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

Markus Hadwiger, KAUST

Reading Assignment #2 (until Sep 14)

Read (required):

- Orange book (GLSL), chapter 4 (*The OpenGL Programmable Pipeline*)
- Brief GLSL overview:

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

- GPU Gems 2 book, chapter 30 (*The GeForce 6 Series GPU Architecture*) available online:

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

What is in a GPU?

Lots of floating point processing power

- Stream processing cores

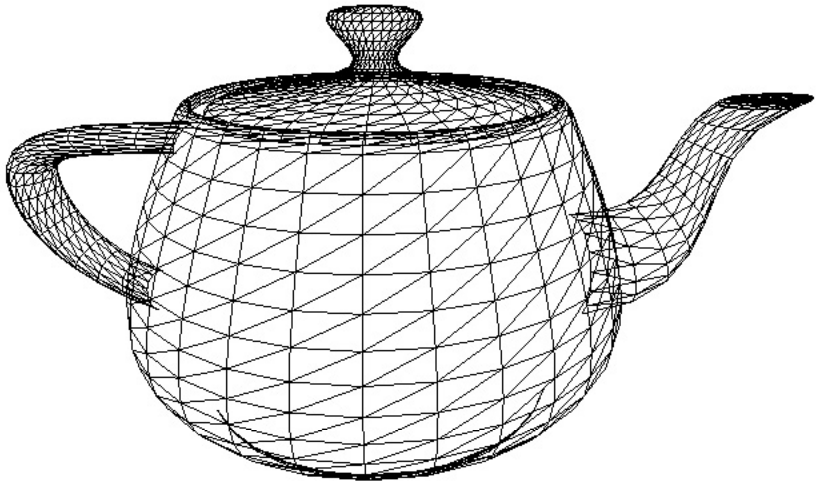
  different names:
  stream processors,
  CUDA cores, ...

- Was vector processing, now scalar cores!

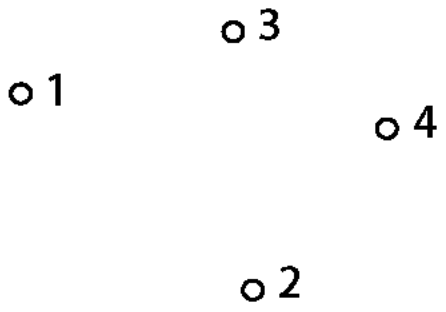
Still lots of fixed graphics functionality

- 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

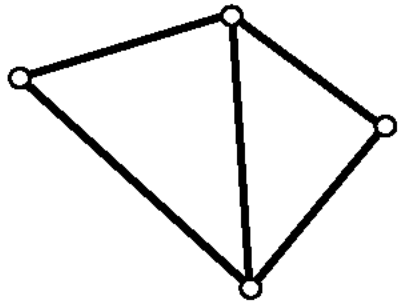
Real-time graphics primitives (entities)



Represent surface as a 3D triangle mesh



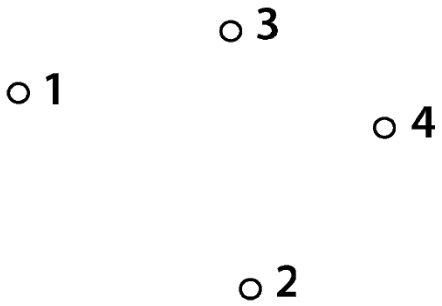
Vertices



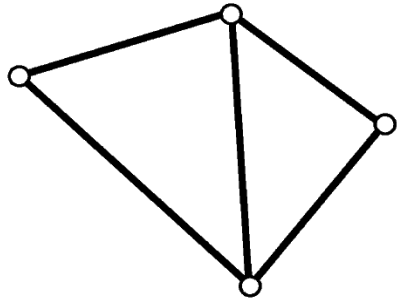
Primitives
(e.g., triangles, points, lines)

Courtesy Kayvon Fatahalian, CMU

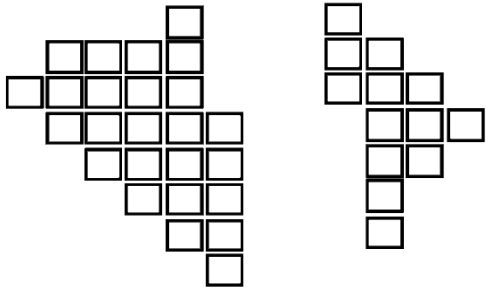
Real-time graphics primitives (entities)



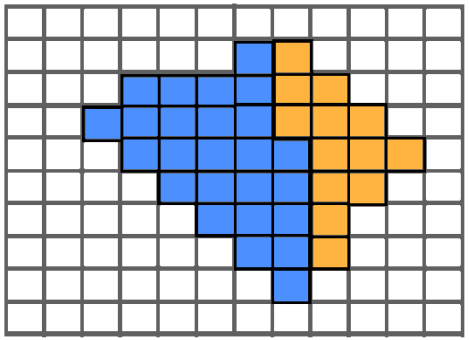
Vertices



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



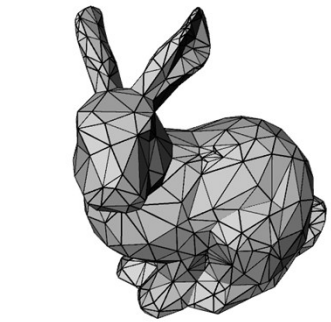
Fragments

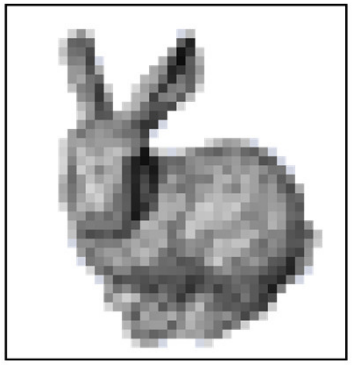


Pixels (in an image)

