multiprocessor)

- 128 FP32 + 64 INT32 cores
- 64 FP64 cores
- 4x 4th gen tensor cores
- ++ thread block clusters, DPX insts., FP8, TMA

4 partitions inside SM

- 32 FP32 + 16 INT32 cores
- 16 FP64 cores
- 8x LD/ST units each
- 1x 4th gen tensor core;
  32x (256b) dot product units / core
- Each has: warp scheduler,
  dispatch unit, 16K register file

NVIDIA Hopper GH100 Architecture (2022)

GH 100 (H100 Tensor Core GPU)

Full GPU: 144 SMs in 8 GPCs, 72 TPCs, (18,432 FP32 cores)

NVIDIA Hopper GH100 Architecture (2022)

GH 100 (H100 Tensor Core GPU)   Full GPU: 144 SMs in 8 GPCs, 72 TPCs

- 64K 32-bit registers / SM = 256 KB register storage per SM
- 256 KB shared memory / L1 per SM

For 144 SMs on full GPU *[SXM5: 132; PCIe: 114]*

- 36 MB register storage, 36 MB shared mem / L1 storage =
  72 MB context+"shared context" storage !
- L2 cache size on H100: 50 MB
- 18,432 FP32 cores (128 FP32 cores per SM) *[SXM5: 16,896]*; 576 tensor cores
- 294,912 max threads in flight (max warps / SM = 64) *[SXM5: 270,336]*

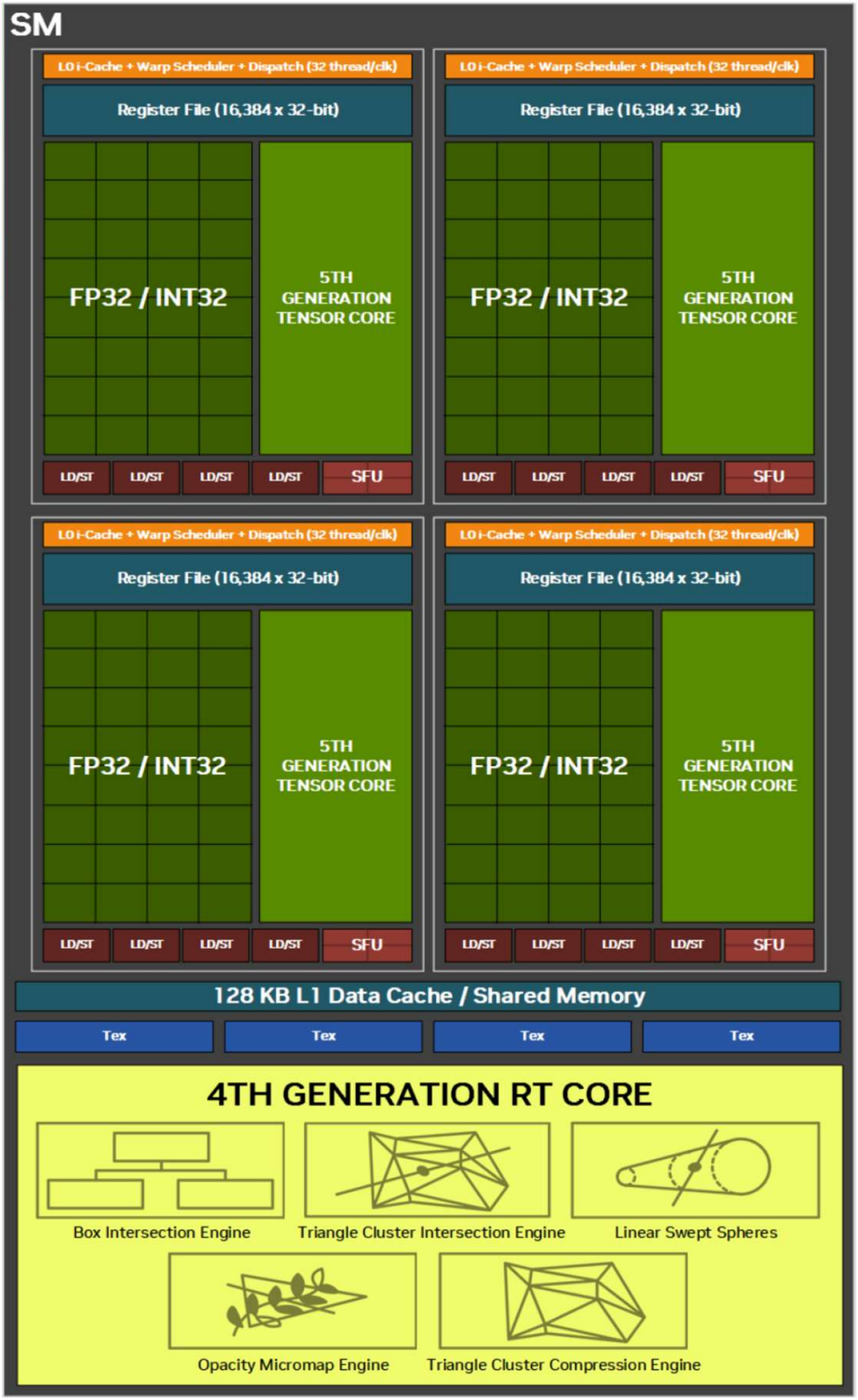
NVIDIA Blackwell SM

CC 12.0 SM (GB 202 Multiprocessor)

- 128 FP32/INT32 cores
- 2 FP64 cores
- 4x 5th gen tensor cores
- ++ thread block clusters, DPX insts., FP8, NVFP4, TMA

4 partitions inside SM

- 32 FP32/INT32 cores
- 4x LD/ST units each
- 1x 5th gen tensor core
- Each has: warp scheduler, dispatch unit, 16K register file



NVIDIA Blackwell GB202 Architecture (2025)

GB 202 (RTX GPU)

Full GPU: 192 SMs in 12 GPCs, 96 TPCs, (24,576 FP32 cores)

NVIDIA Blackwell GB202 Architecture (2025)

GB 202 (RTX GPU)

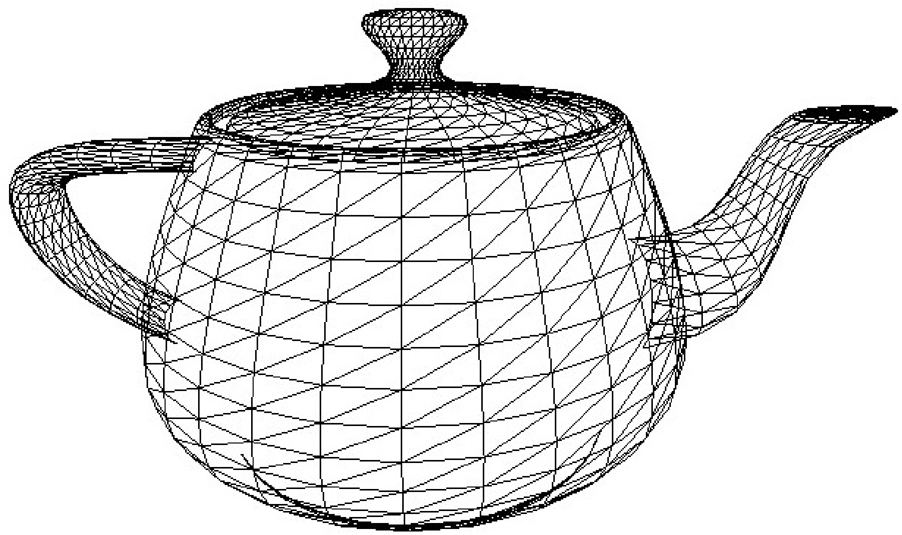
Full GPU: 192 SMs in 12 GPCs, 96 TPCs
(RTX 5090: 170 SMs, 11 GPCs, 85 TPCs)

