King Abdullah University of Science and Technology

KAUST

CS 380 - GPU and GPGPU Programming
Lecture 4: GPU Architecture, Pt. 1

Markus Hadwiger, KAUST

Reading Assignment #2 (until Sep 13)

Read (required):

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

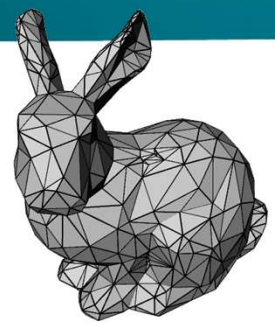
OpenGL Tutorial (Attendance Optional)

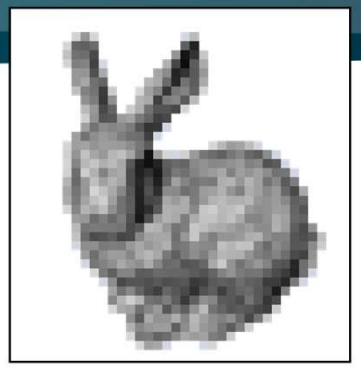
With Amani

Tomorrow, 15:00 – 16:30, Room 3223

Come with your conceptual or coding questions!

Graphics Hardware





Scene Description

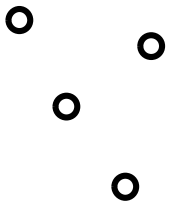