Courtesy Kayvon Fatahalian, CMU

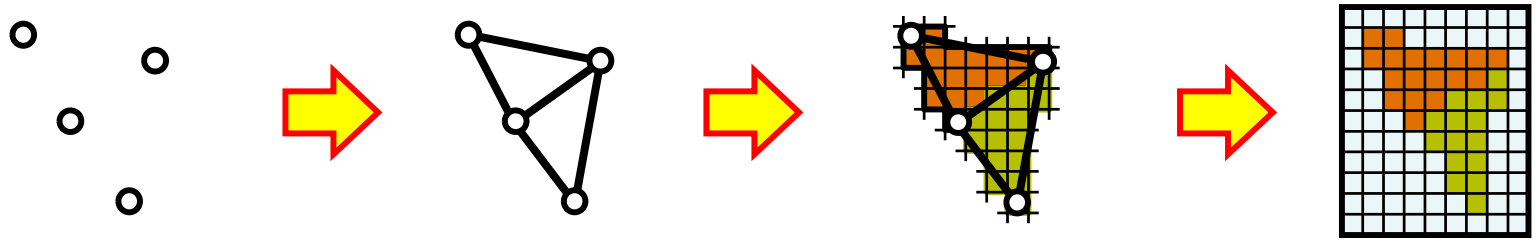
Graphics Pipeline





Scene Description

Raster Image



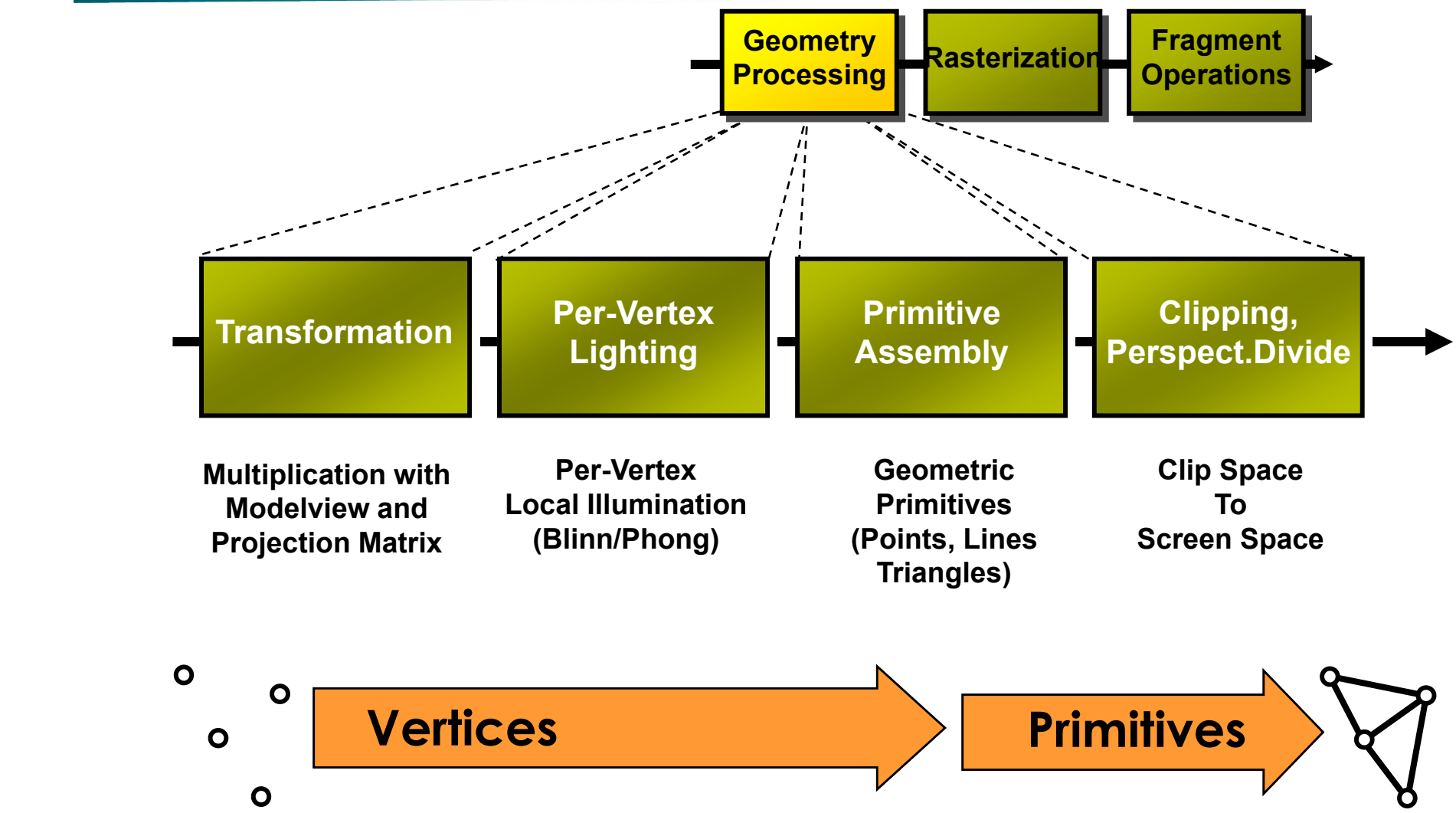
Vertices

Primitives

Fragments

Pixels

Geometry Processing



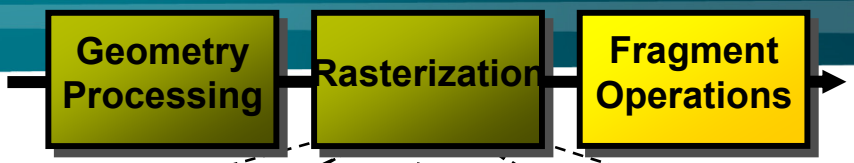
Rasterization

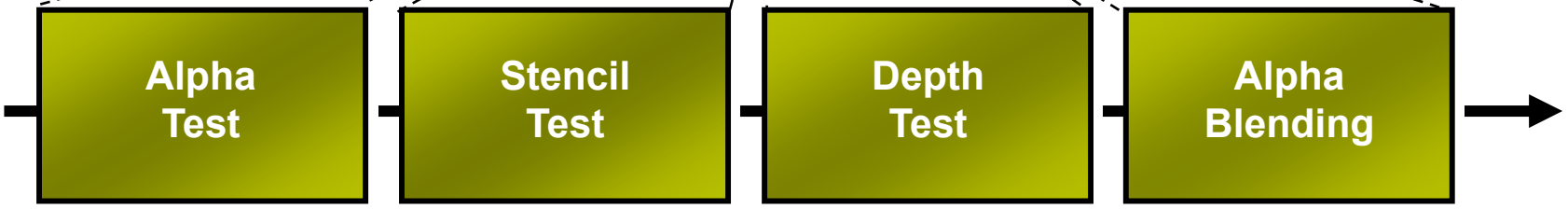
Decomposition of primitives into fragments

Interpolation of texture *coordinates*
Filtering of texture color

Combination of primary color with texture color

Fragment Operations



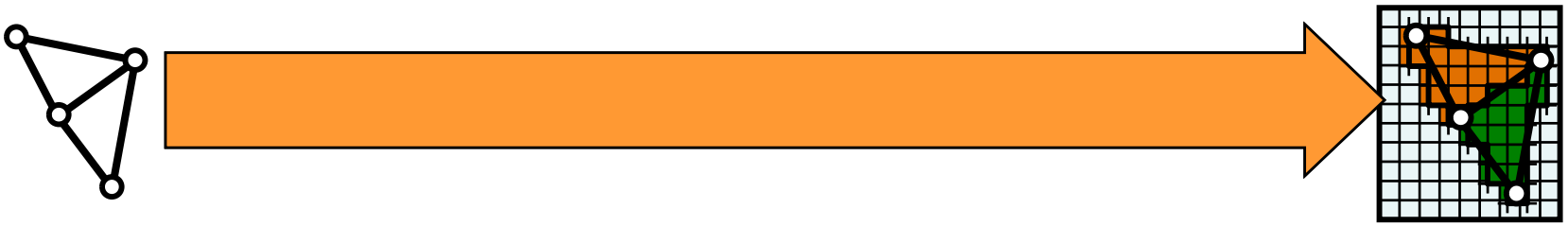


Discard all fragments within a certain alpha range

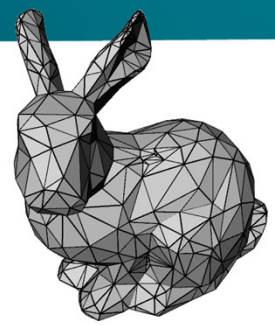
Discard a fragment if the stencil buffer is set