- 64K 32-bit registers / SM = 256 KB register storage per SM
- 128 KB shared memory / L1 per SM

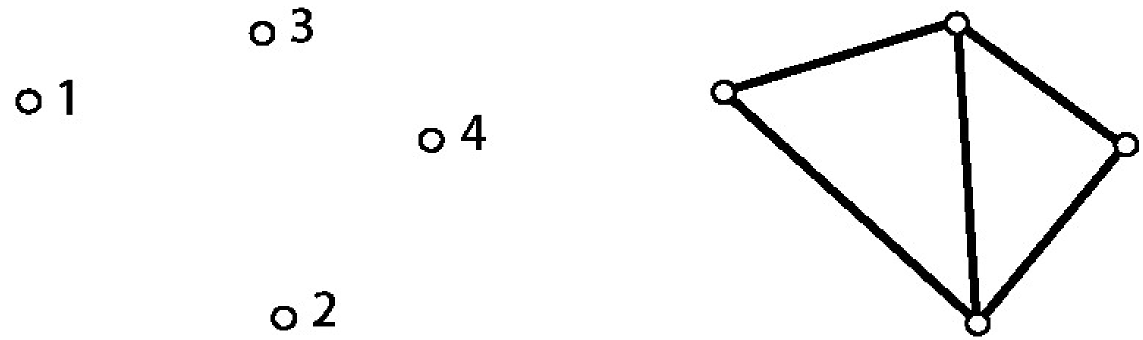
For 192 SMs on full GPU (RTX PRO 6000: 12 GPCs, 94 TPCs, 188 SMs = 24,064 FP32 cores)

- 48 MB register storage, 24 MB shared mem / L1 storage =
  72 MB context+"shared context" storage !
- L2 cache size 128 MB (e.g., RTX PRO 6000) (RTX 5090: 96 MB, RTX 5080: 64 MB)
- 24,576 FP32 cores (128 FP32 cores per SM), 768 tensor cores
  (RTX PRO 6000: 752 tensor cores; RTX 5090: 170 SMs = 21,760 FP32 cores, 680 tensor cores)
- 294,912 max threads in flight (max warps / SM = 48)

Real-time graphics primitives (entities)



Represent surface as a 3D triangle mesh

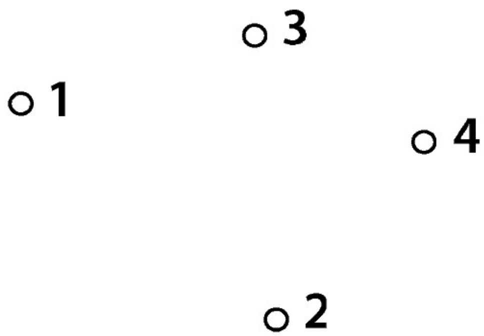


Vertices

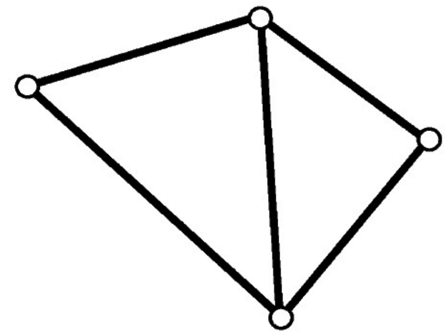
Primitives
(e.g., triangles, points, lines)

Courtesy Kayvon Fatahalian, CMU

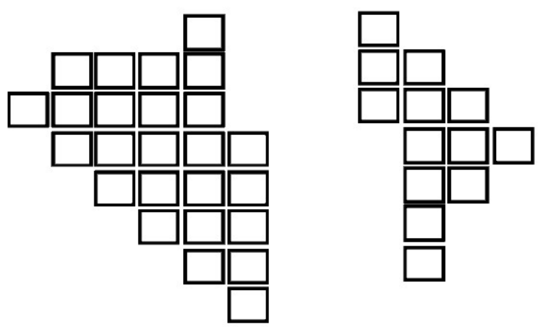
Real-time graphics primitives (entities)



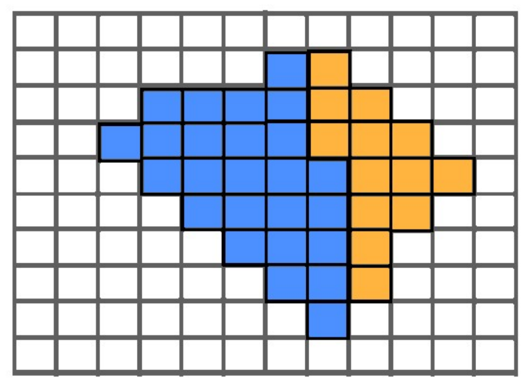
Vertices



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



Fragments

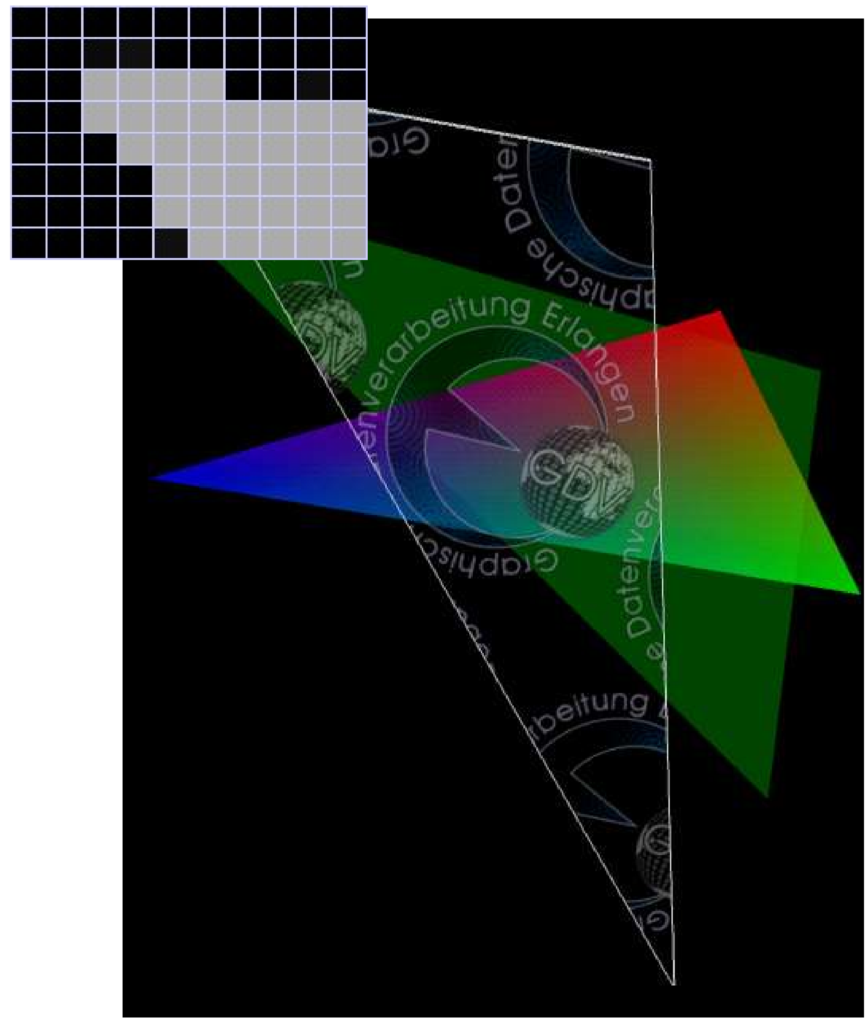


Pixels (in an image)