Programmable Pipeline

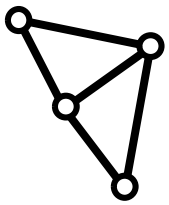
Raster Image

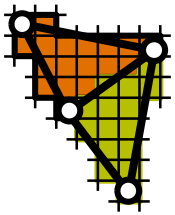
Vertex Shader

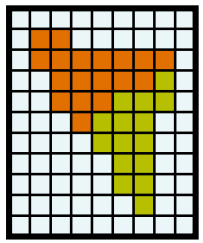
Fragment Shader

Fragment Operations









Vertices

Primitives

Fragments

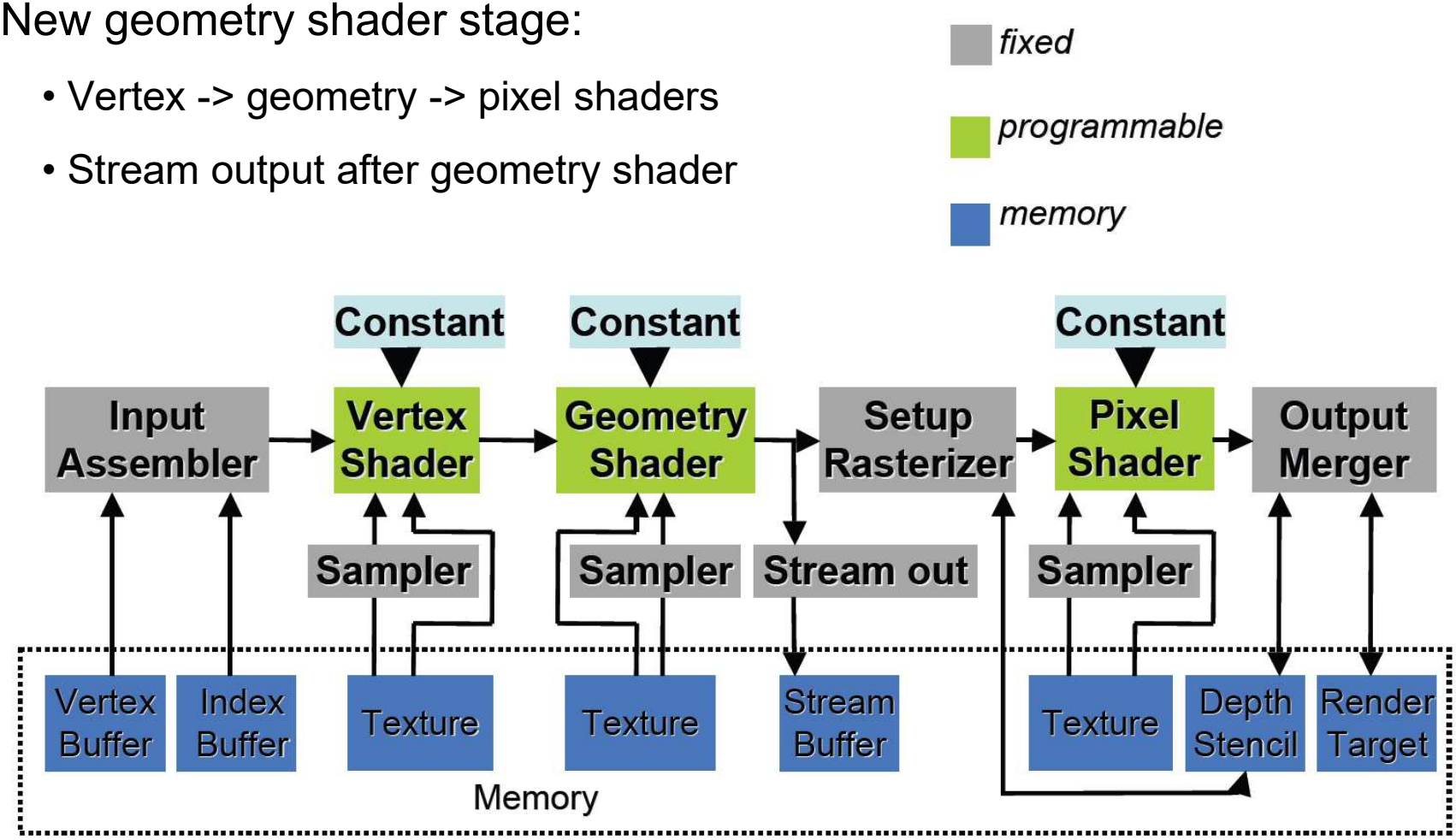
Pixels

ROPs = raster operations
(render output units)

Direct3D 10 Pipeline (~OpenGL 3.2)

New geometry shader stage:

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



Courtesy David Blythe, Microsoft

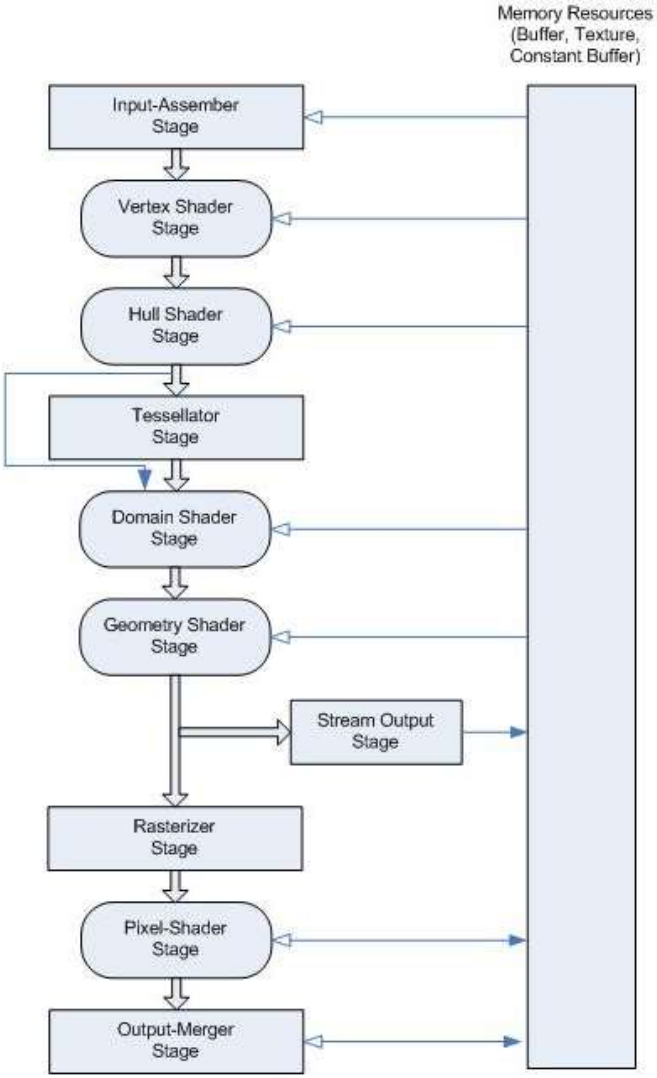
Direct3D 11 Pipeline (~OpenGL 4.x)