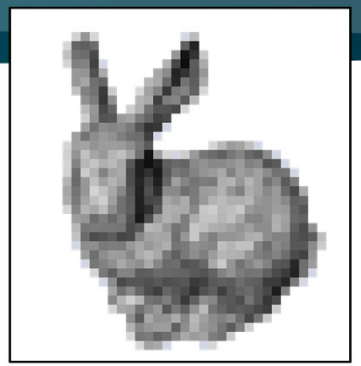
Discard all occluded fragments

Combination of primary color with texture color



Graphics Pipeline





Scene Description

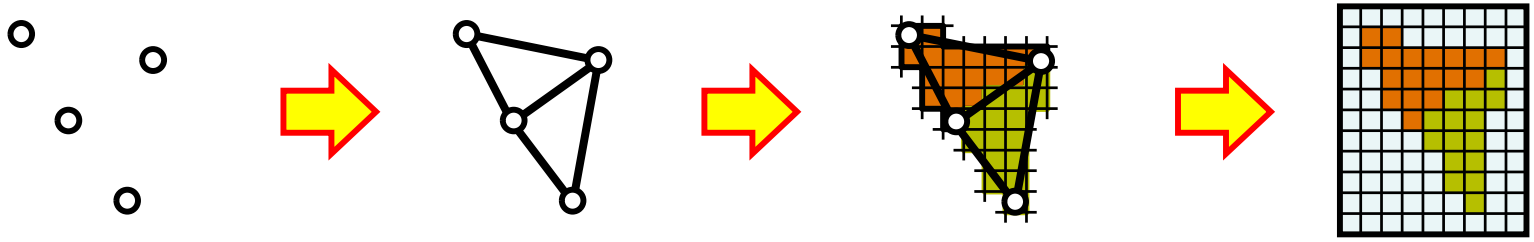
Programmable Pipeline

Raster Image

Vertex Shader

Fragment Shader

Fragment Operations



Vertices

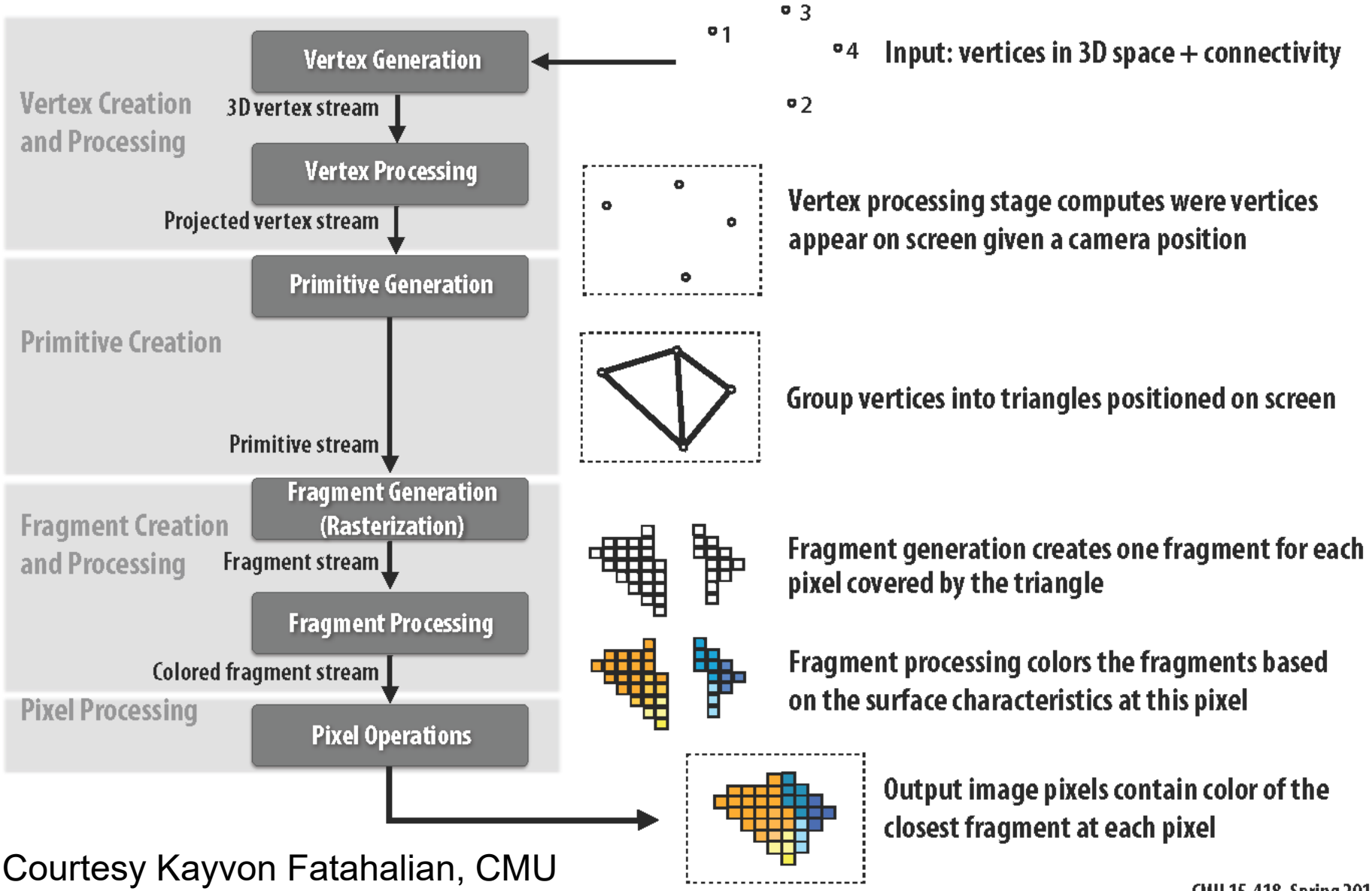
Primitives

Fragments

Pixels

Graphics pipeline architecture

Performs operations on vertices, triangles, fragments, and pixels

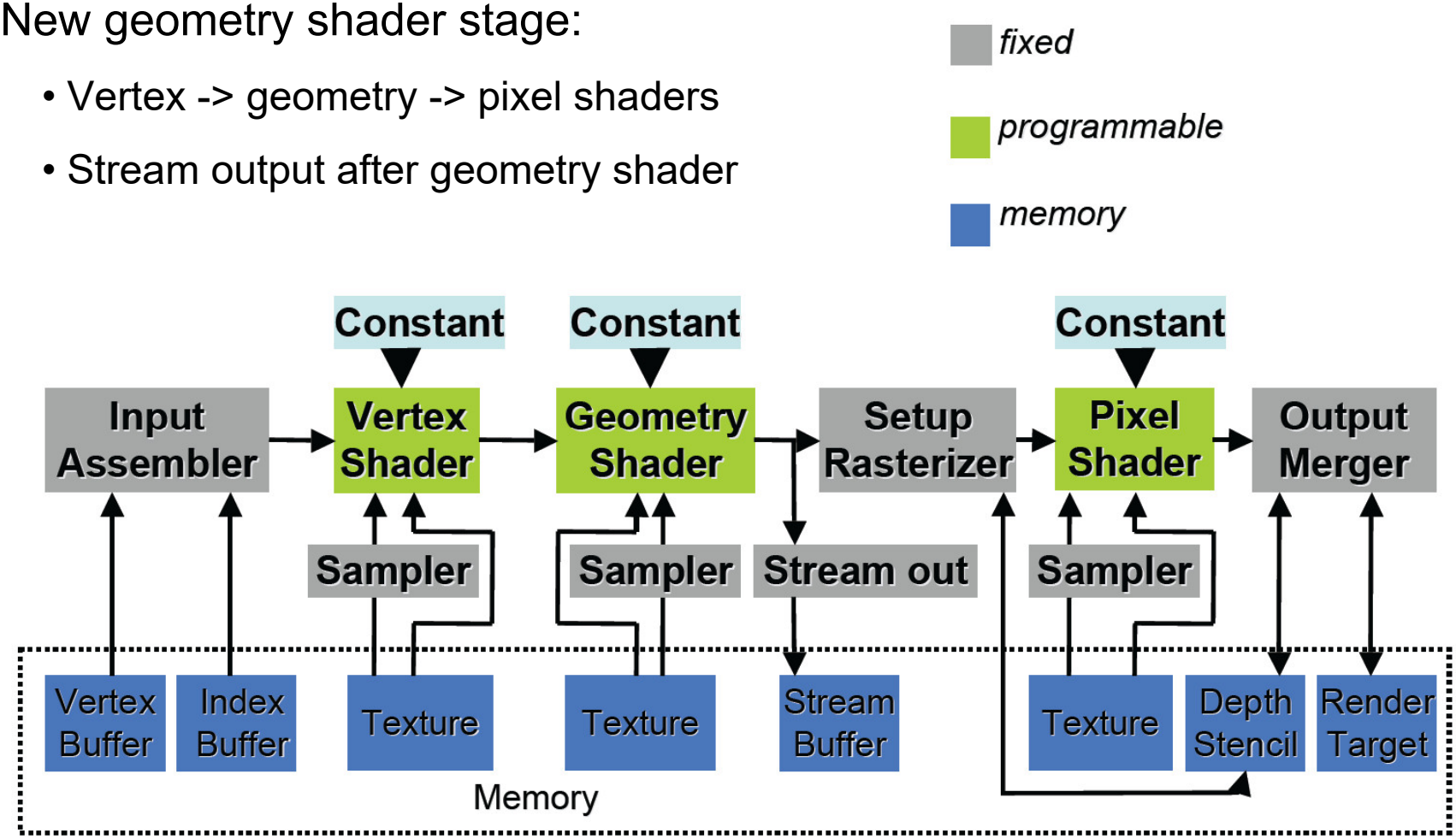


Courtesy Kayvon Fatahalian, CMU

Direct3D 10 Pipeline (~OpenGL 3.2)

New geometry shader stage:

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



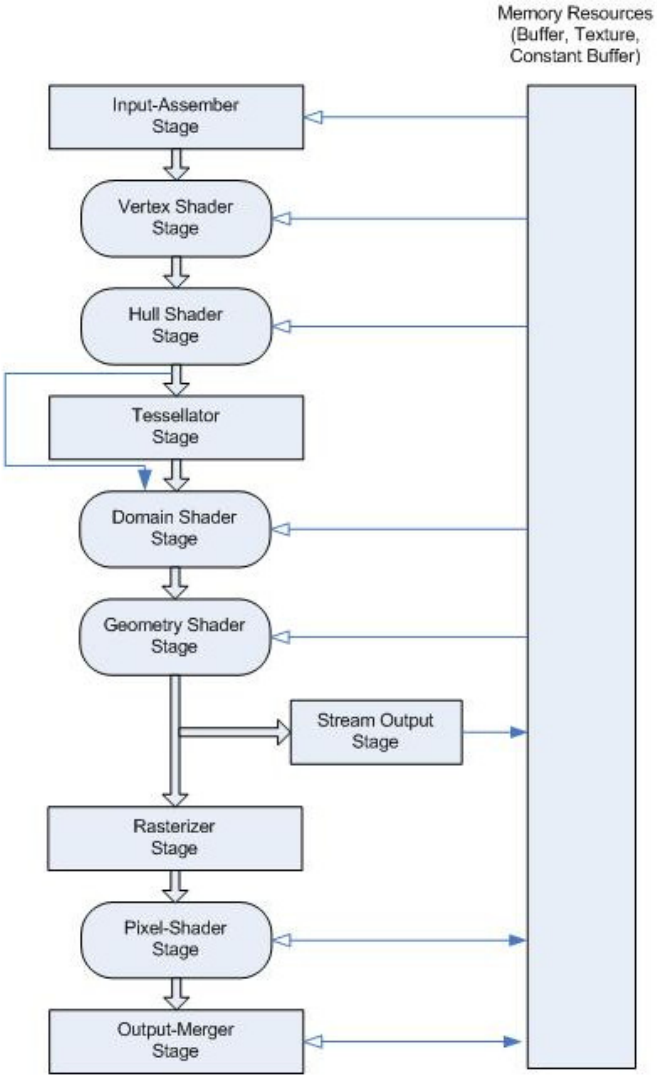
Courtesy David Blythe, Microsoft

Direct3D 11 Pipeline (~OpenGL 4.x)