Courtesy Kayvon Fatahalian, CMU

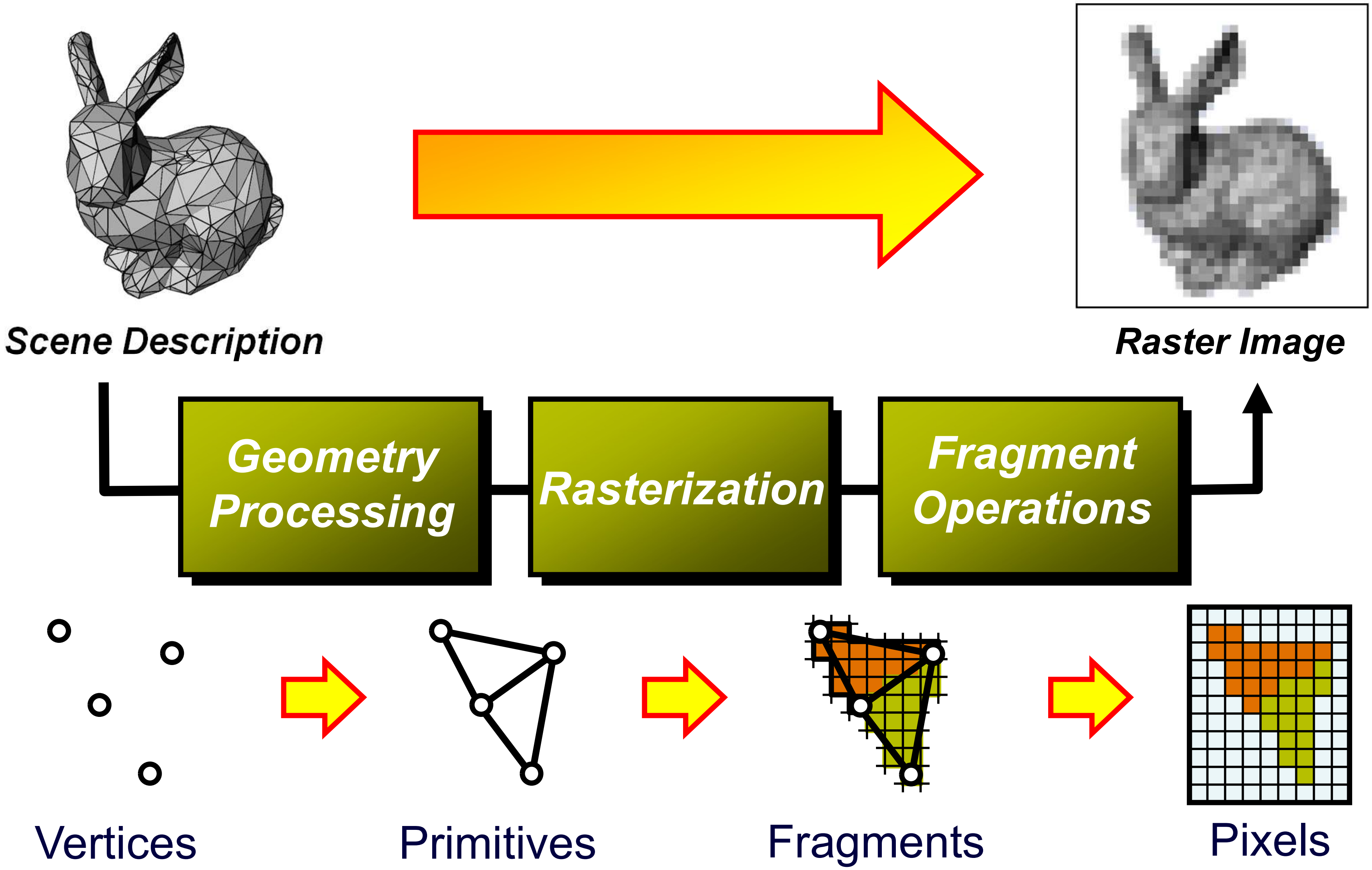
What can the hardware do?

- **Rasterization**
  - Decomposition into fragments
  - Interpolation of color
  - Texturing
    - Interpolation/Filtering
    - Fragment Shading

