King Abdullah University of
Science and Technology

KAUST

CS 380 - GPU and GPGPU Programming
Lecture 24: Graphics Pipelines; GPU Texturing, Pt. 1

Markus Hadwiger, KAUST

Reading Assignment #9 (until Nov 4)

Read (required):

- Programming Massively Parallel Processors book, 4th edition
  Chapter 11: Prefix Sum (Scan) – an introduction to work efficiency in parallel algorithms
- Warp Shuffle Functions
  - CUDA Programming Guide, Chapter 10.22 (pdf; 7.22 online)

Read (optional):

- Guy E. Blelloch: Prefix Sums and their Applications
  - `https://www.cs.cmu.edu/~guyb/papers/Ble93.pdf/`
- CUDA Cooperative Groups
  - CUDA Programming Guide, Chapter 11 (pdf; 8 online)
  - `https://developer.nvidia.com/blog/cooperative-groups/`
- Warp Matrix Functions (==tensor core programming)
  - CUDA Programming Guide, Chapter 10.24 (pdf; 7.24 online)

Next Lectures

Lecture 25: Mon, Nov 4

Lecture 26: Tue, Nov 5 (make-up lecture; 14:30 – 15:45)

Lecture 27: Thu, Nov 7: Vulkan tutorial #2

What is in a GPU?

Lots of floating point processing power

- Stream processing cores

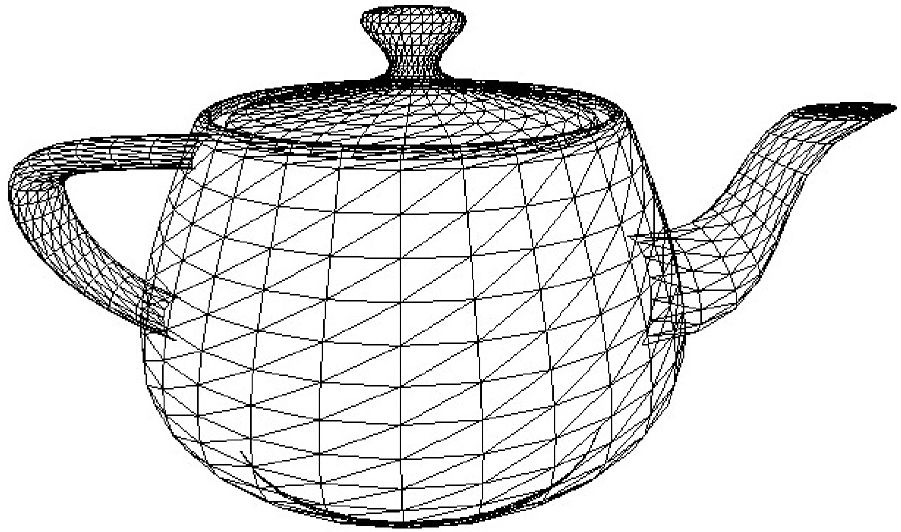
  different names:
  stream processors,
  CUDA cores, ...

- Was vector processing, now scalar cores!

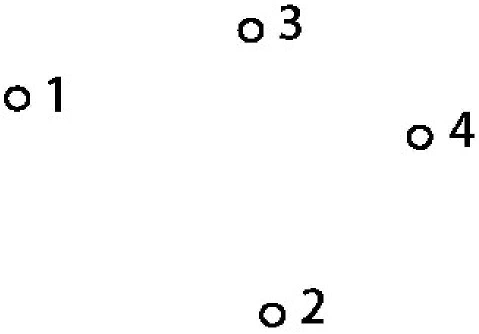
Still lots of fixed graphics functionality

- Attribute interpolation (per-vertex -> per-fragment)
- Rasterization (turning triangles into fragments/pixels)
- Texture samping and filtering
- Depth buffering (per-pixel visibility)
- Blending/compositing (semi-transparent geometry, ...)
- Frame buffers

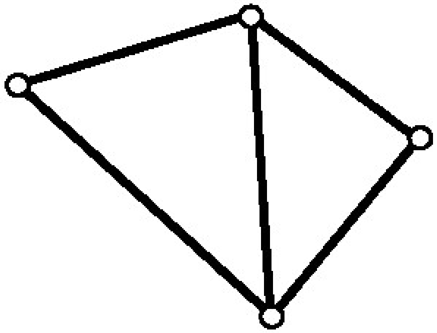
Real-time graphics primitives (entities)



Represent surface as a 3D triangle mesh



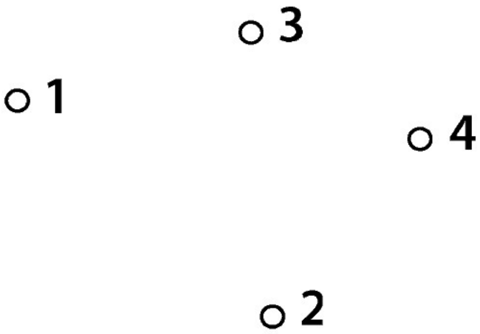
Vertices



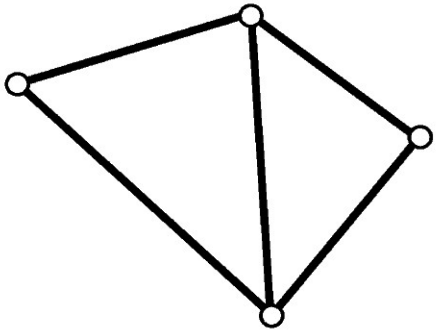
Primitives
(e.g., triangles, points, lines)

Courtesy Kayvon Fatahalian, CMU

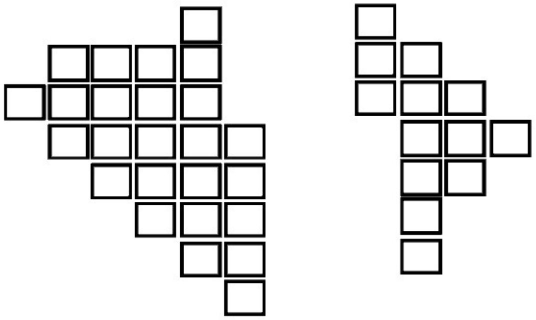
Real-time graphics primitives (entities)