New tessellation stages

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

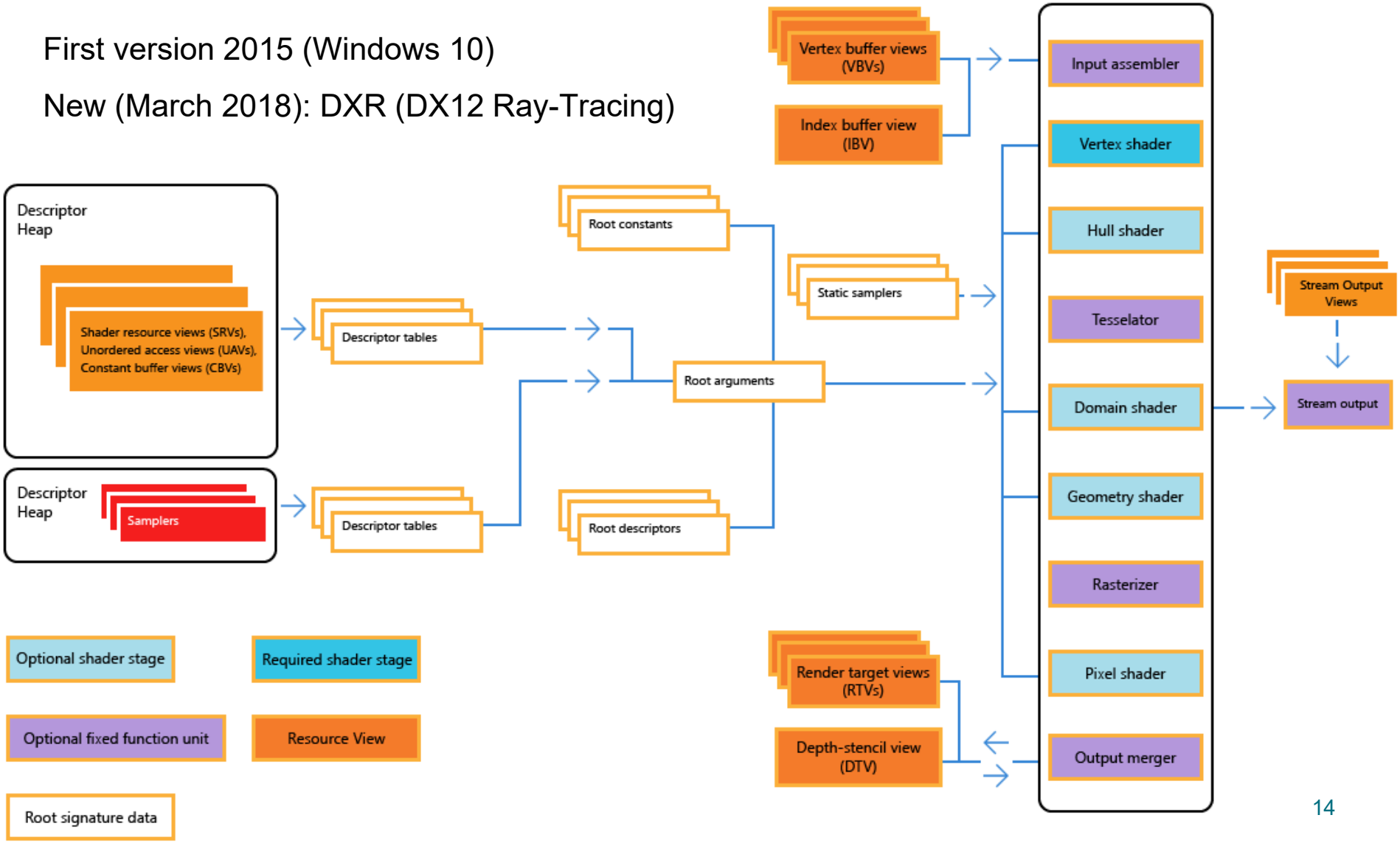
- In future versions, there might be yet more stages, but for some time now all additions were outside this pipeline:
  – Compute shaders
  – Vulkan
  – Ray tracing cores



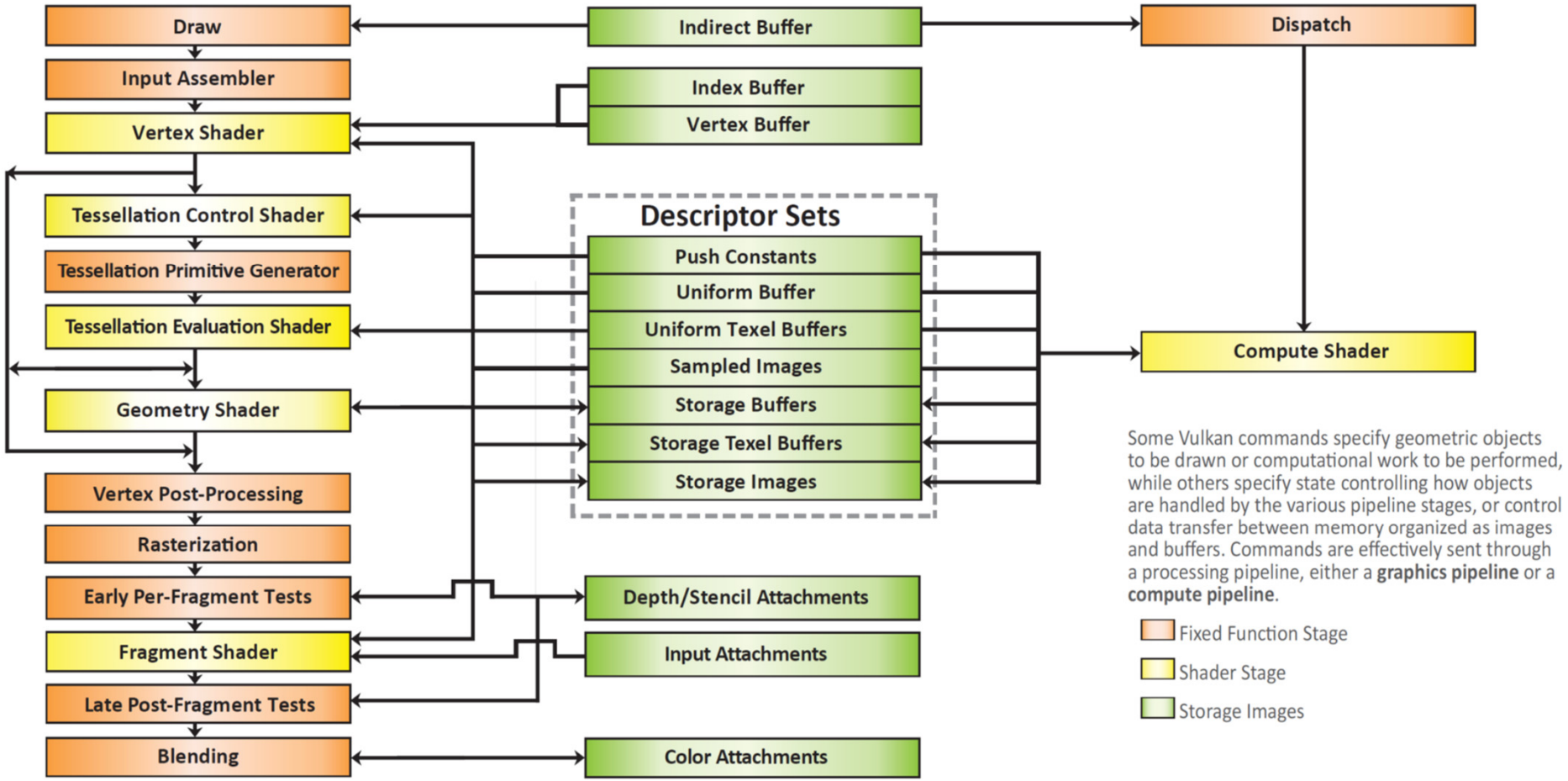
Direct3D 12 Pipeline (and Later Updates)

First version 2015 (Windows 10)