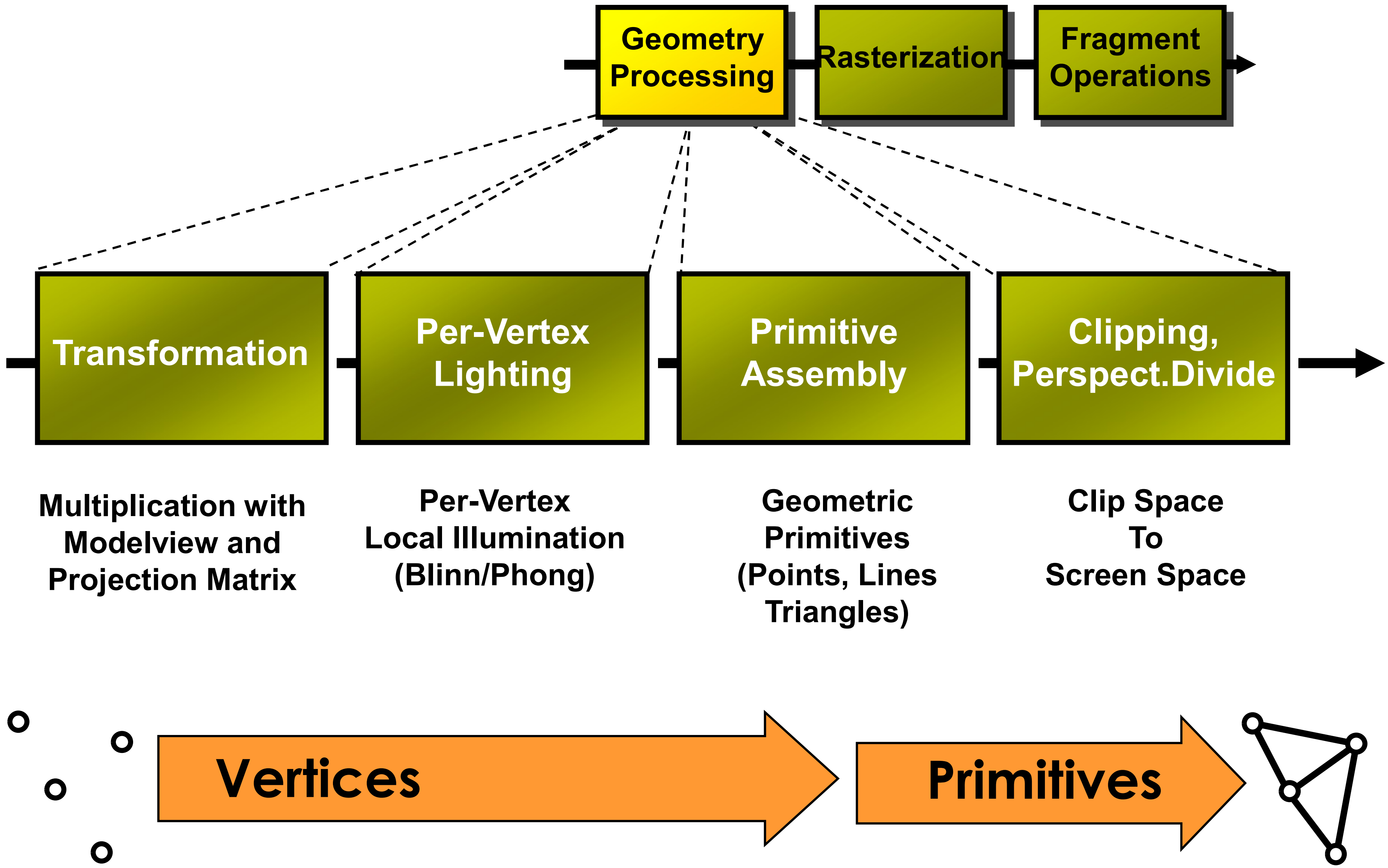


- **Fragment Operations**
  - Depth Test (Z-Test)
  - Alpha Blending (Compositing)

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

Geometry Processing → Rasterization → **Fragment Operations**

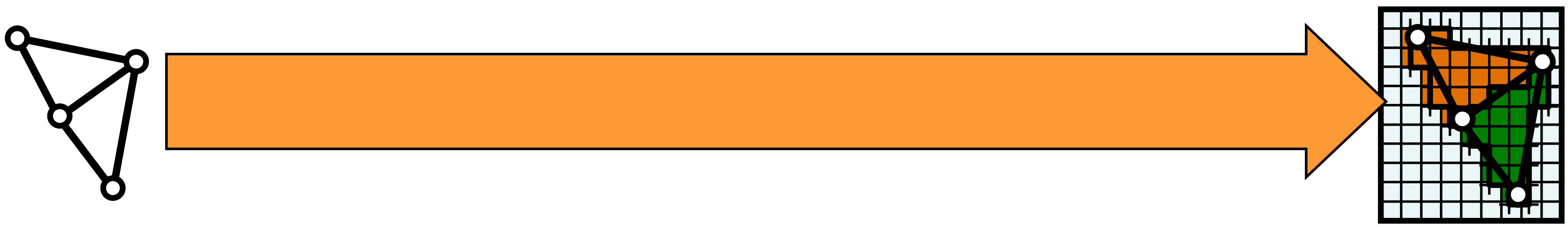
Alpha Test → **Stencil Test** → **Depth Test** → **Alpha Blending** →

Alpha Test: Discard all fragments within a certain alpha range

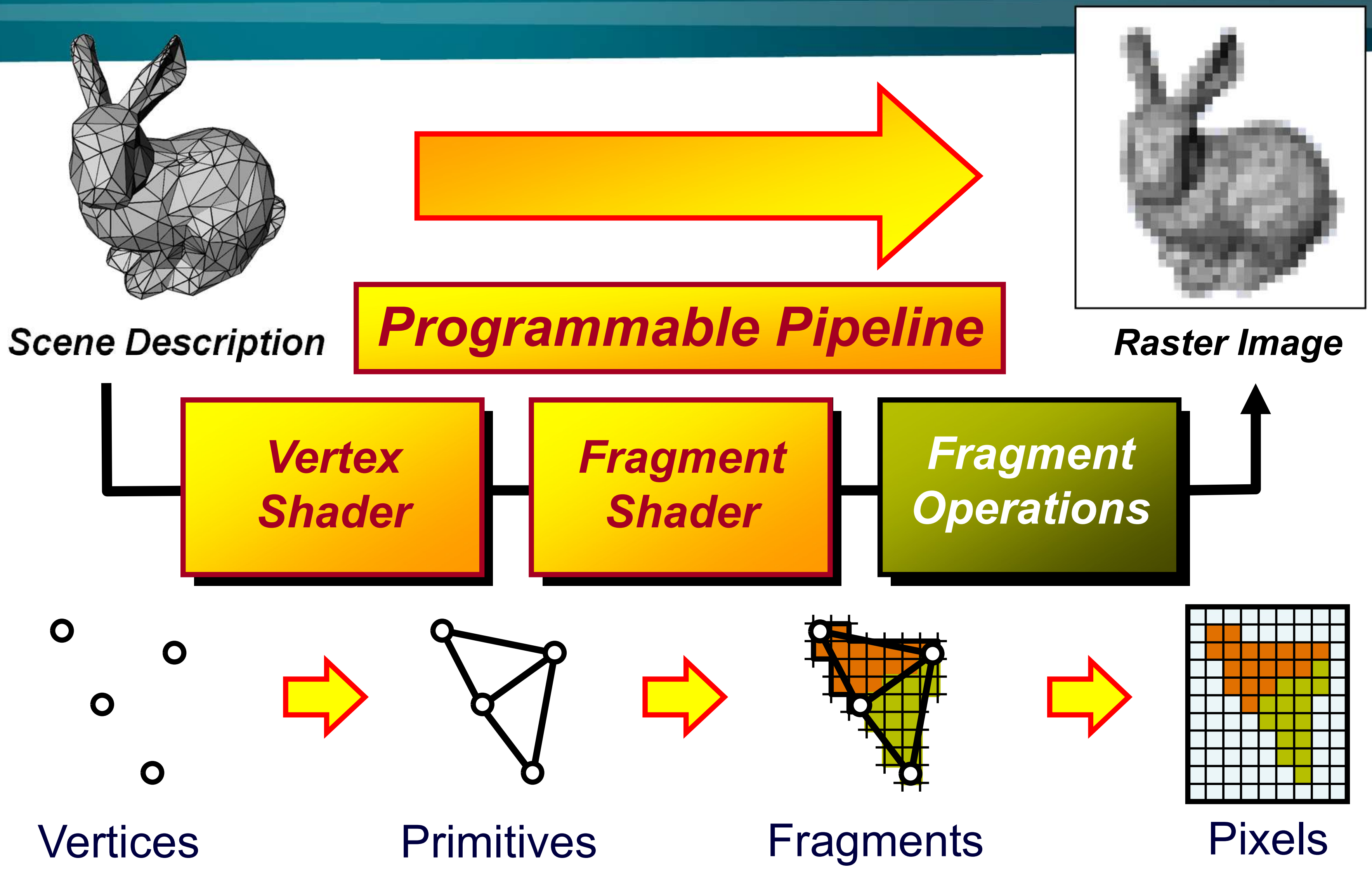
Stencil Test: Discard a fragment if the stencil buffer is set