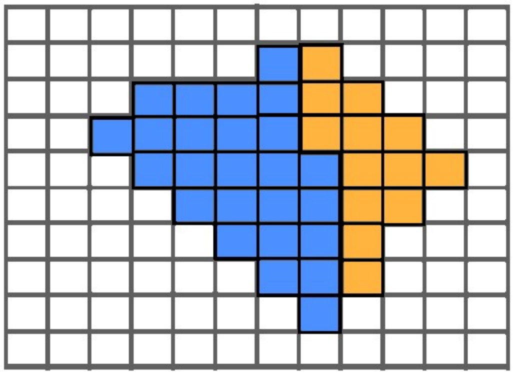
Vertices



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



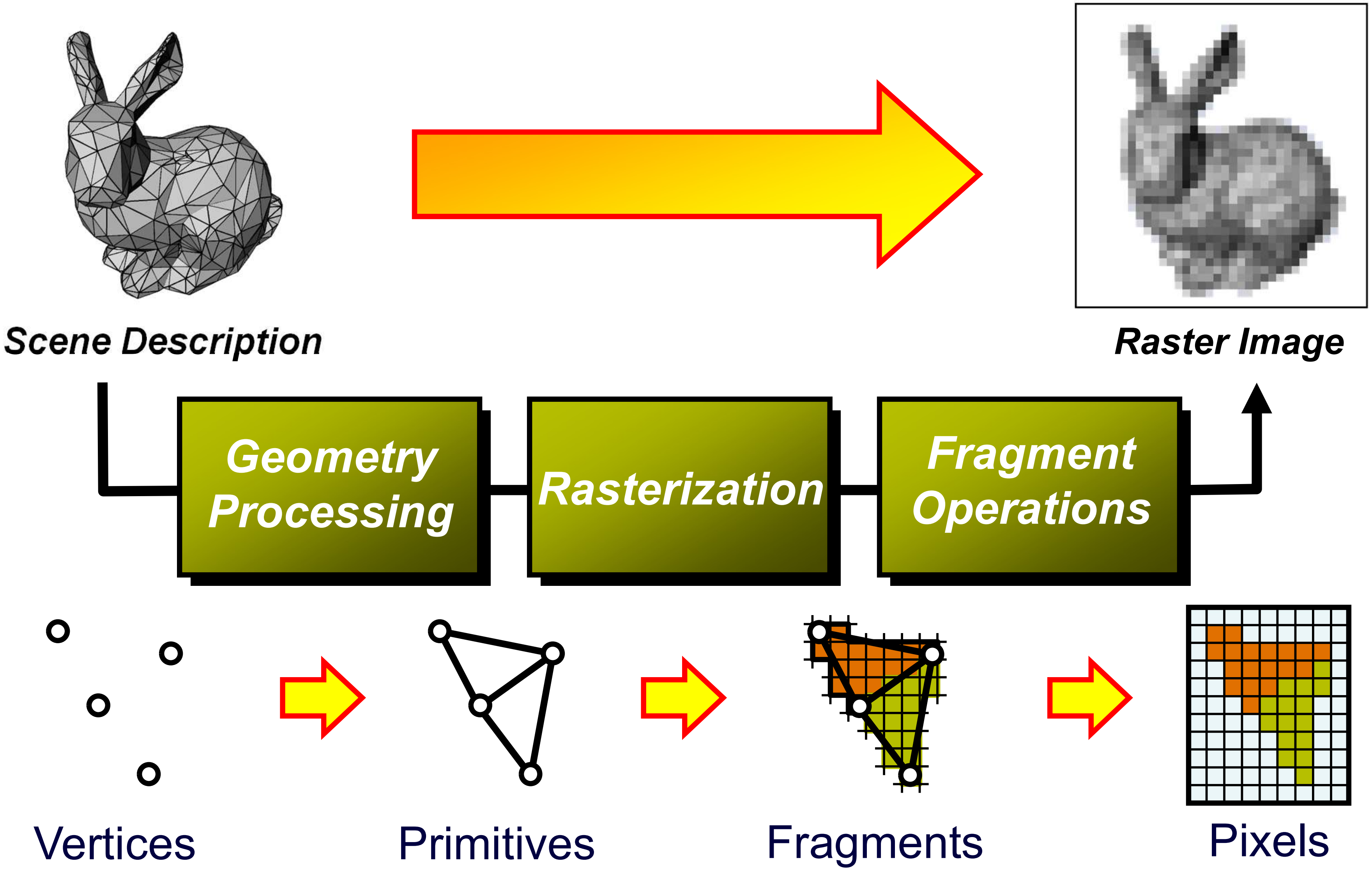
Fragments



Pixels (in an image)

Courtesy Kayvon Fatahalian, CMU

Graphics Pipeline



Scene Description

Raster Image

Geometry Processing

Rasterization

Fragment Operations

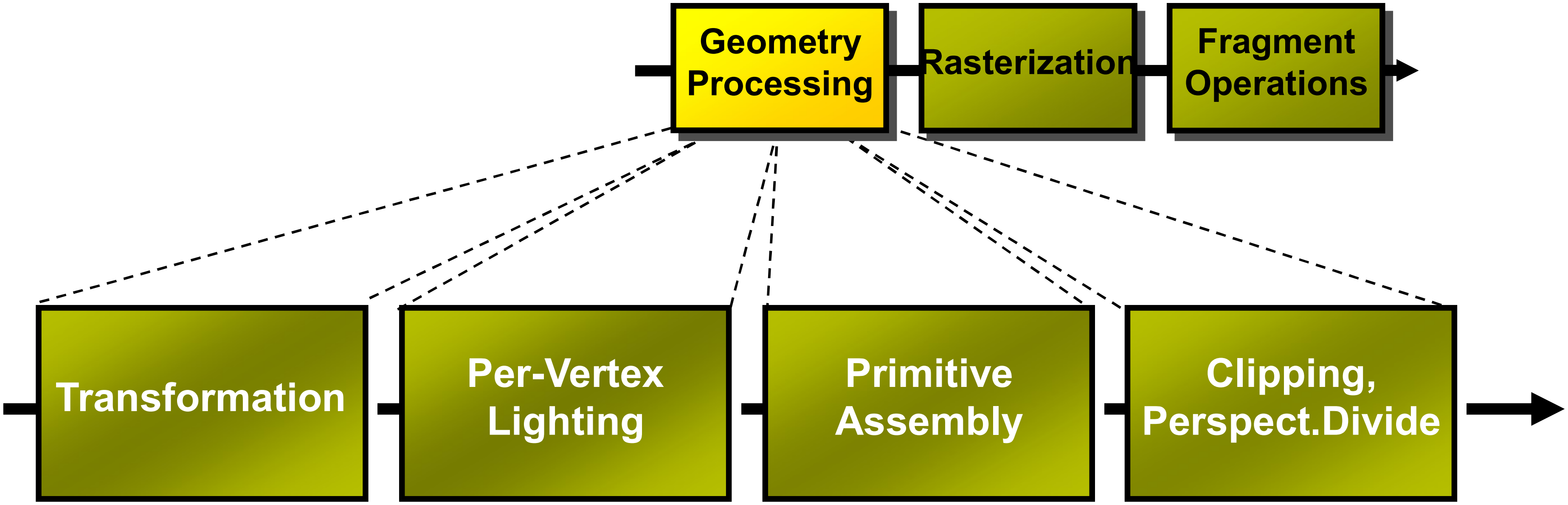
Vertices

Primitives

Fragments

Pixels

Geometry Processing



Geometry Processing

Rasterization

Fragment Operations

Transformation

Per-Vertex Lighting

Primitive Assembly

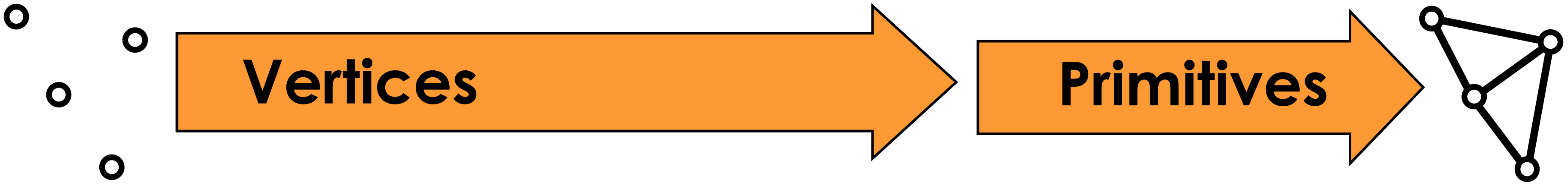
Clipping, Perspect.Divide

Multiplication with
Modelview and
Projection Matrix

Per-Vertex
Local Illumination
(Blinn/Phong)

Geometric
Primitives
(Points, Lines
Triangles)