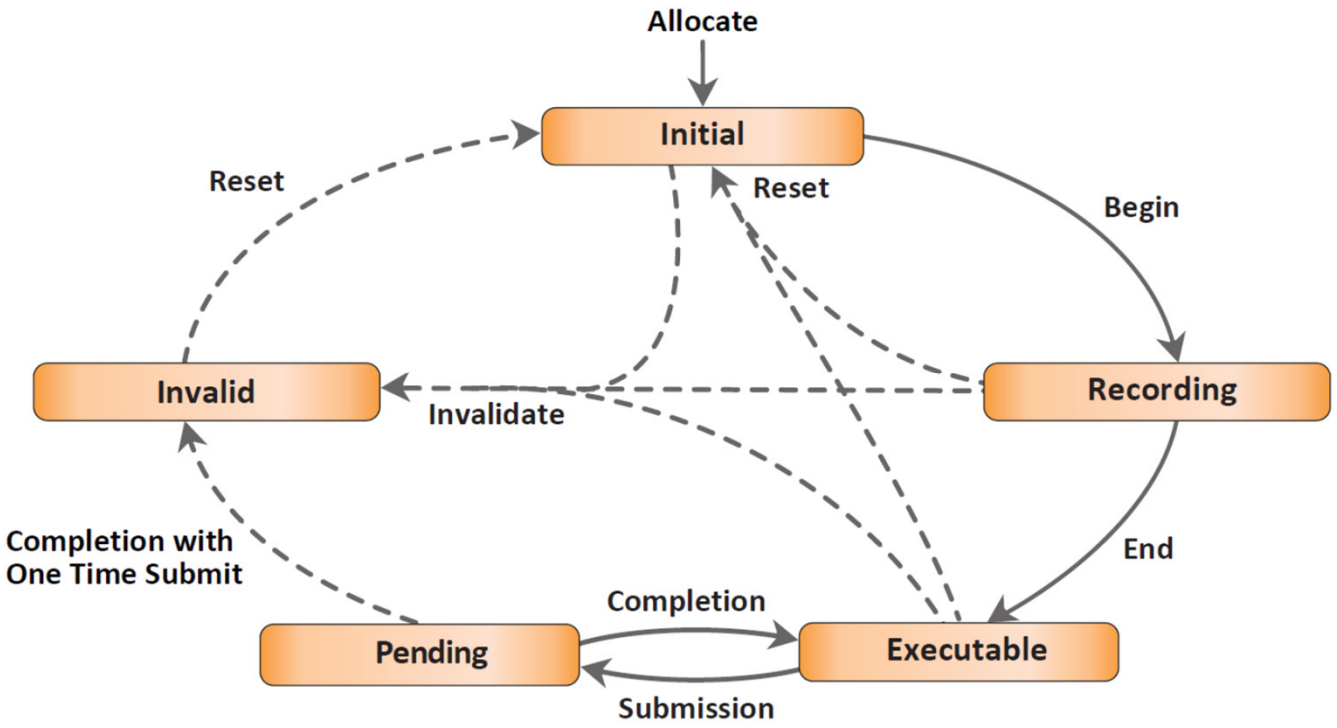
New (March 2018): DXR (DX12 Ray-Tracing)



Vulkan (1.1) Pipeline



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.

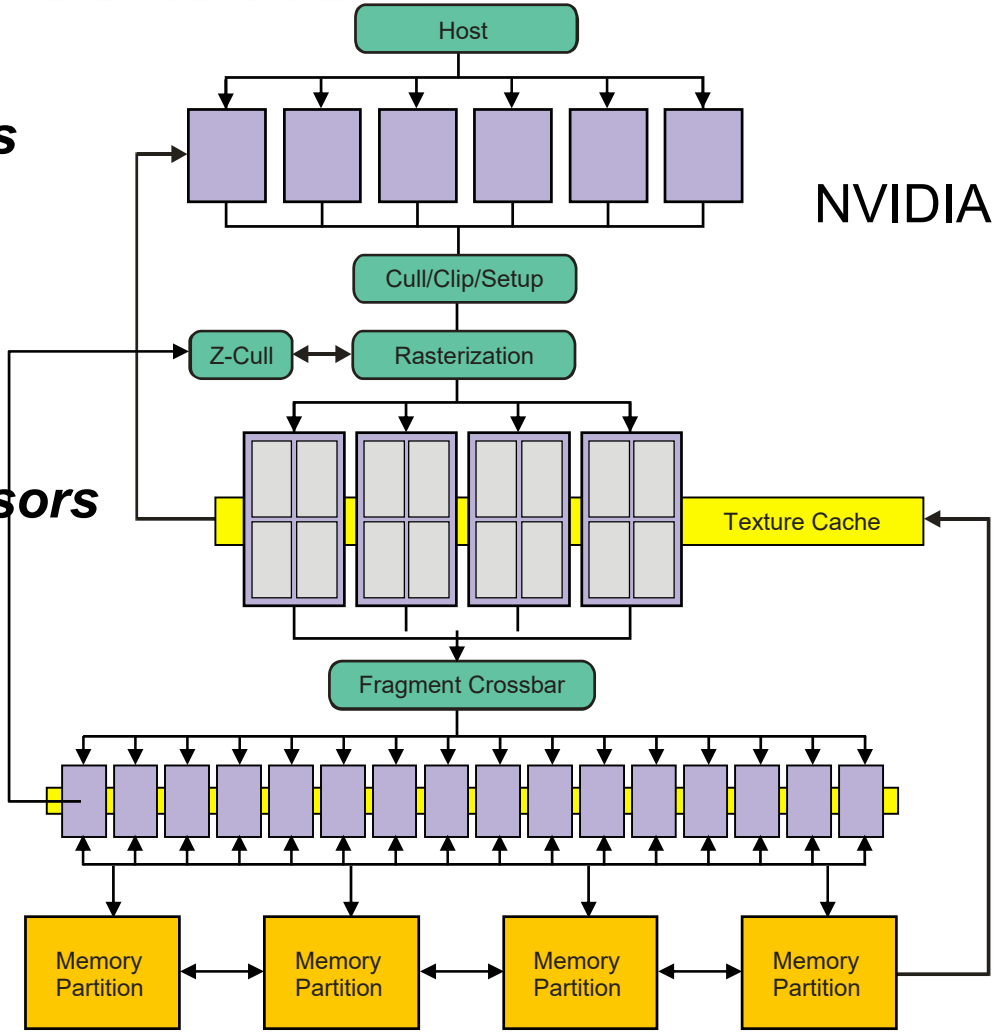
Invalid state
Some operations will transition the command buffer into the invalid state, in which it can only be reset or freed.

GPU Structure Before Unified Shaders

Vertex Processors

Fragment Processors

Memory Access
Z-Compare and Blending

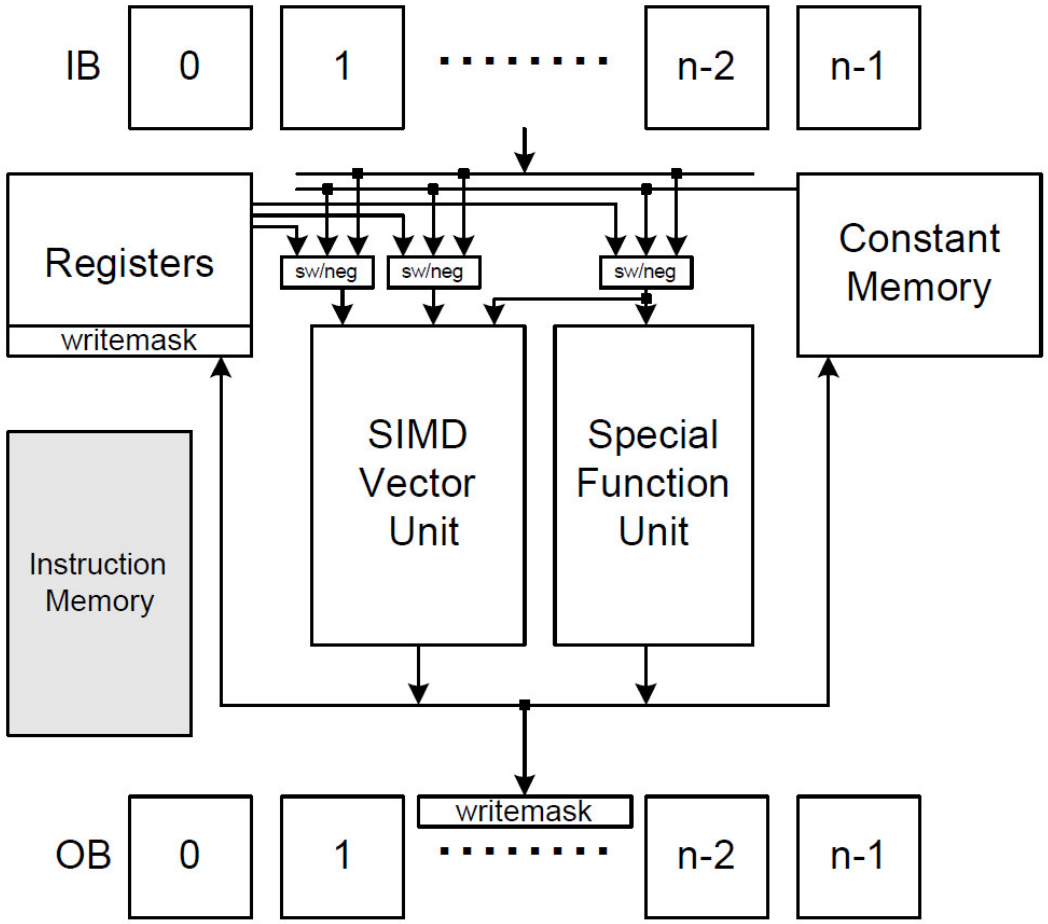


Example
NVIDIA Geforce 6/7,
2004, 2005

Legacy Vertex Shading Unit (1)