Depth Test: Discard all occluded fragments

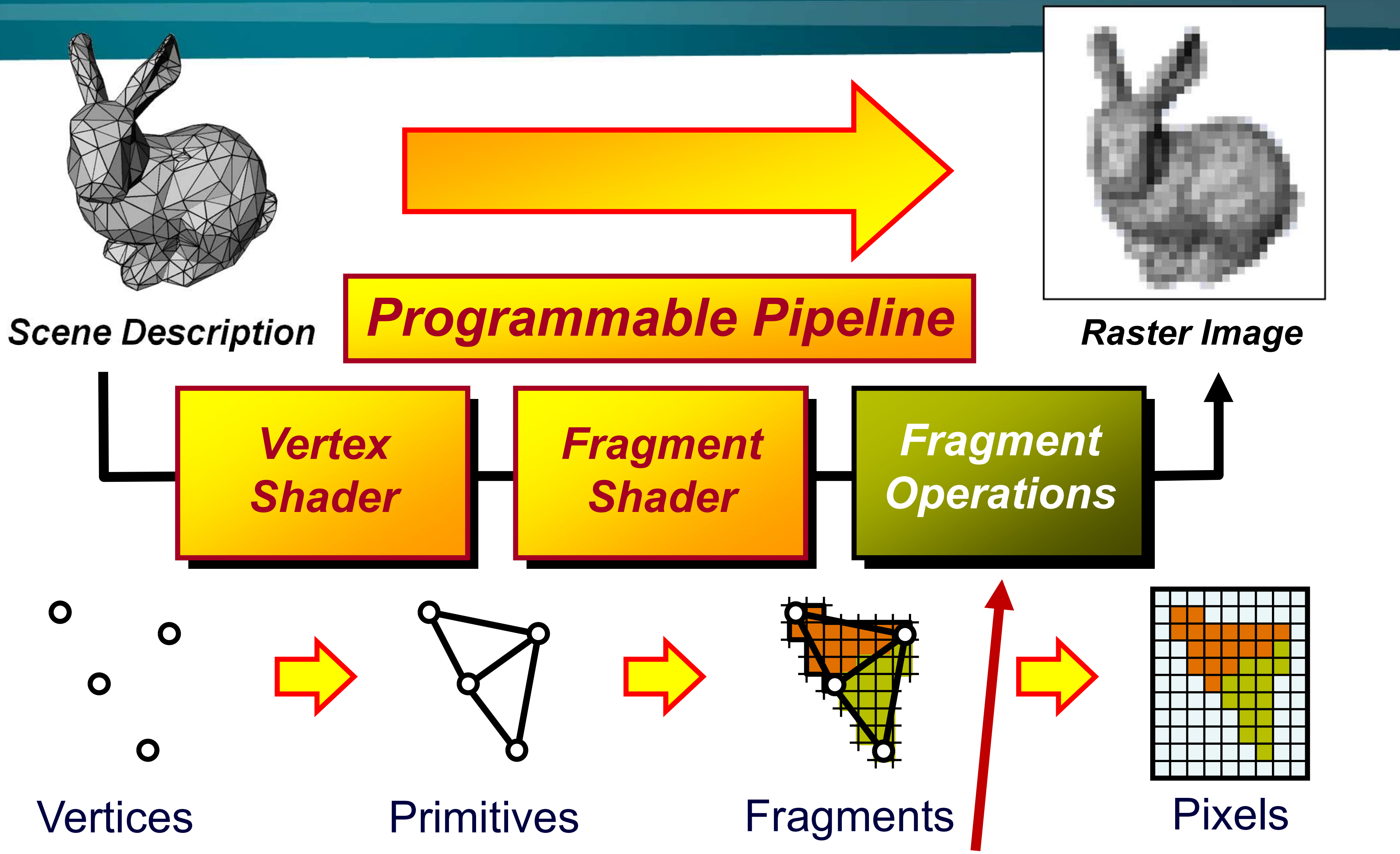
Alpha Blending: Combination of primary color with texture color



Graphics Pipeline



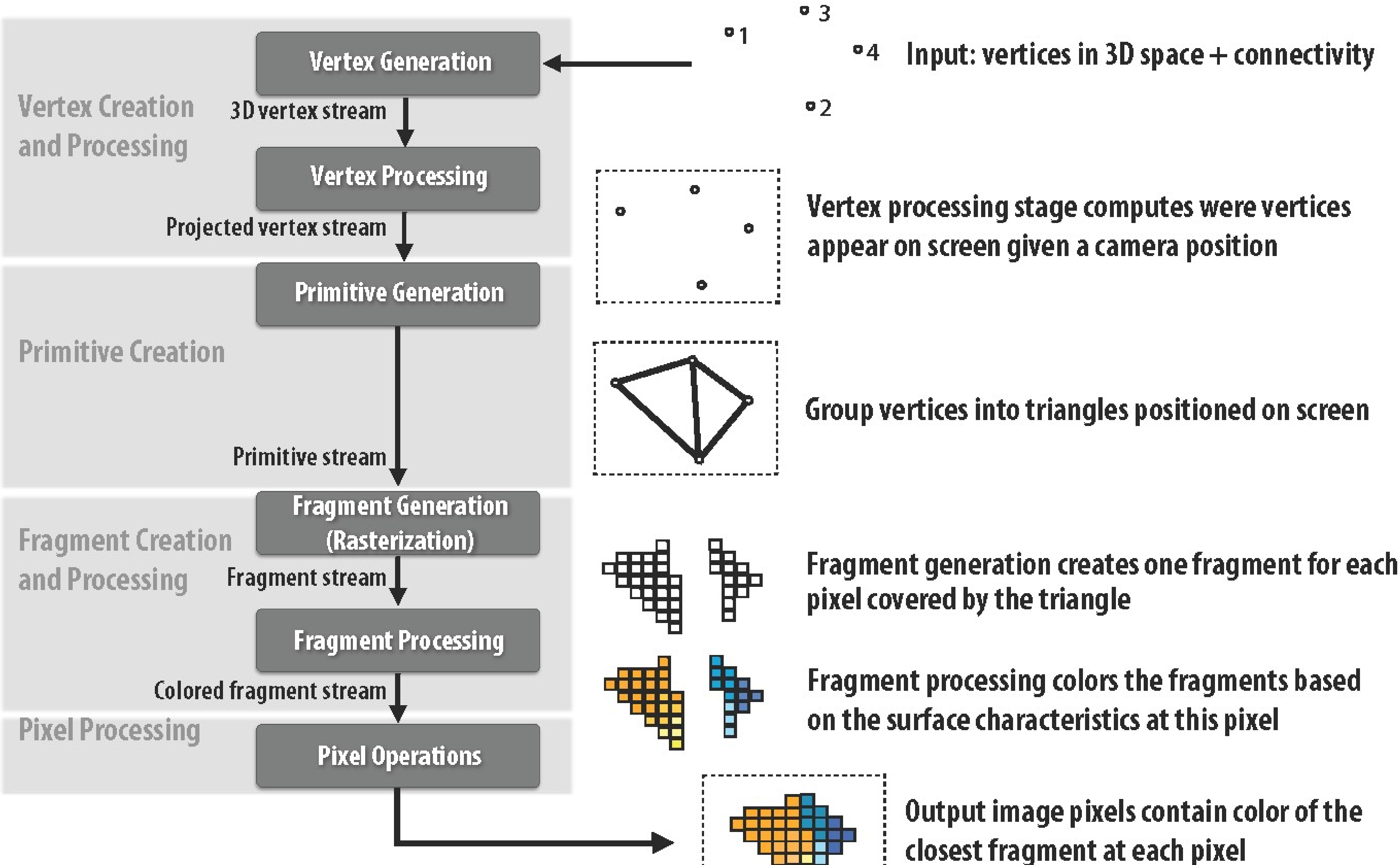
Graphics Pipeline



ROPs = raster operations
(render output units)