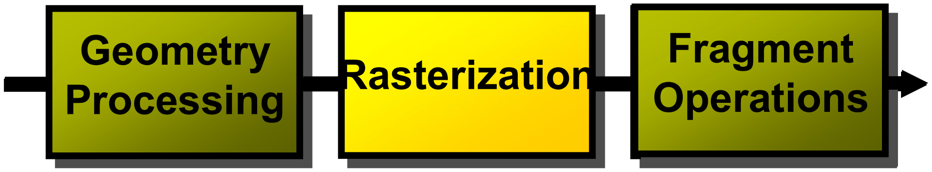
Clip Space
To
Screen Space



Vertices

Primitives

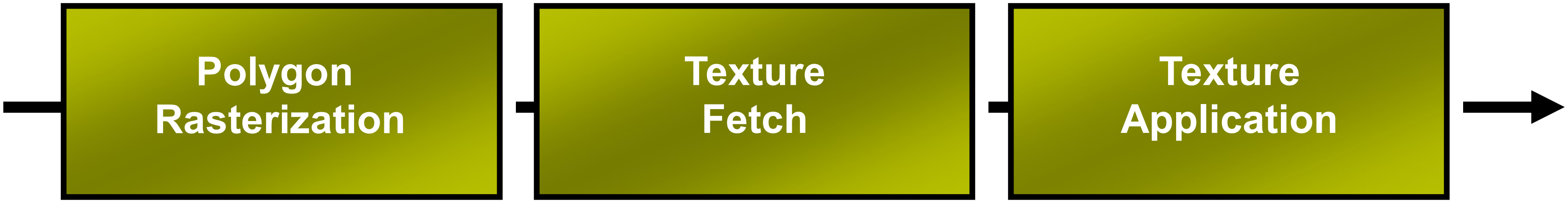
Rasterization



Geometry Processing

Rasterization

Fragment Operations



Polygon Rasterization

Texture Fetch

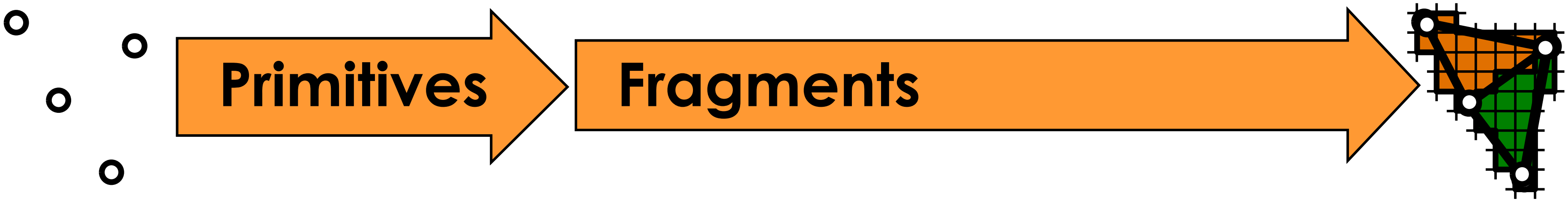
Texture Application

Decomposition of primitives into fragments

Interpolation of texture *coordinates*

Filtering of texture color

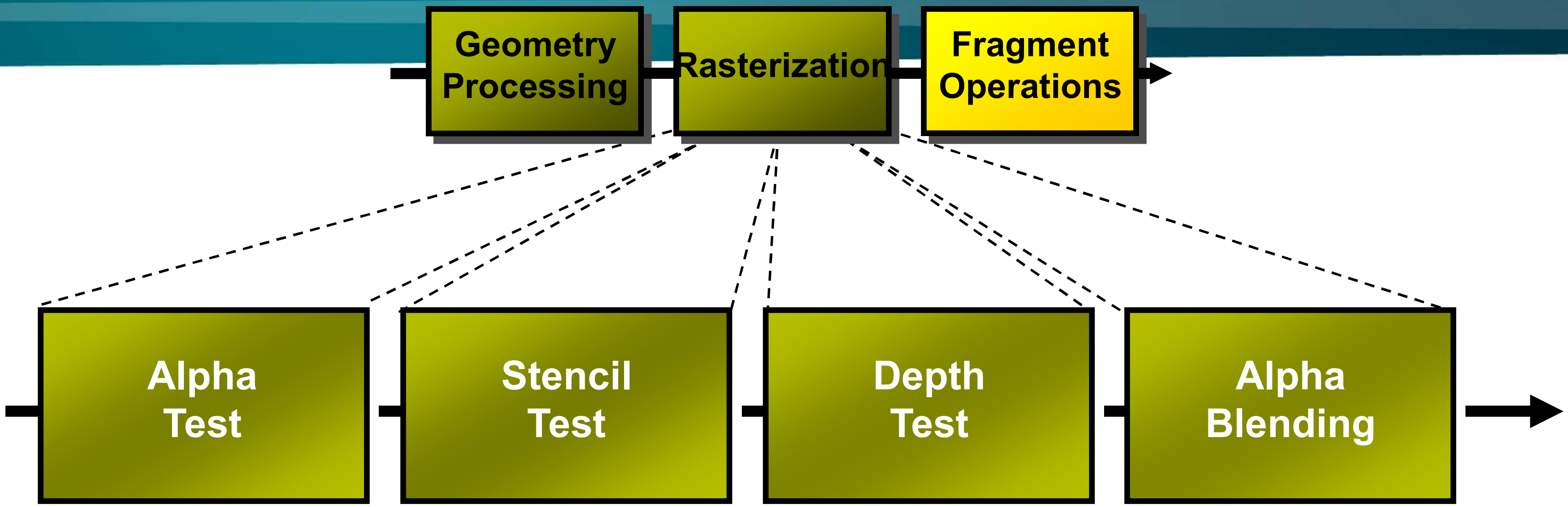
Combination of primary color with texture color



Primitives

Fragments

Fragment Operations



Geometry Processing

Rasterization

Fragment Operations

Alpha Test

Stencil Test

Depth Test

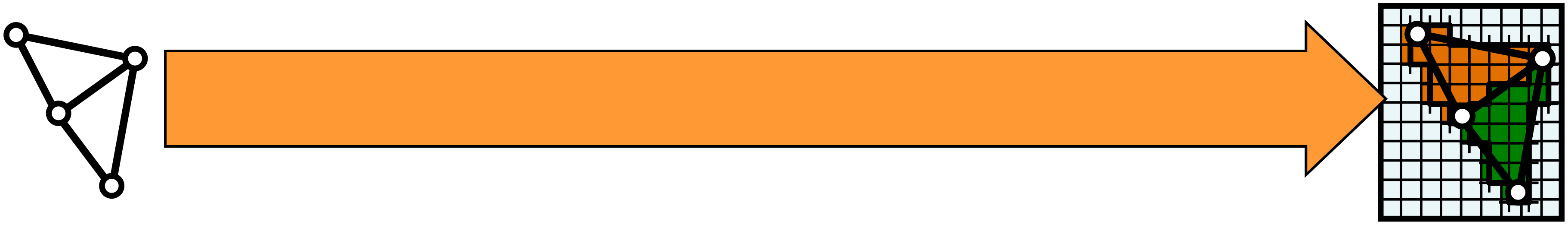
Alpha Blending

Alpha Test: Discard all fragments within a certain alpha range

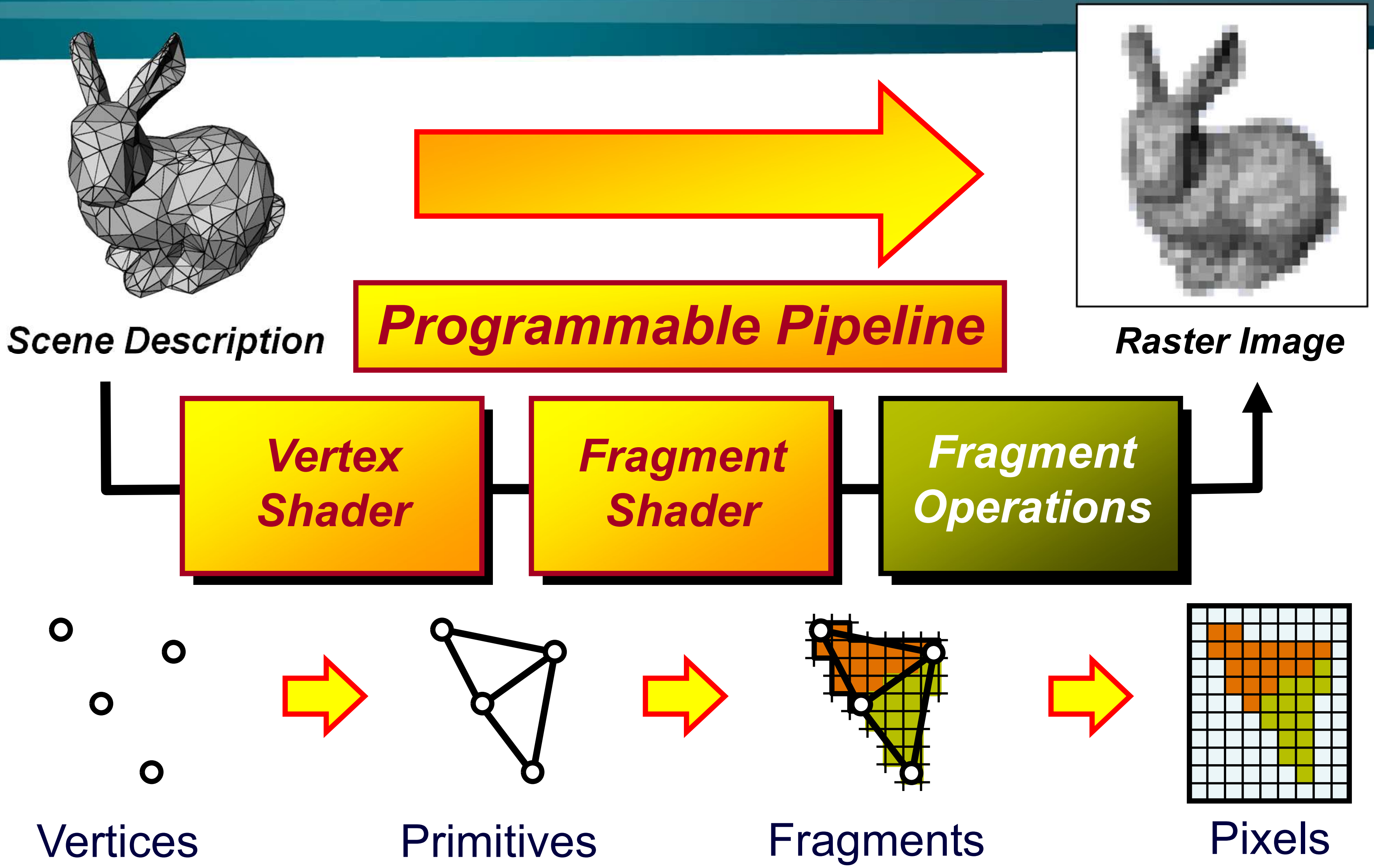
Stencil Test: Discard a fragment if the stencil buffer is set