Geforce 3 (NV20), 2001

- floating point 4-vector vertex engine
- still very instructive for understanding GPUs in general



Lindholm et al., A User-Programmable Vertex Engine, SIGGRAPH 2001

Legacy Vertex Shading Unit (2)

Input attributes

Vertex Attribute Register	Conventional Per-vertex Parameter	Conventional Per-vertex Parameter Command	Conventional Component Mapping
0	Vertex position	`glVertex`	*x,y,z,w*
1	Vertex weights	`glVertexWeightEXT`	*w,0,0,1*
2	Normal	`glNormal`	
3	Primary color	`glColor`	*r,g,b,a*
4	Secondary color	`glSecondaryColorEXT`	*r,g,b,1*
5	Fog coordinate	`glFogCoordEXT`	*f,0,0,1*
6	-	-	-
7	-	-	-
8	Texture coord 0	`glMultiTexCoordARB(GL_TEXTURE0…)`	*s,t,r,q*
9	Texture coord 1	`glMultiTexCoordARB(GL_TEXTURE1…)`	*s,t,r,q*
10	Texture coord 2	`glMultiTexCoordARB(GL_TEXTURE2…)`	*s,t,r,q*
11	Texture coord 3	`glMultiTexCoordARB(GL_TEXTURE3…)`	*s,t,r,q*
12	Texture coord 4	`glMultiTexCoordARB(GL_TEXTUER4…)`	*s,t,r,q*
13	Texture coord 5	`glMultiTexCoordARB(GL_TEXTUER5…)`	*s,t,r,q*
14	Texture coord 6	`glMultiTexCoordARB(GL_TEXTUER6…)`	*s,t,r,q*
15	Texture coord 7	`glMultiTexCoordARB(GL_TEXTUER7…)`	*s,t,r,q*

Code examples

```
DP4 o[HPOS].x, c[0], v[OPOS];
MUL R1, R0.zxyw, R2.yzxw ;
MAD R1, R0.yzxw, R2.zxyw, -R1;
```

swizzling!

Legacy Vertex Shading Unit (3)

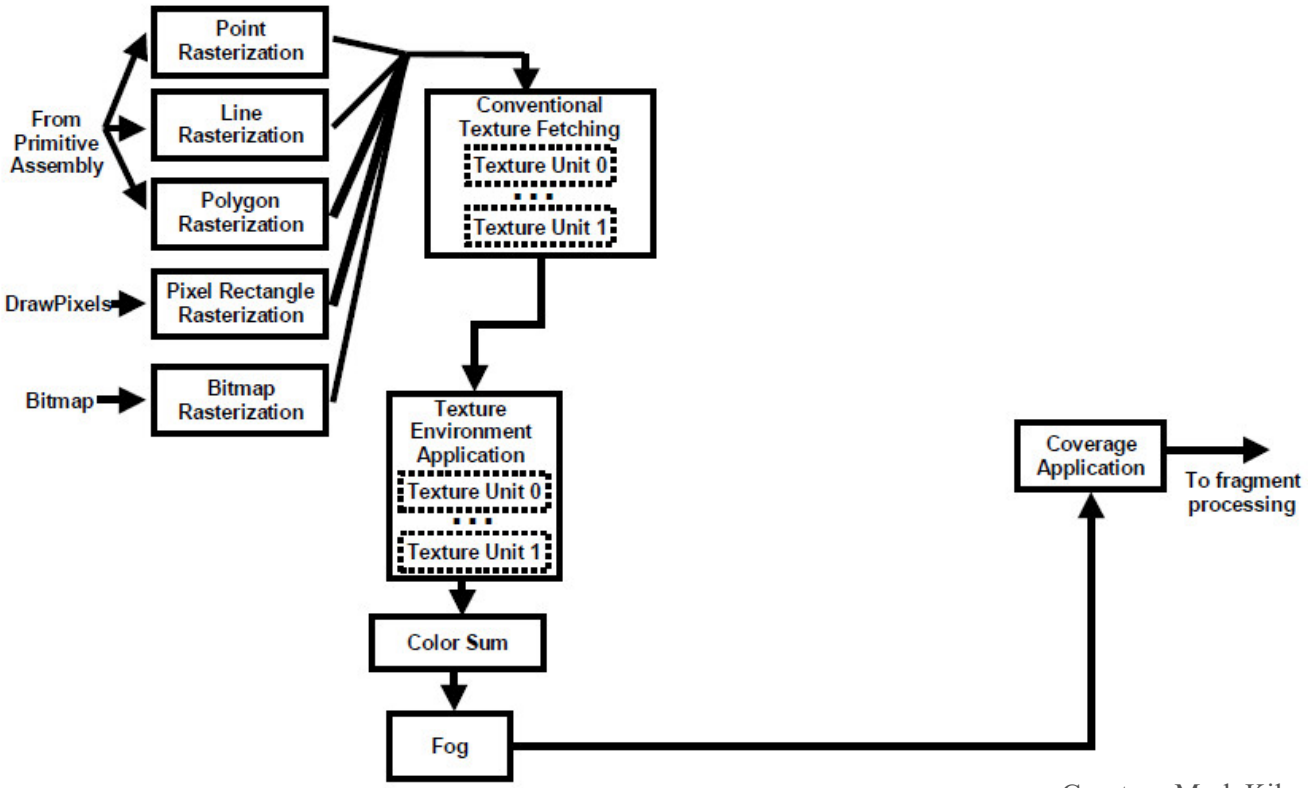
Vector instruction set, very few instructions; no branching yet!

OpCode	Full Name	Description
MOV	Move	vector -> vector
MUL	Multiply	vector -> vector
ADD	Add	vector -> vector
MAD	Multiply and add	vector -> vector
DST	Distance	vector -> vector
MIN	Minimum	vector -> vector
MAX	Maximum	vector -> vector
SLT	Set on less than	vector -> vector
SGE	Set on greater or equal	vector -> vector
RCP	Reciprocal	scalar-> replicated scalar
RSQ	Reciprocal square root	scalar-> replicated scalar
DP3	3 term dot product	vector-> replicated scalar
DP4	4 term dot product	vector-> replicated scalar
LOG	Log base 2	miscellaneous
EXP	Exp base 2	miscellaneous
LIT	Phong lighting	miscellaneous
ARL	Address register load	miscellaneous