New tessellation stages

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

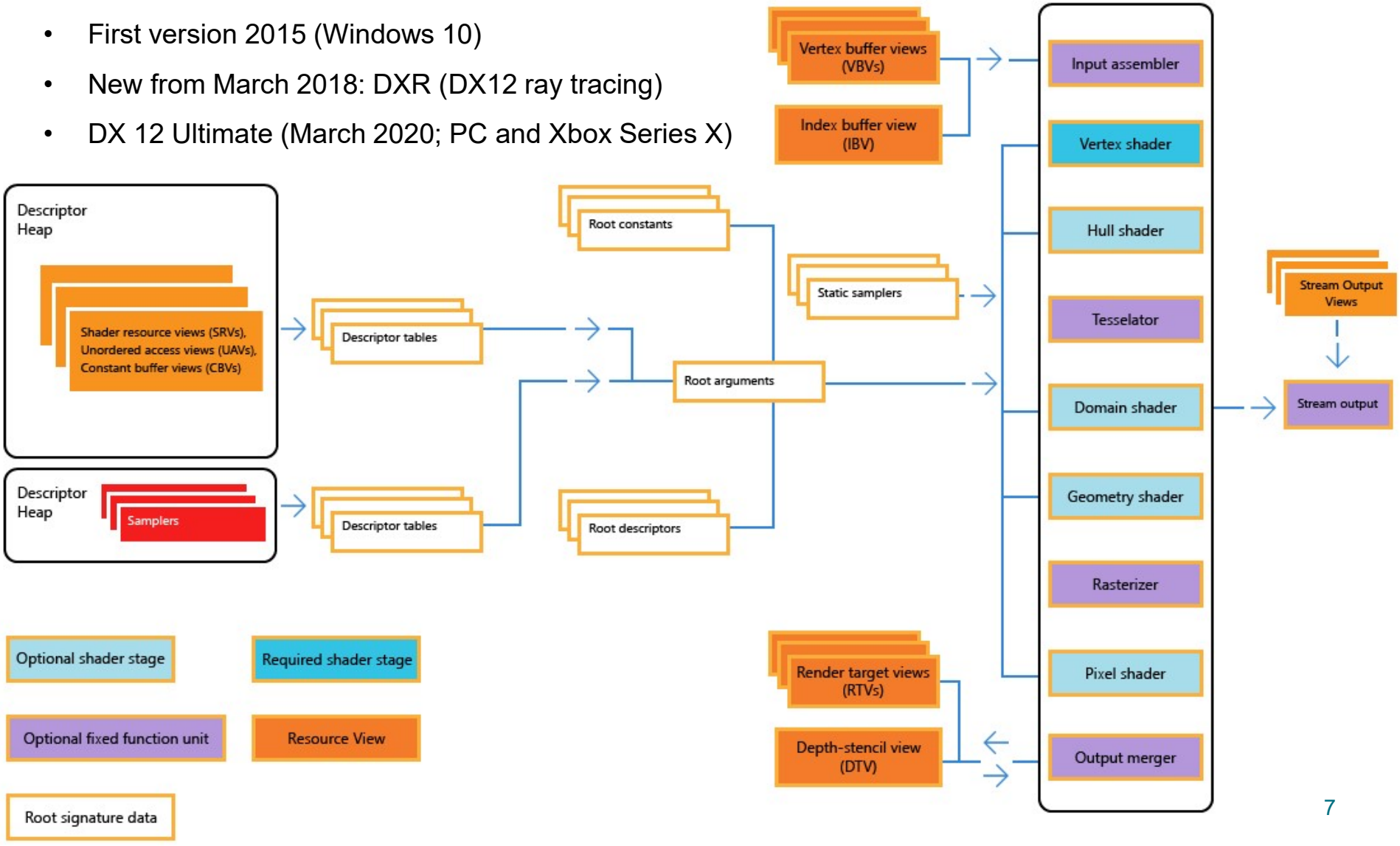
Outside this pipeline

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



Direct3D 12 Traditional Geometry Pipeline

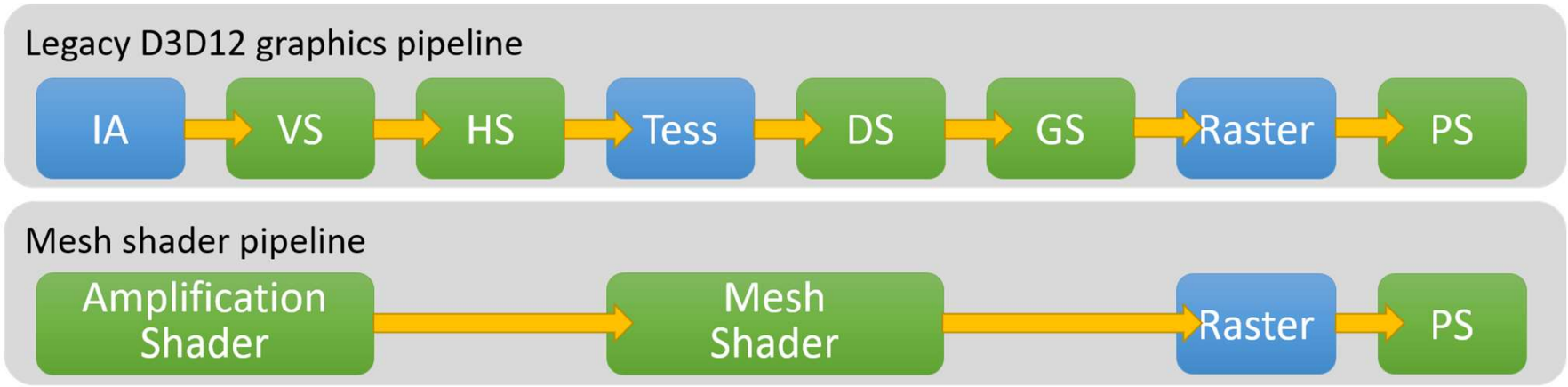
- First version 2015 (Windows 10)
- New from March 2018: DXR (DX12 ray tracing)
- DX 12 Ultimate (March 2020; PC and Xbox Series X)