Depth Test: Discard all occluded fragments

Alpha Blending: Combination of primary color with texture color



Graphics Pipeline



Scene Description

Programmable Pipeline

Raster Image

Vertex Shader

Fragment Shader

Fragment Operations

Vertices

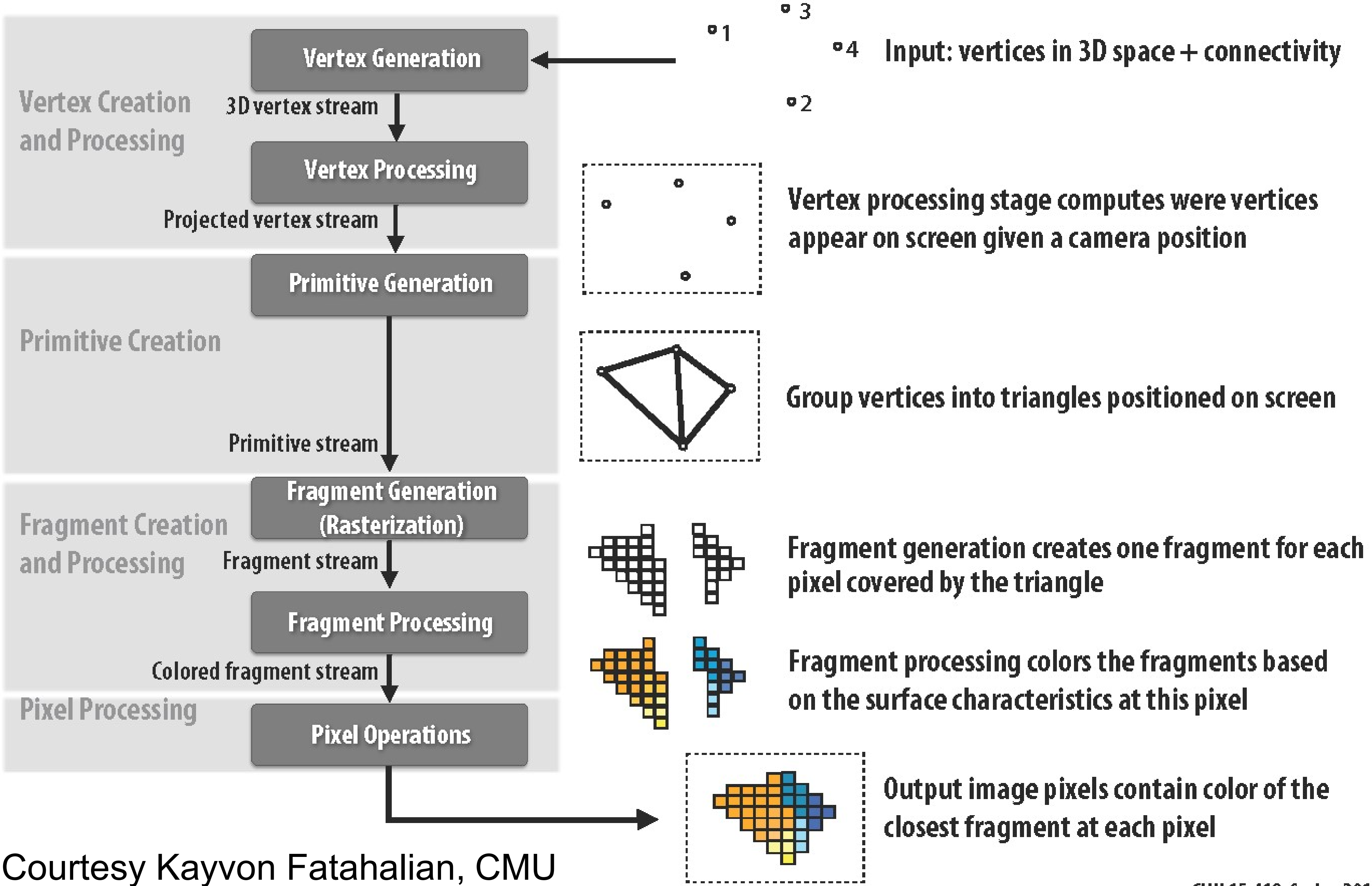
Primitives

Fragments

Pixels

Graphics pipeline architecture

Performs operations on vertices, triangles, fragments, and pixels



Vertex Creation and Processing

Vertex Generation

3D vertex stream

Vertex Processing

Projected vertex stream

Primitive Creation

Primitive Generation

Primitive stream

Fragment Creation and Processing

Fragment Generation (Rasterization)

Fragment stream

Fragment Processing

Colored fragment stream

Pixel Processing

Pixel Operations

Input: vertices in 3D space + connectivity

Vertex processing stage computes were vertices appear on screen given a camera position

Group vertices into triangles positioned on screen

Fragment generation creates one fragment for each pixel covered by the triangle

Fragment processing colors the fragments based on the surface characteristics at this pixel

Output image pixels contain color of the closest fragment at each pixel

Courtesy Kayvon Fatahalian, CMU

Direct3D 10 Pipeline (~OpenGL 3.2)

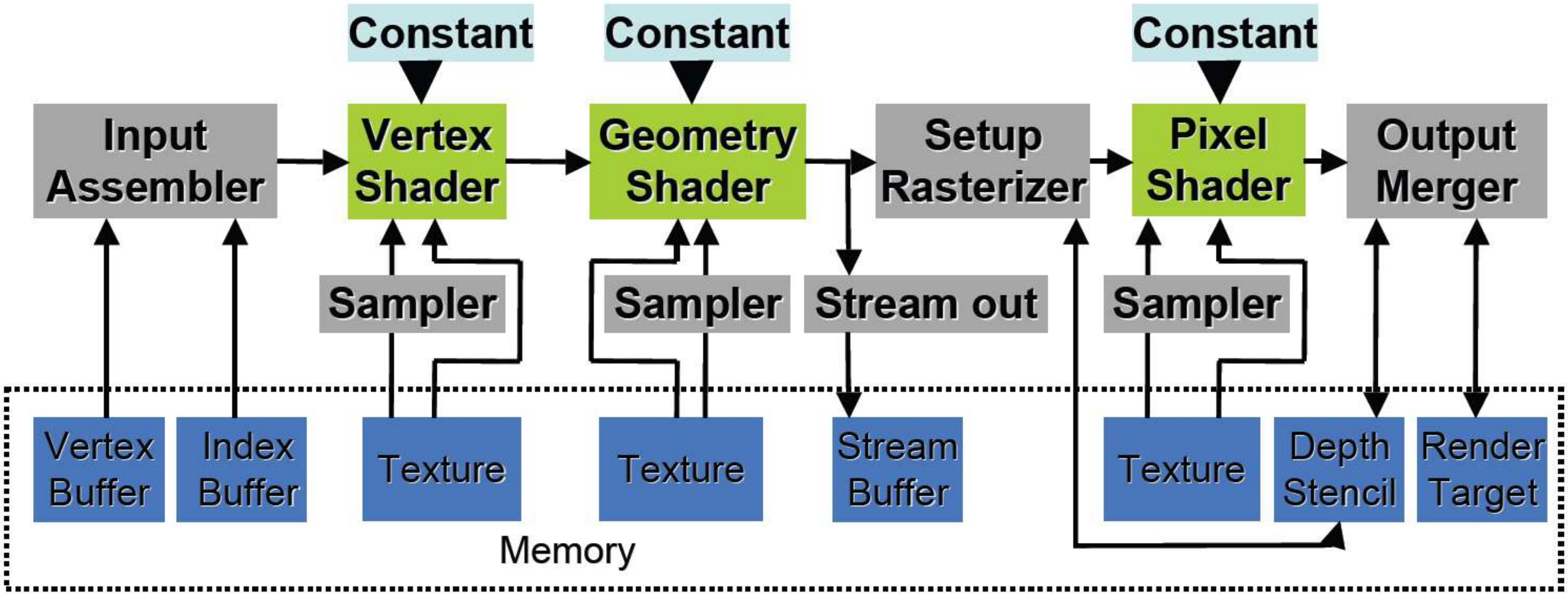
New geometry shader stage:

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

◼ *fixed*

🟩 *programmable*

🟦 *memory*



Courtesy David Blythe, Microsoft

Direct3D 11 Pipeline (~OpenGL 4.x)

New tessellation stages

- Hull shader

  (OpenGL: *tessellation control*)