Graphics pipeline architecture

Performs operations on vertices, triangles, fragments, and pixels

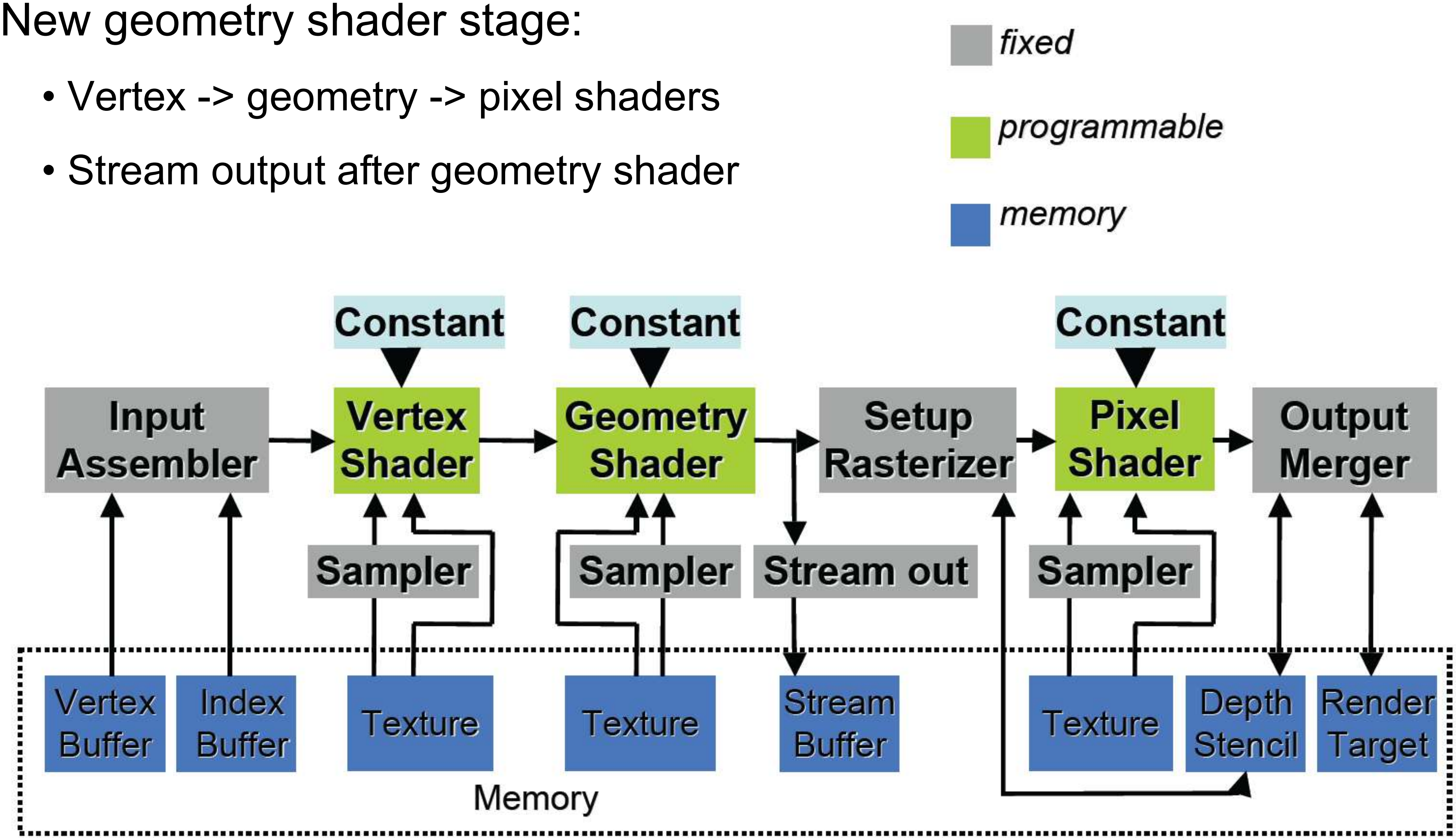


Courtesy Kayvon Fatahalian, CMU

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

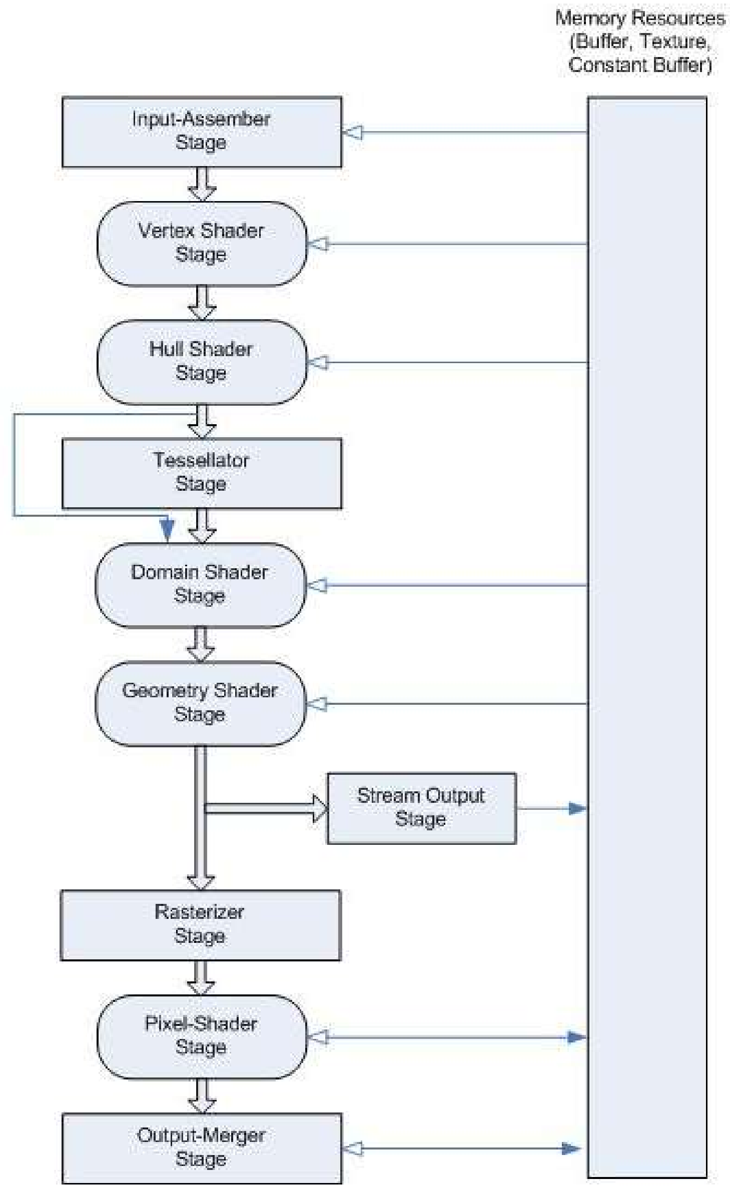
  (OpenGL: *tessellation primitive generator*)