Direct3D 12 Mesh Shader Pipeline

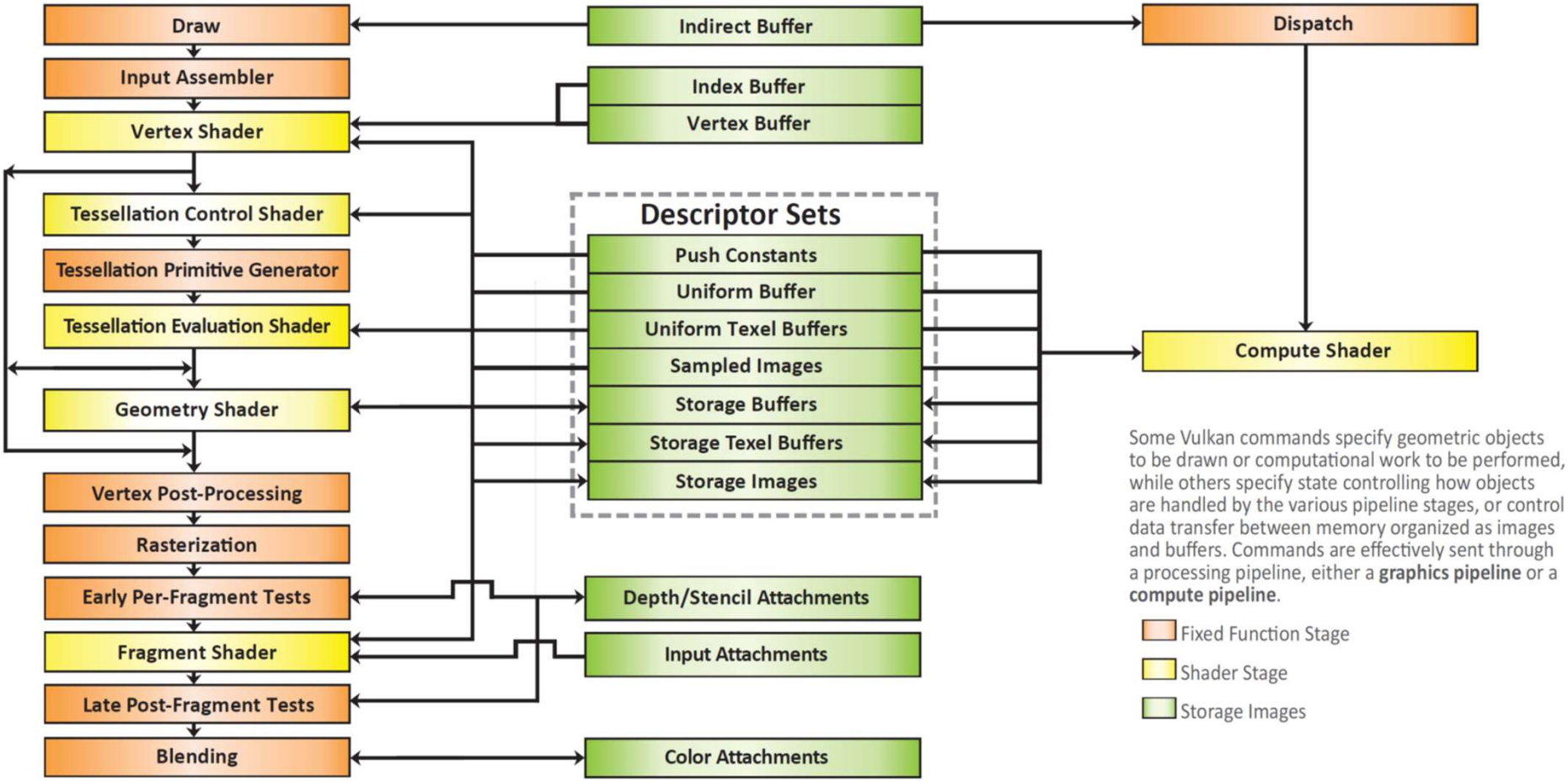
Reinventing the Geometry Pipeline

- 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