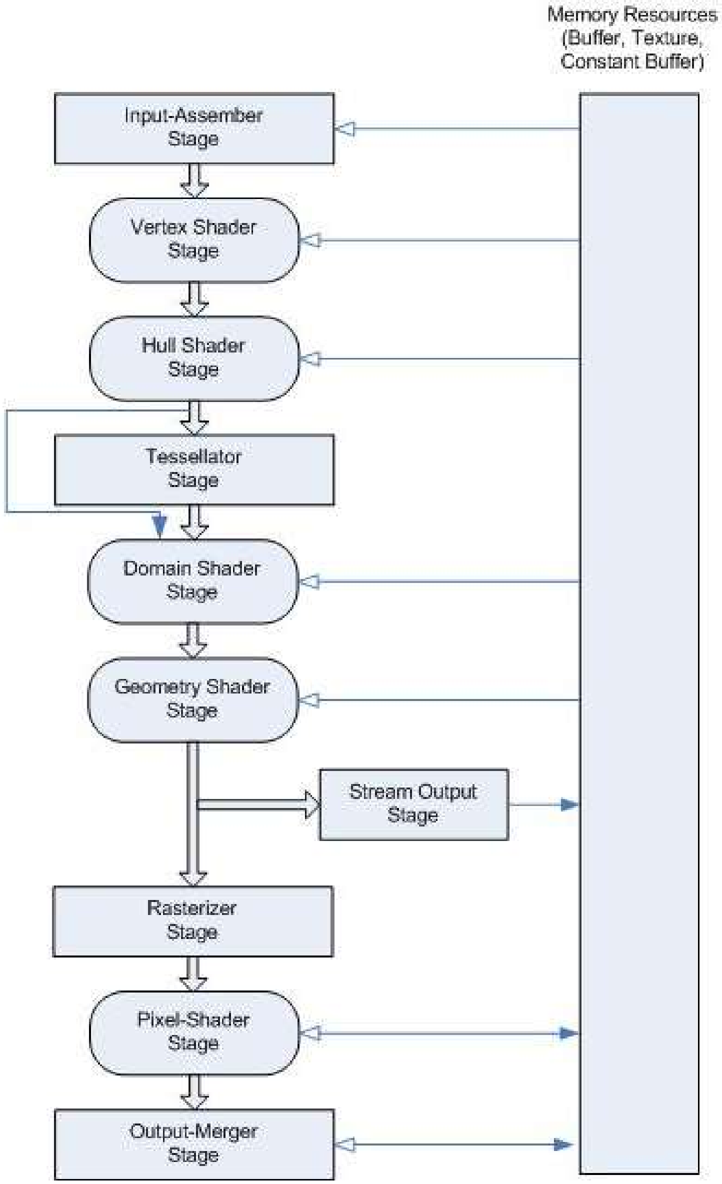
- Tessellator

  (OpenGL: *tessellation primitive generator*)

- Domain shader

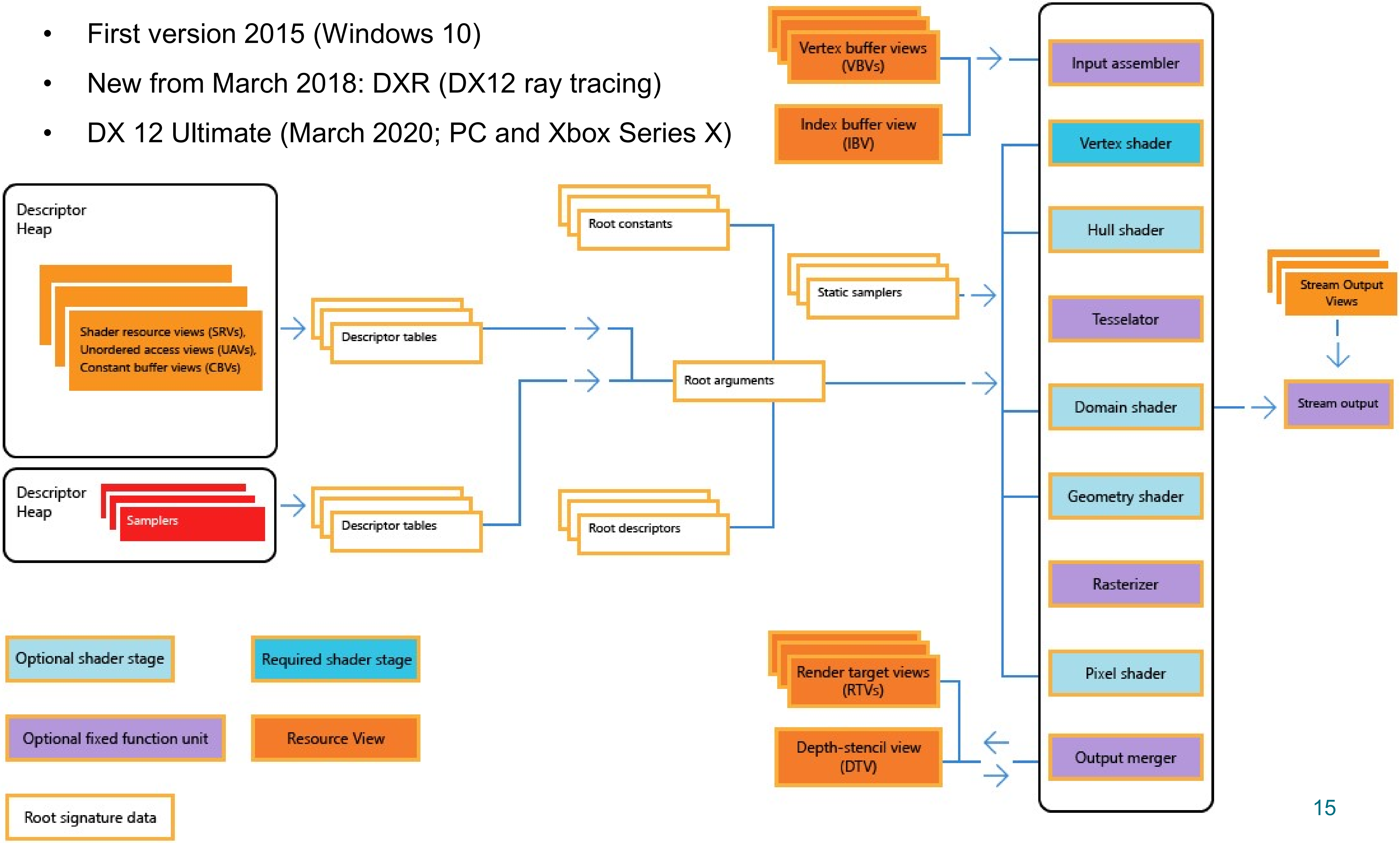
  (OpenGL: *tessellation evaluation*)

- In future versions, there might be yet more stages, but for some time now all additions were outside this pipeline:
  - Compute shaders
  - Vulkan
  - Ray tracing cores



Direct3D 12 Geometry Pipeline (Traditional)

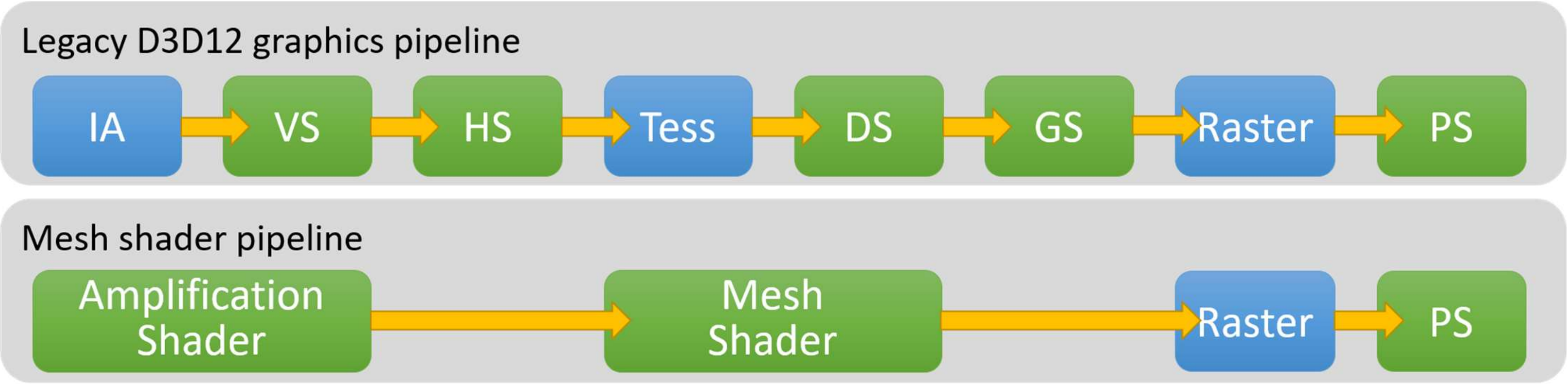
- First version 2015 (Windows 10)
- New from March 2018: DXR (DX12 ray tracing)
- DX 12 Ultimate (March 2020; PC and Xbox Series X)