- Domain shader

  (OpenGL: *tessellation evaluation*)

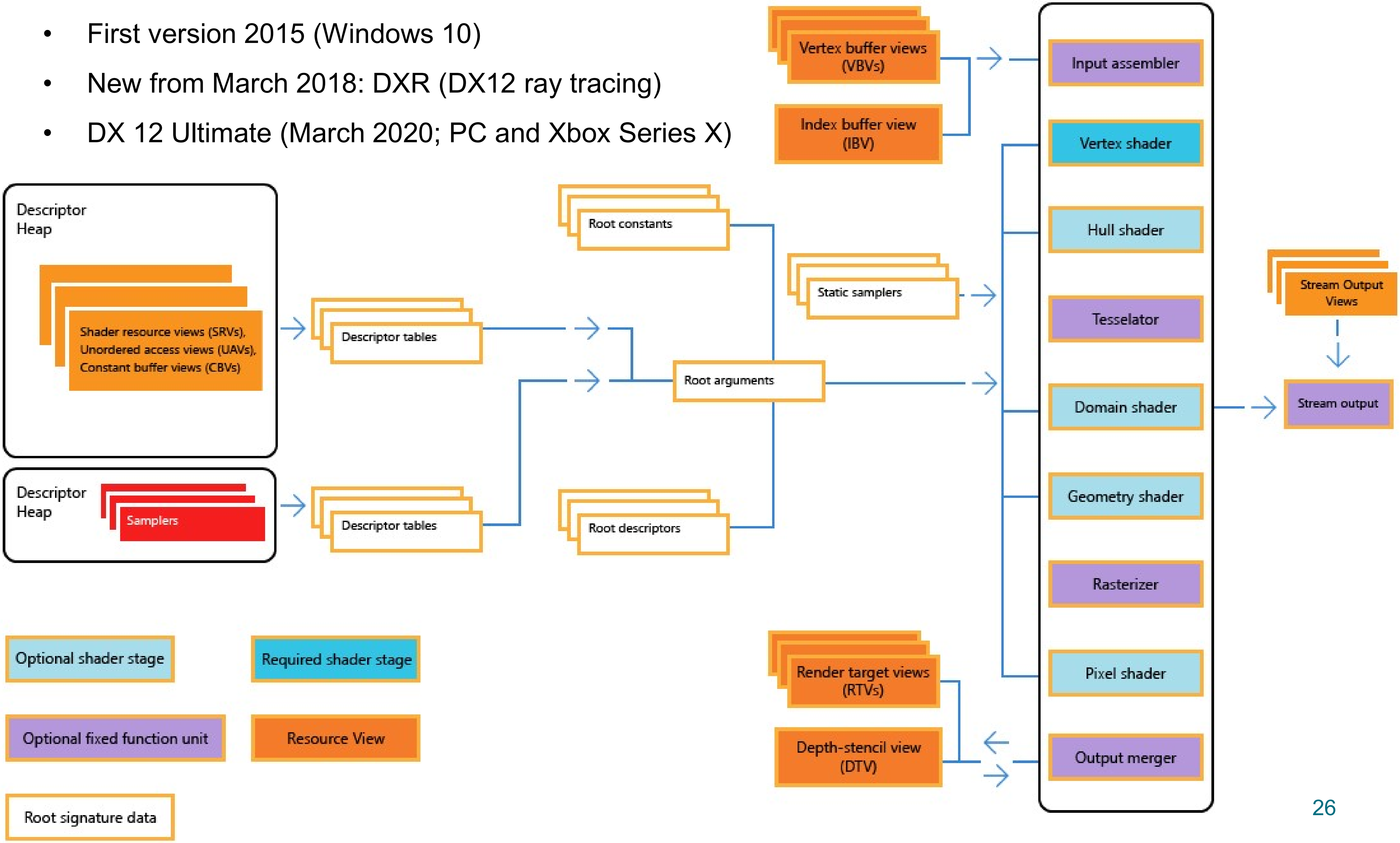
Outside this pipeline

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



Direct3D 12 Traditional Geometry Pipeline

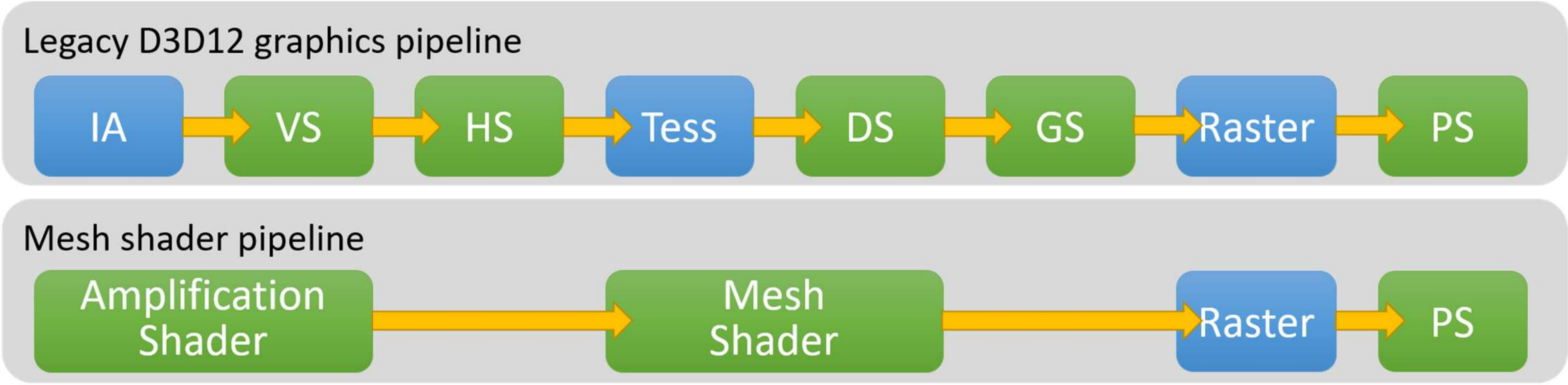
- First version 2015 (Windows 10)
- New from March 2018: DXR (DX12 ray tracing)
- DX 12 Ultimate (March 2020; PC and Xbox Series X)