Fast Forward to Programm. Fragment Shading

Core OpenGL Fragment Texturing & Coloring

< 1999



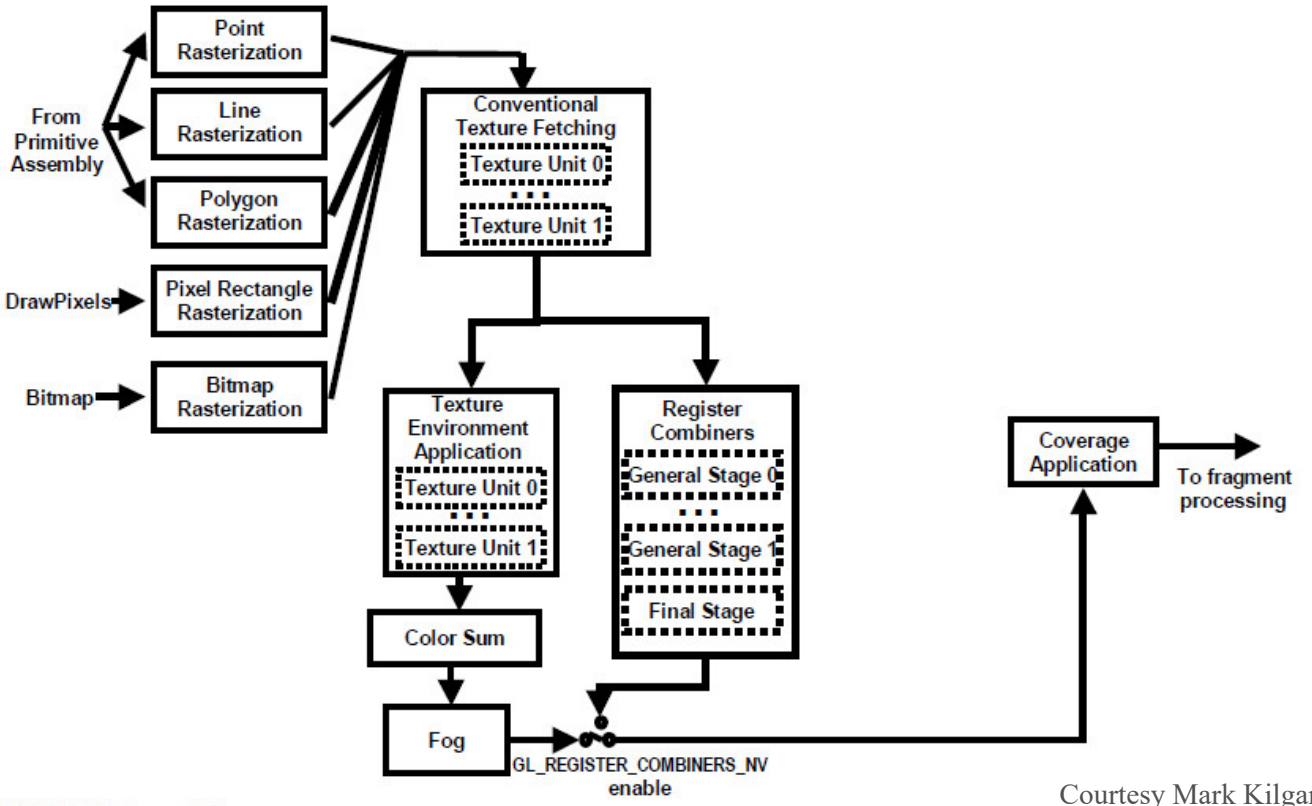
Courtesy Mark Kilgard

NVIDIA Proprietary

Fast Forward to Programm. Fragment Shading

NV10 OpenGL Fragment Texturing & Coloring

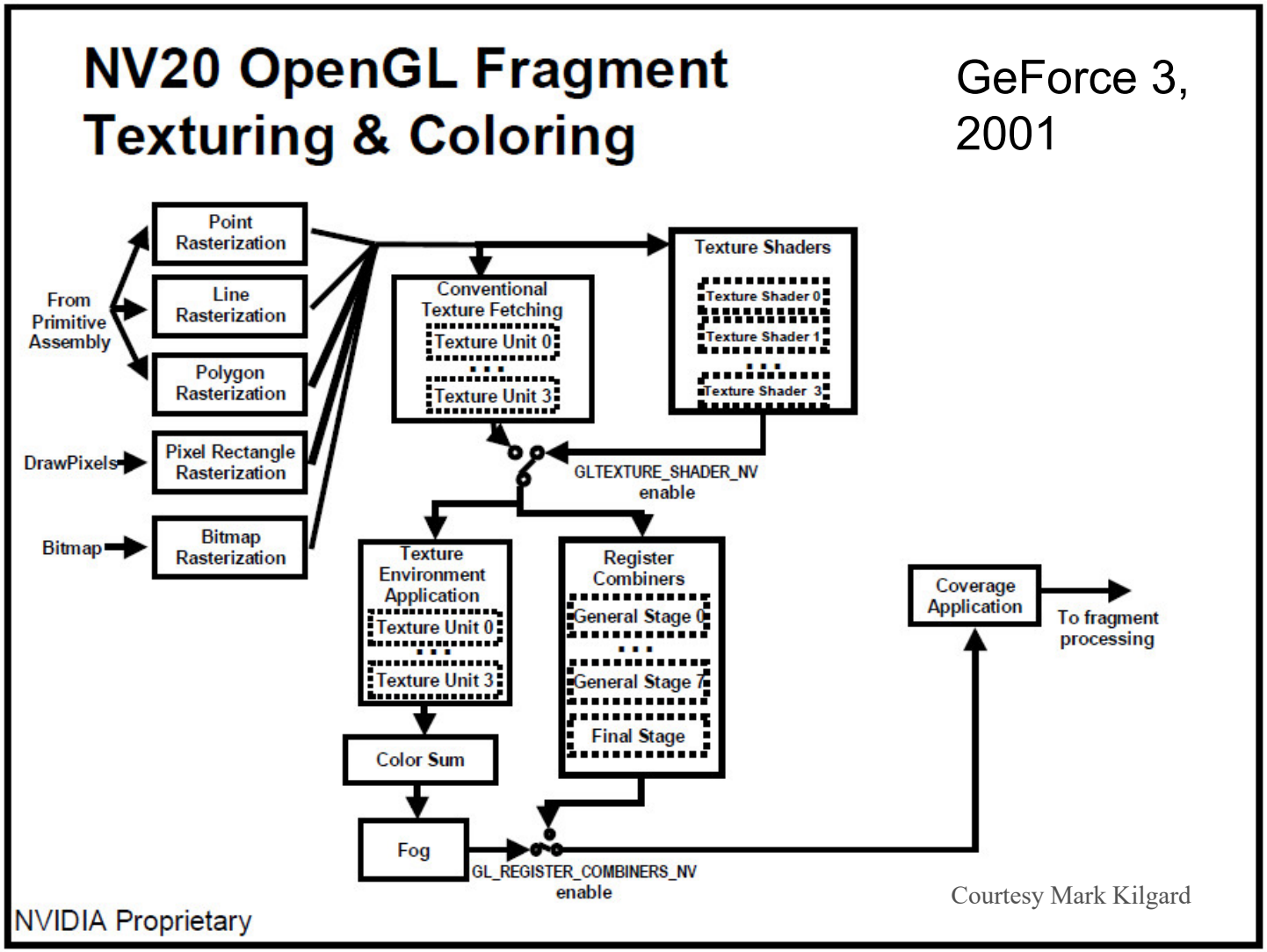
GeForce 256, 1999



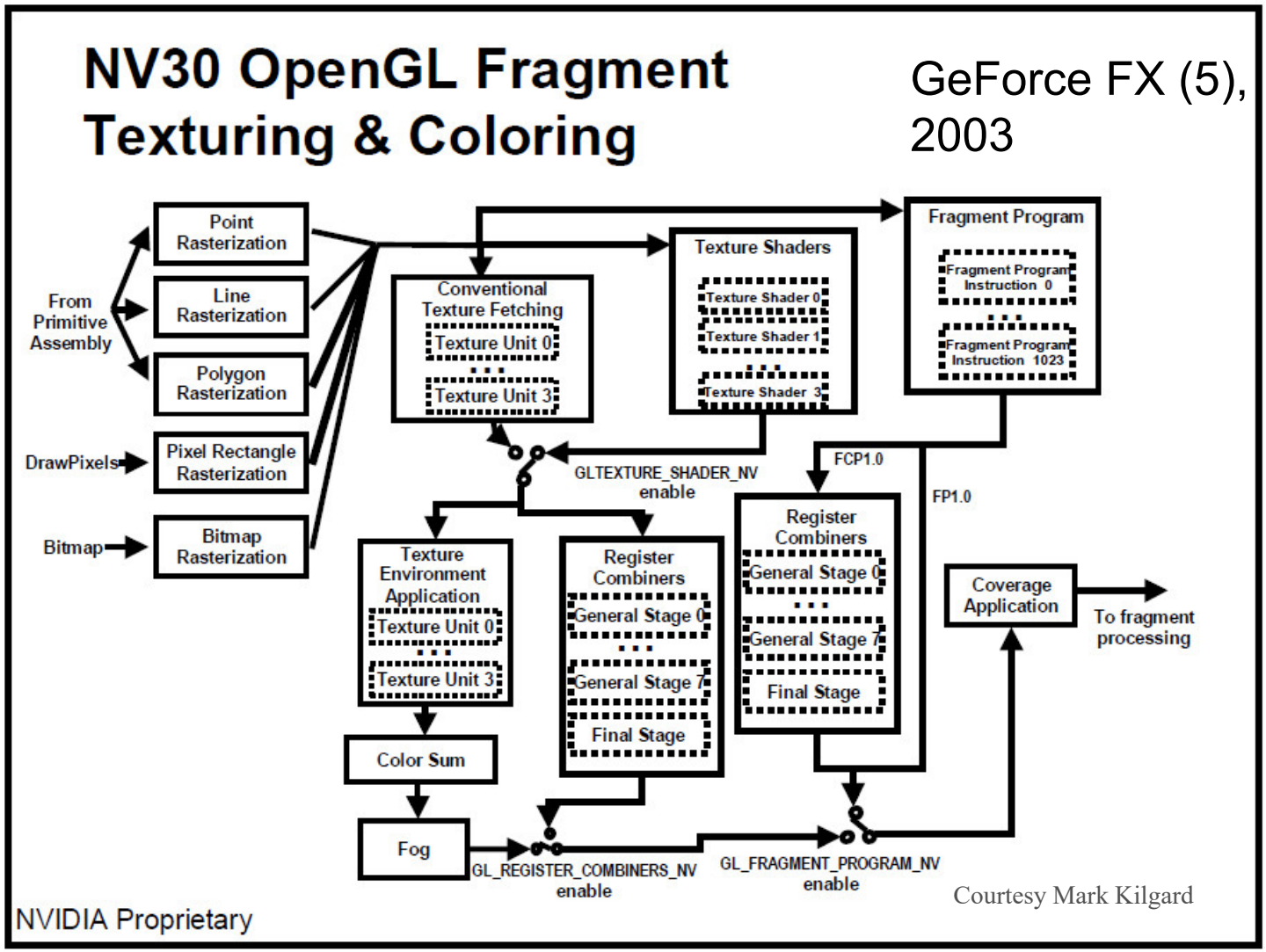
NVIDIA Proprietary

Courtesy Mark Kilgard

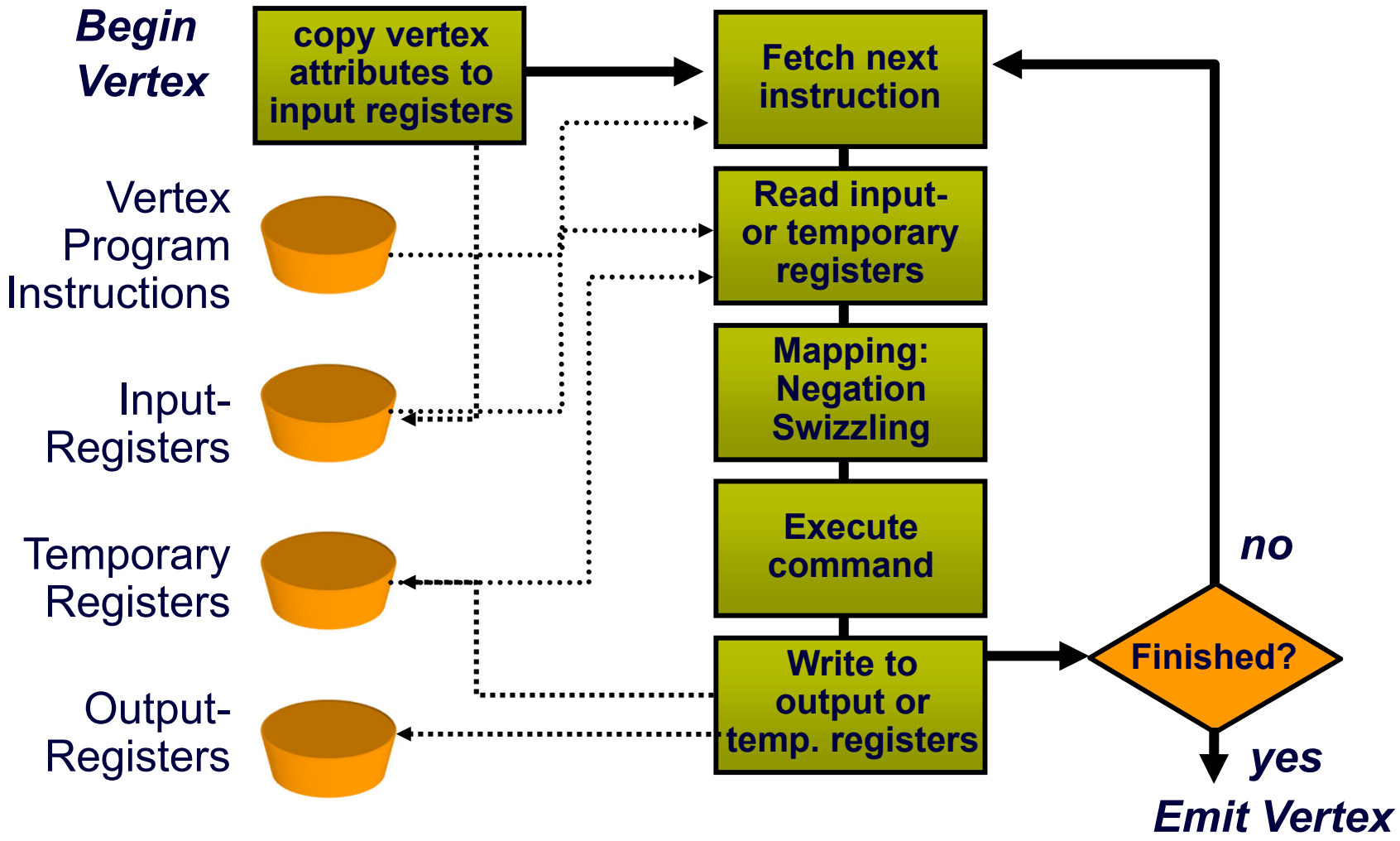