Direct3D 12 Mesh Shader Pipeline

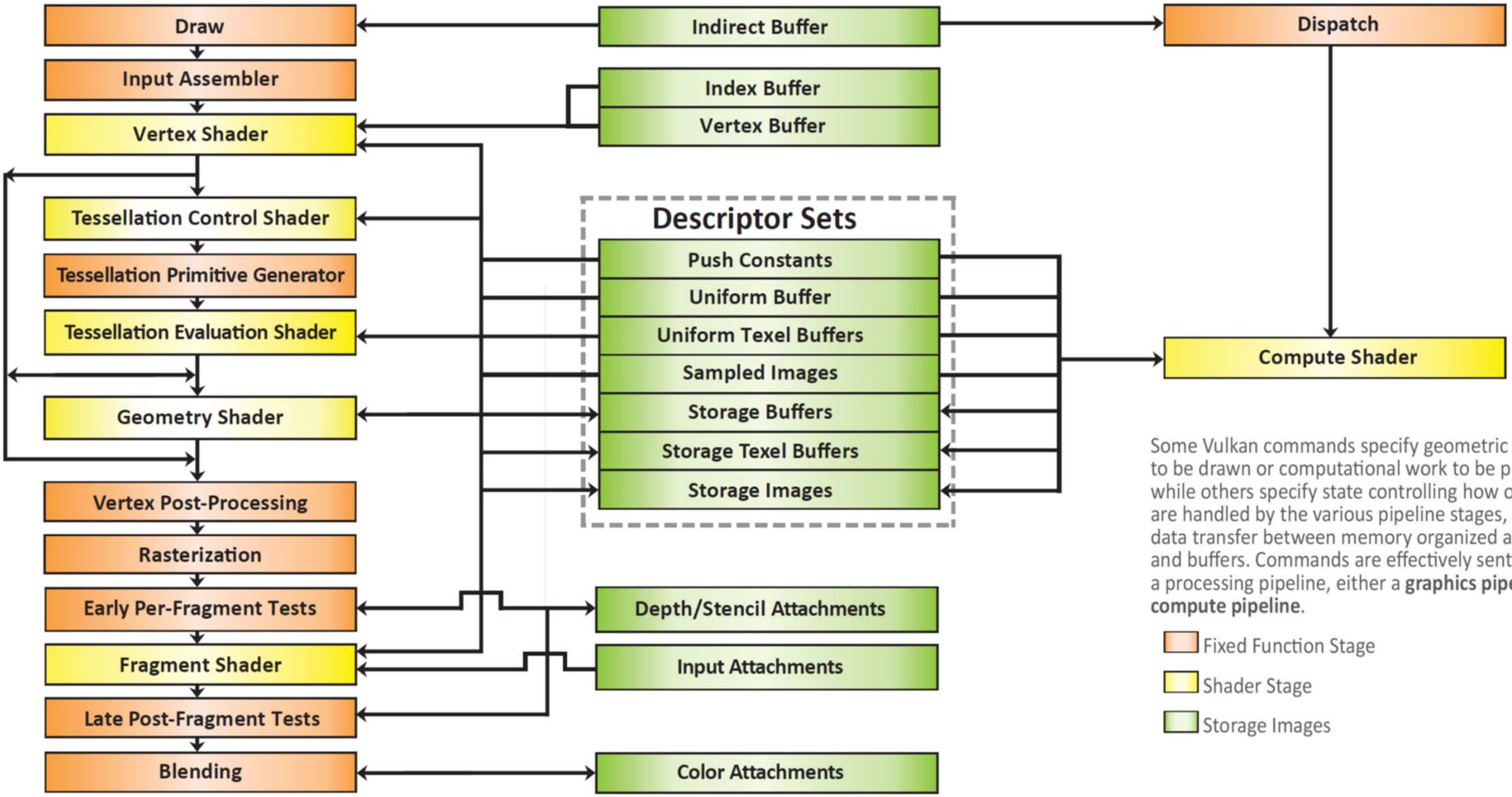
Reinventing the Geometry Pipeline

- 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) Pipeline (Traditional)