Direct3D 12 Mesh Shader Pipeline

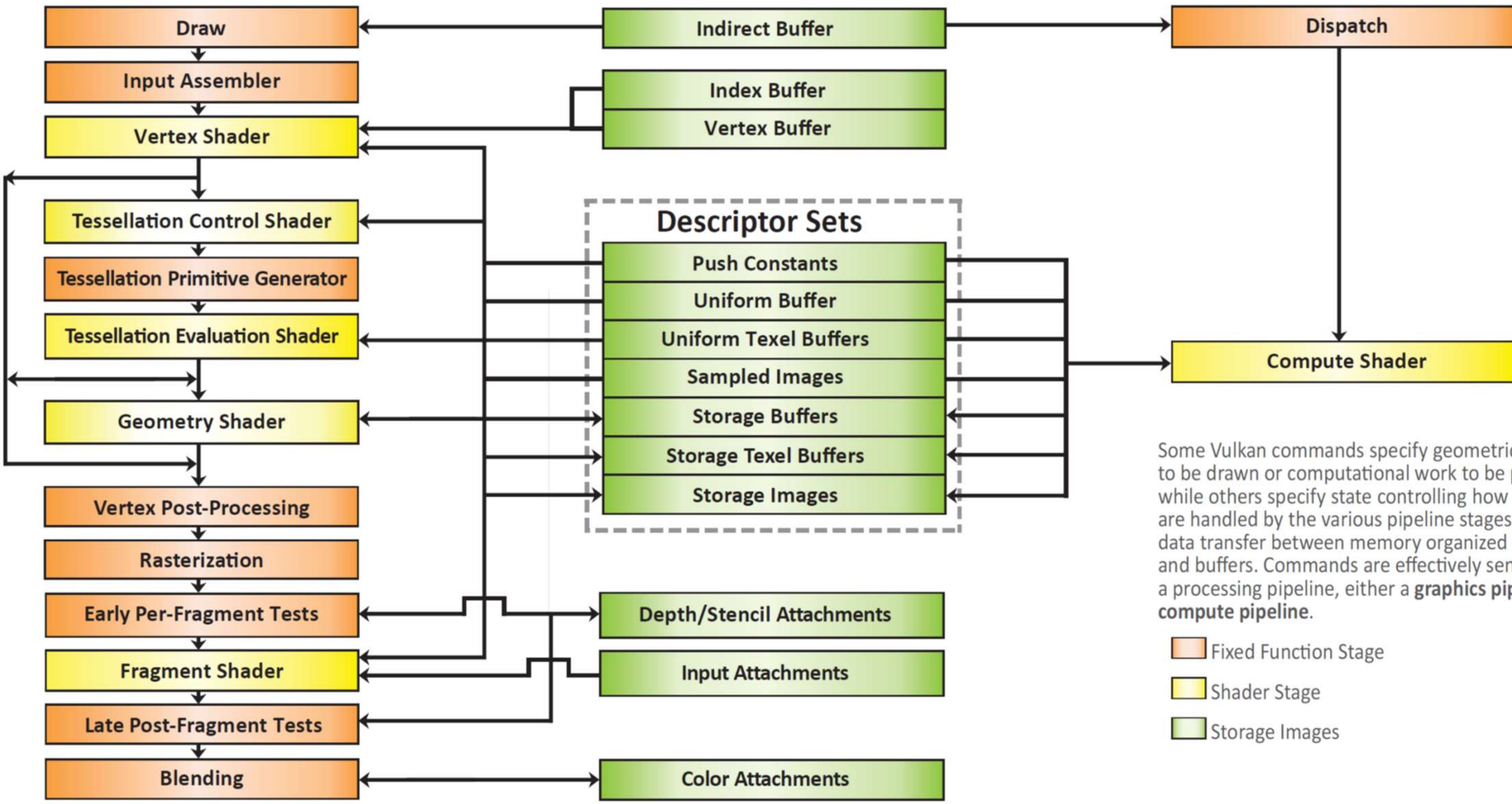
Reinventing the Geometry Pipeline

- 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)

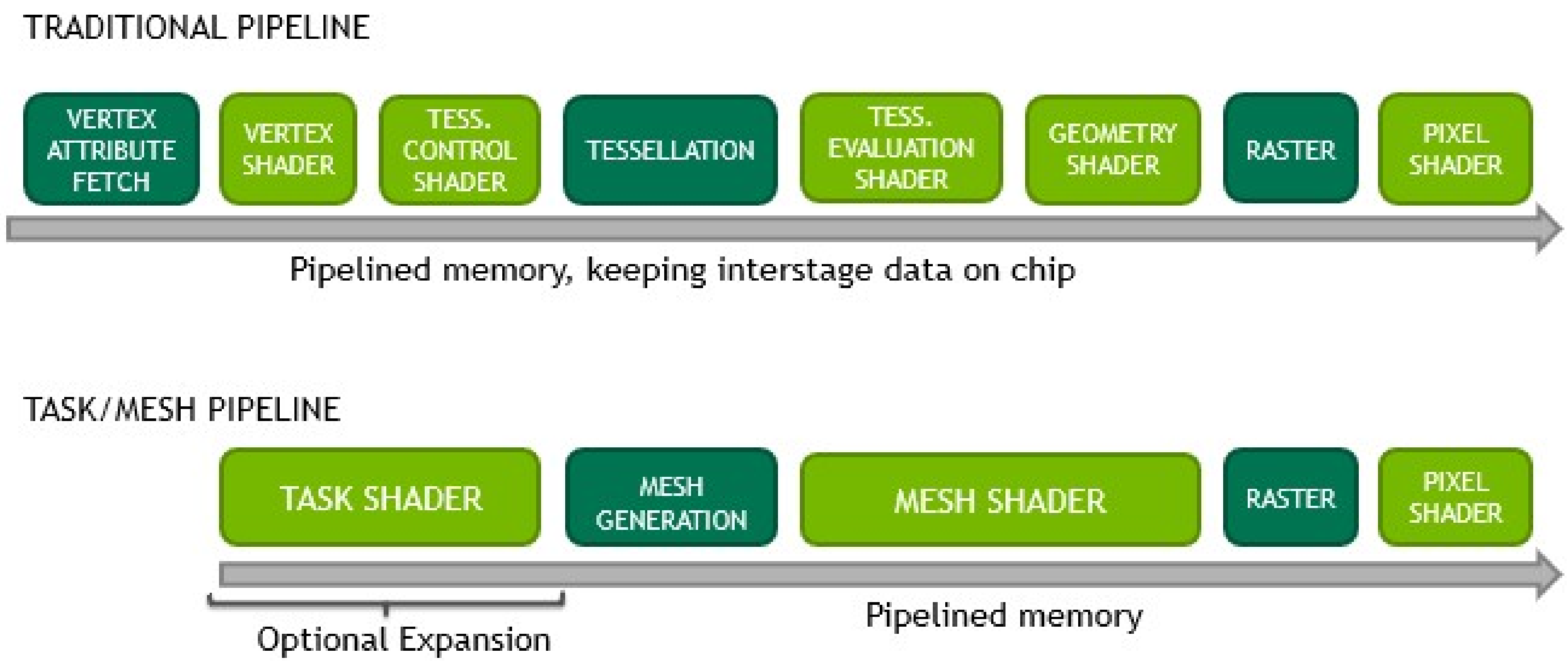


Some Vulkan commands specify geometric objects to be drawn or computational work to be performed, while others specify state controlling how objects are handled by the various pipeline stages, or control data transfer between memory organized as images and buffers. Commands are effectively sent through 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

Thank you.