Fast Forward to Programm. Fragment Shading



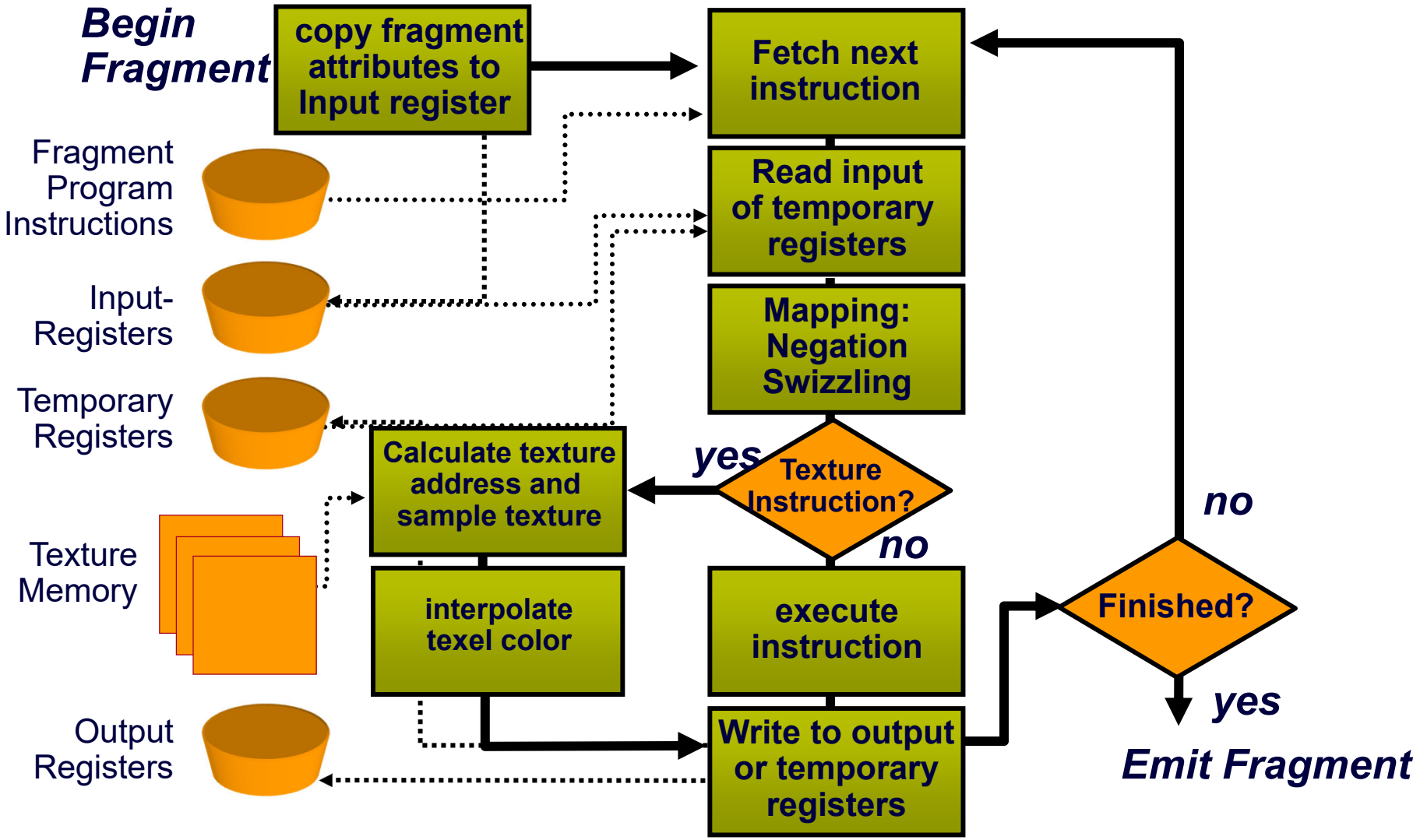
Fast Forward to Programm. Fragment Shading



Vertex Processor



Fragment Processor



A diffuse reflectance shader

```
sampler mySamp;
Texture2D<float3> myTex;
float3 lightDir;

float4 diffuseShader(float3 norm, float2 uv)
{
  float3 kd;
  kd = myTex.Sample(mySamp, uv);
  kd *= clamp( dot(lightDir, norm), 0.0, 1.0);
  return float4(kd, 1.0);
}
```

Independent, but no explicit parallelism

Thank you.