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

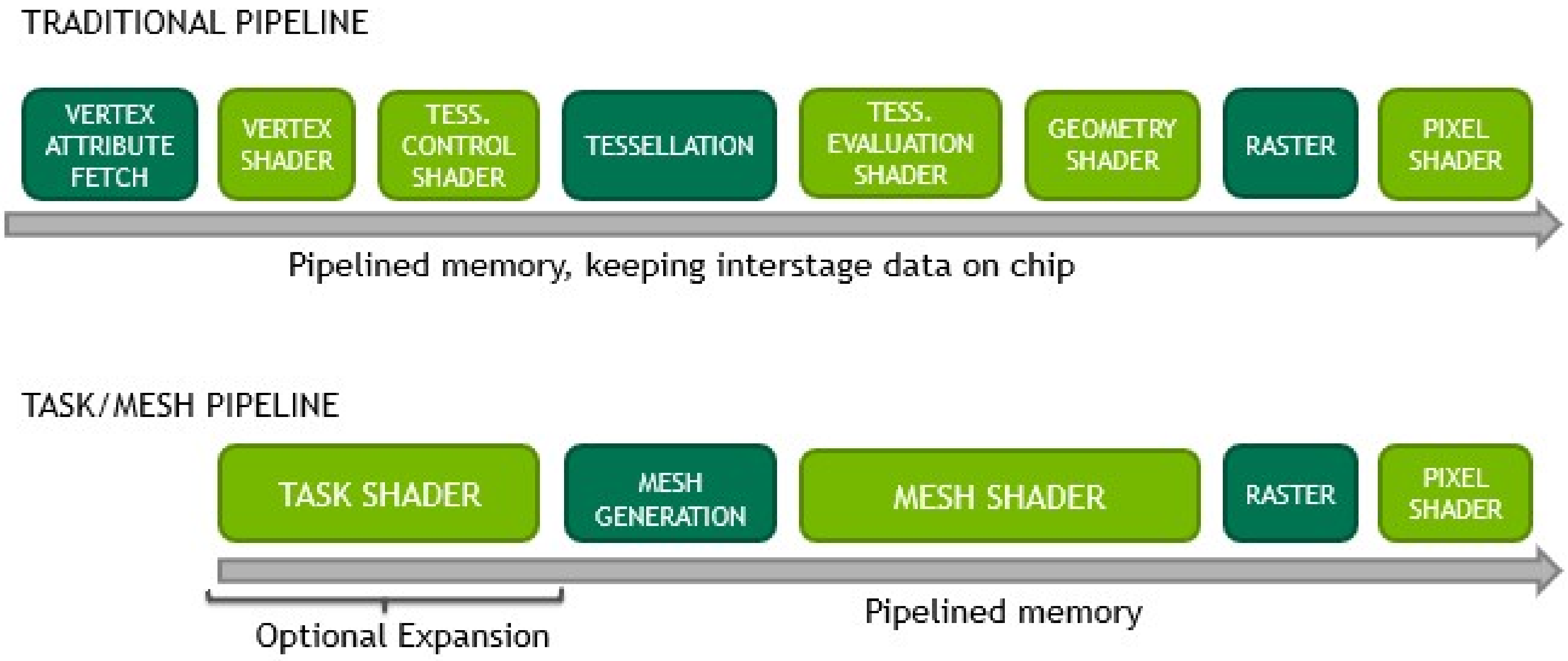
🟨 Shader Stage

🟩 Storage Images

Vulkan (1.3) Pipelines

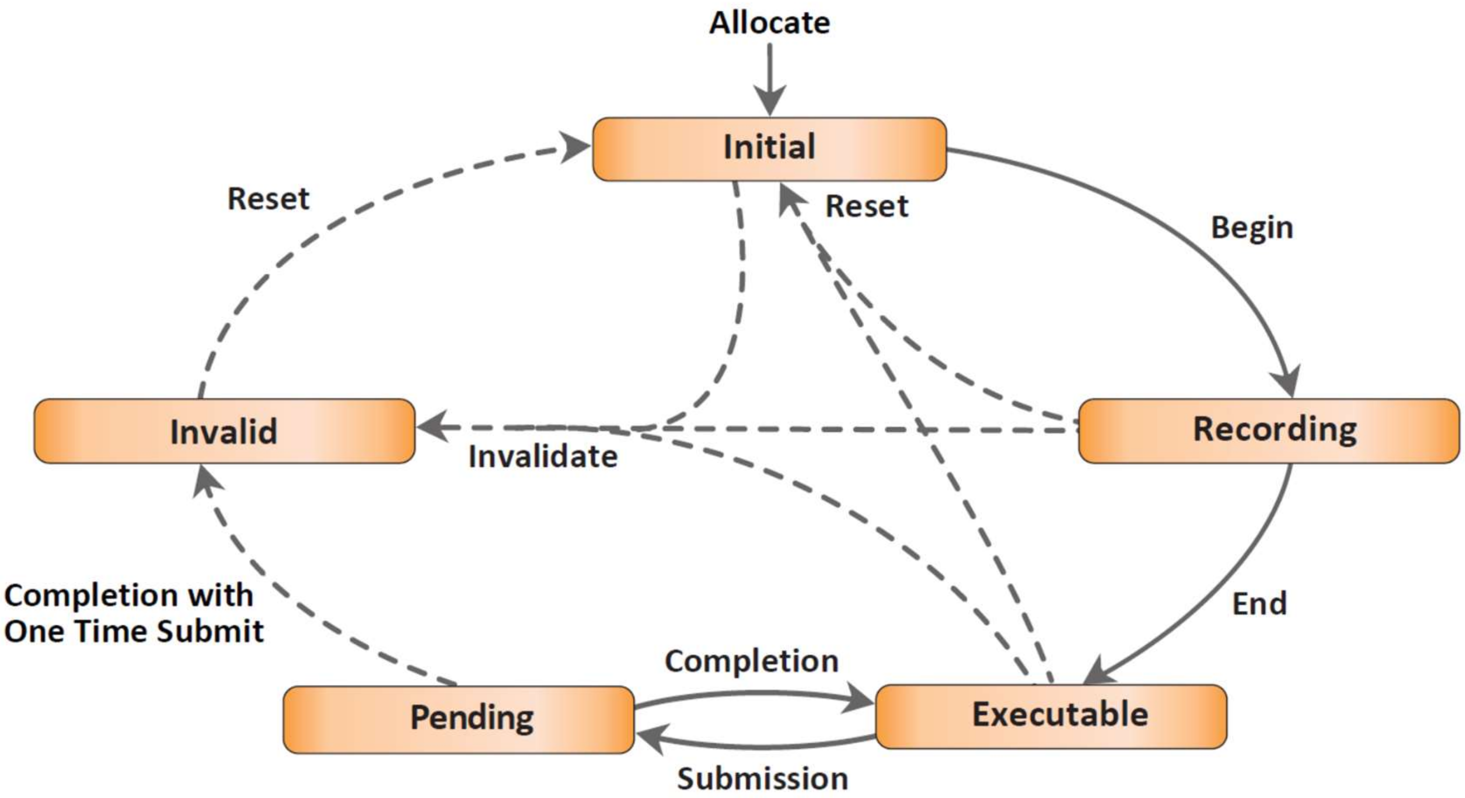
- Mesh and task shaders: new high-performance geometry pipeline based on compute shaders

  (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

Vulkan Command Buffer Lifecycle



Initial state

The state when a command buffer is first allocated. The command buffer may be reset back to this state from any of the executable, recording, or invalid states. Command buffers in the initial state can only be moved to recording, or freed.

Recording state

vkBeginCommandBuffer changes the state from initial to recording. Once in the recording state, **vkCmd*** commands can be used to record to the command buffer.

Executable state

vkEndCommandBuffer moves a command buffer state from recording to executable.

Executable command buffers can be submitted, reset, or recorded to another command buffer.

Pending state

Queue submission changes the state from executable to pending, in which applications must not attempt to modify the command buffer in any way. The state reverts back to executable when current executions complete, or to invalid.

Invalid state

Some operations will transition the command buffer into the invalid state, in which it can only be reset or freed.

GPU Texturing

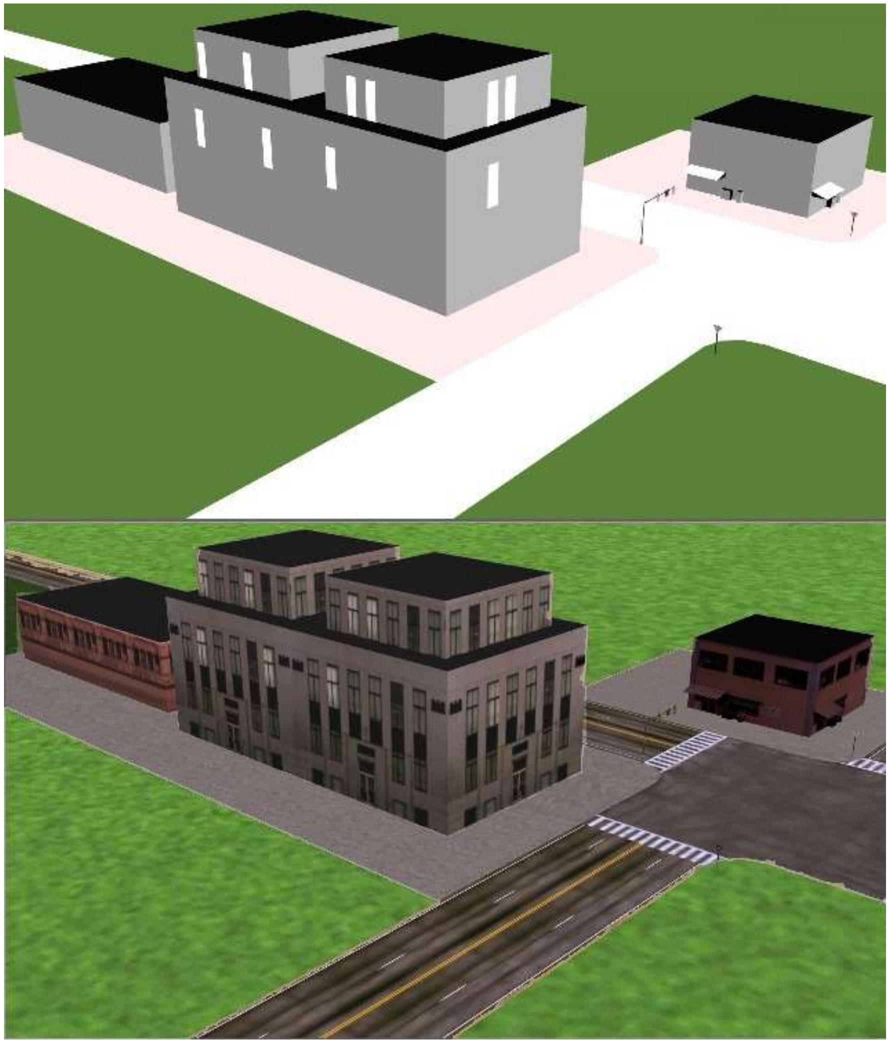
GPU Texturing

FG
freakygaming.com

Rage / id Tech 5 (id Software)

Why Texturing?

- Idea: enhance visual appearance of surfaces by applying fine / high-resolution details



OpenGL Texture Mapping

- Basis for most real-time rendering effects
- Look and feel of a surface
- Definition:
  - A *regularly sampled function* that is mapped onto every *fragment* of a surface
  - Traditionally an image, but…
- Can hold arbitrary information
  - Textures become general data structures
  - Sampled and interpreted by fragment programs
  - Can render into textures → important!

Types of Textures

- Spatial layout
  - Cartesian grids: 1D, 2D, 3D, 2D_ARRAY, …
  - Cube maps, …

for Vulkan, see vkImageView

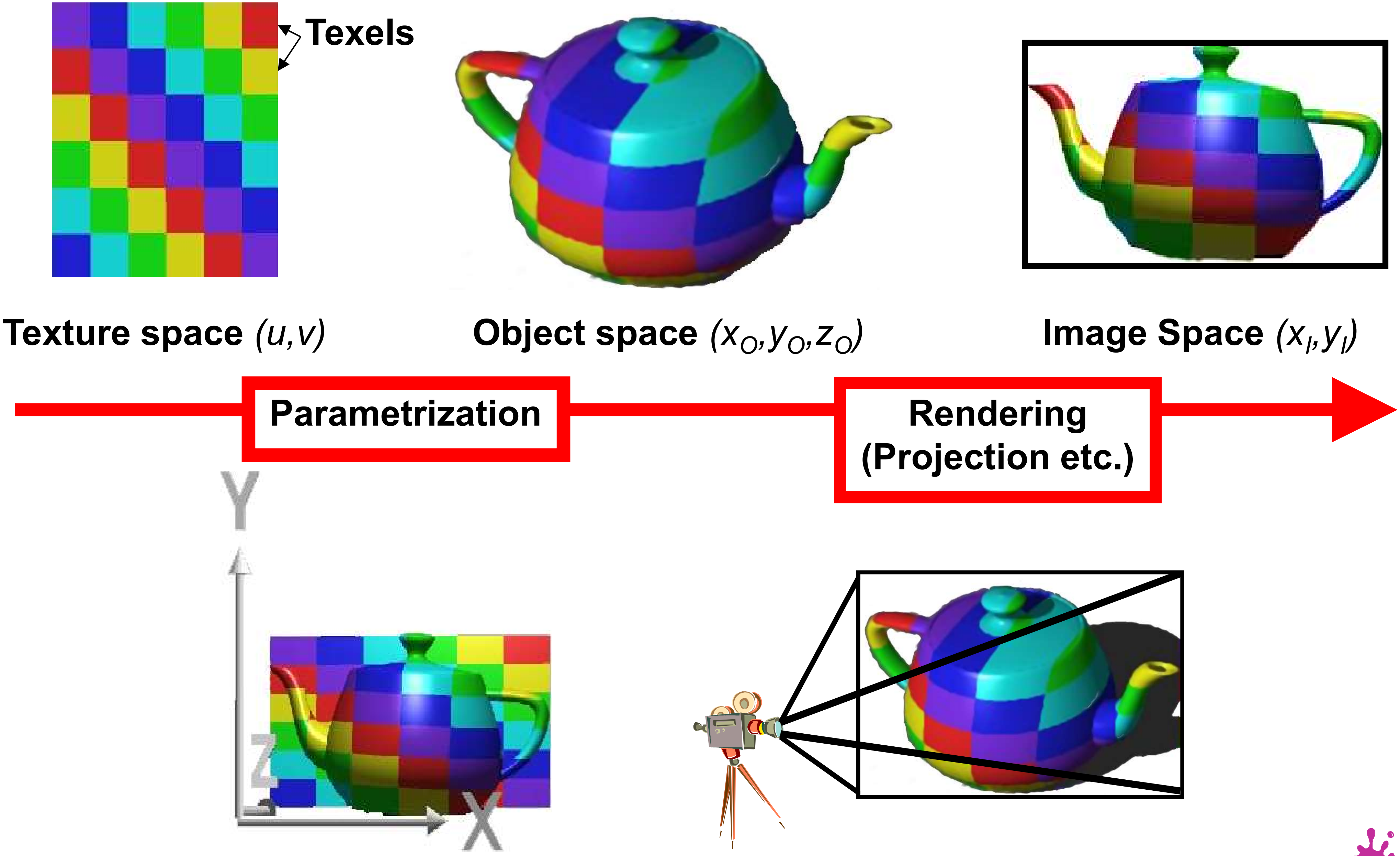
- Formats (too many), e.g. OpenGL
  - GL_LUMINANCE16_ALPHA16
  - GL_RGB8, GL_RGBA8, …: integer texture formats
  - GL_RGB16F, GL_RGBA32F, …: float texture formats
  - compressed formats, high dynamic range formats, …

- External (CPU) format vs. internal (GPU) format
  - OpenGL driver converts from external to internal

for Vulkan, see vkImage
and vkImageView

use VK_IMAGE_TILING_OPTIMAL
for VkImageCreateInfo::tiling

Texturing: General Approach



Texels

Texture space *(u,v)*

Object space *(x_O,y_O,z_O)*

Image Space *(x_I,y_I)*

Parametrization

Rendering (Projection etc.)

Texture Mapping

2D (3D) Texture Space

↓ Texture Transformation

2D Object Parameters

↓ Parameterization

3D Object Space

↓ Model Transformation

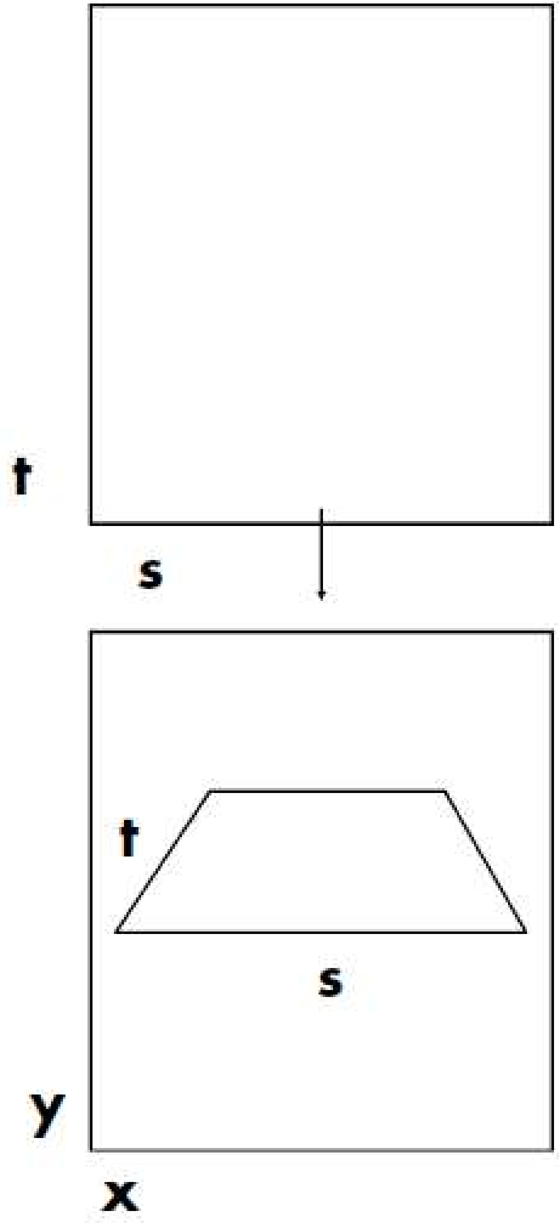
3D World Space

↓ Viewing Transformation

3D Camera Space

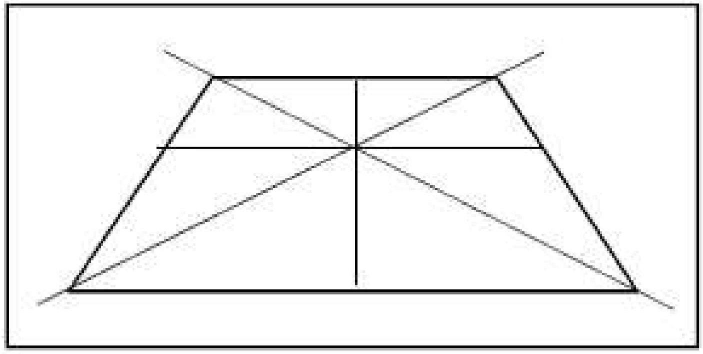
↓ Projection

2D Image Space

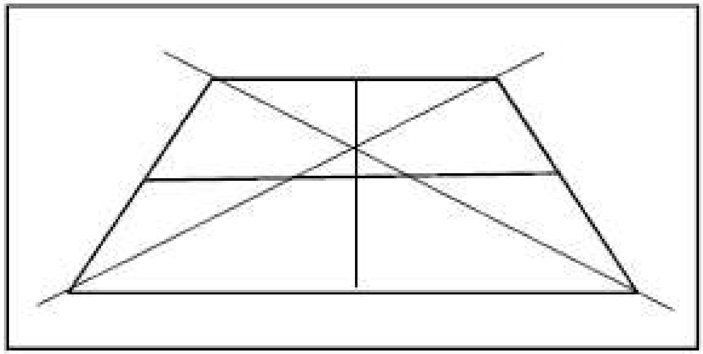


Kurt Akeley, Pat Hanrahan

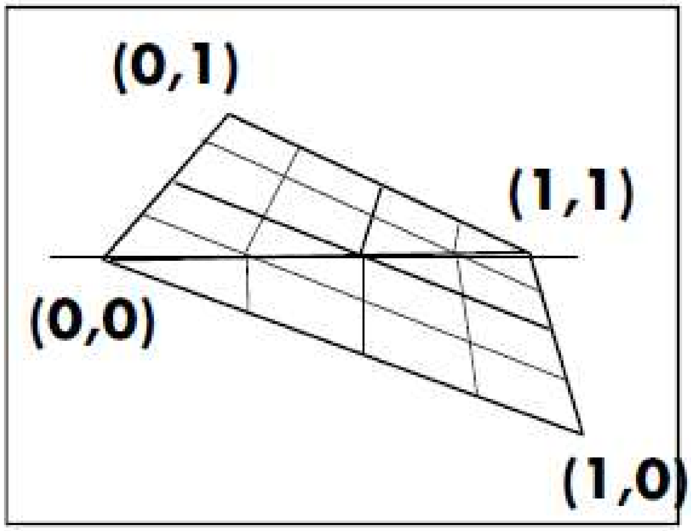
Linear Perspective



Correct Linear Perspective



Incorrect Perspective



Linear Interpolation, *Bad*

Perspective Interpolation, *Good*

Kurt Akeley, Pat Hanrahan

Thank you.