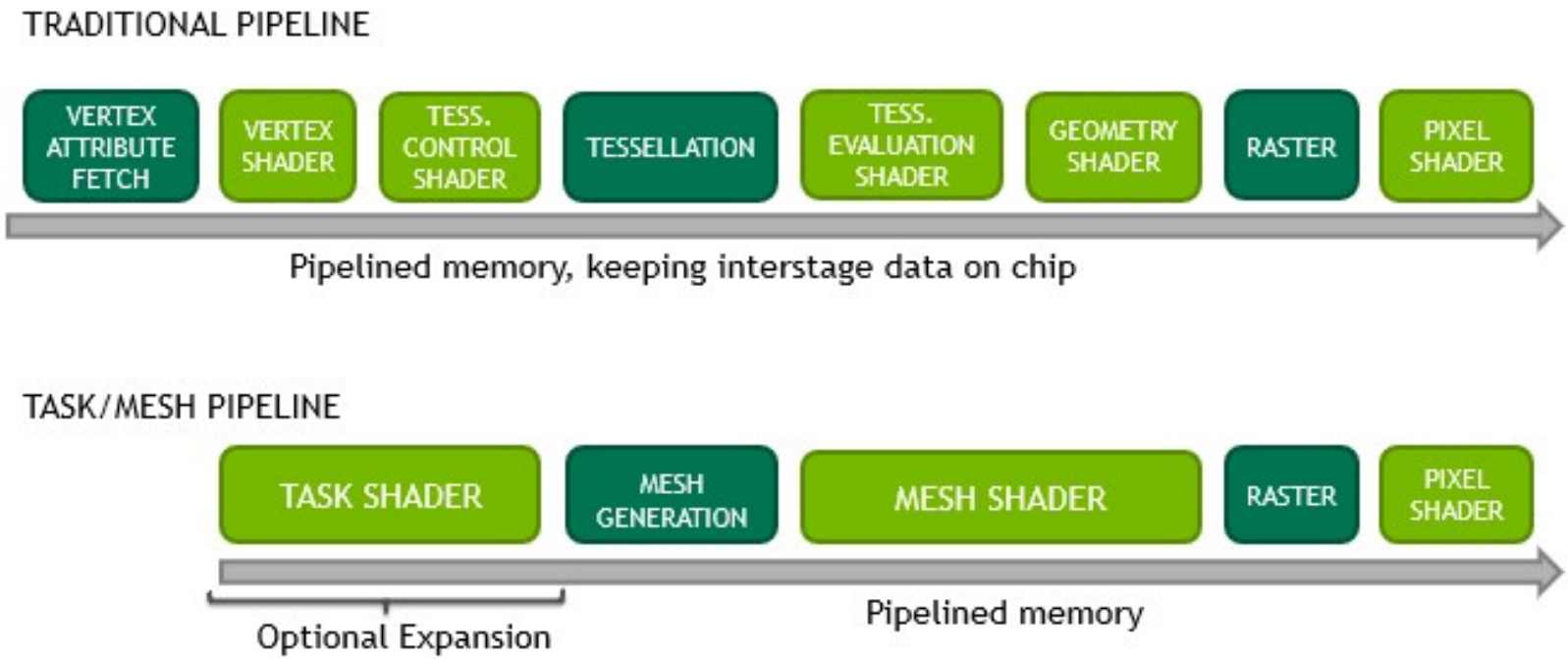
See talk by Shawn Hargreaves: `https://www.youtube.com/watch?v=CFXKTXtil34`

Vulkan 1.1/1.2

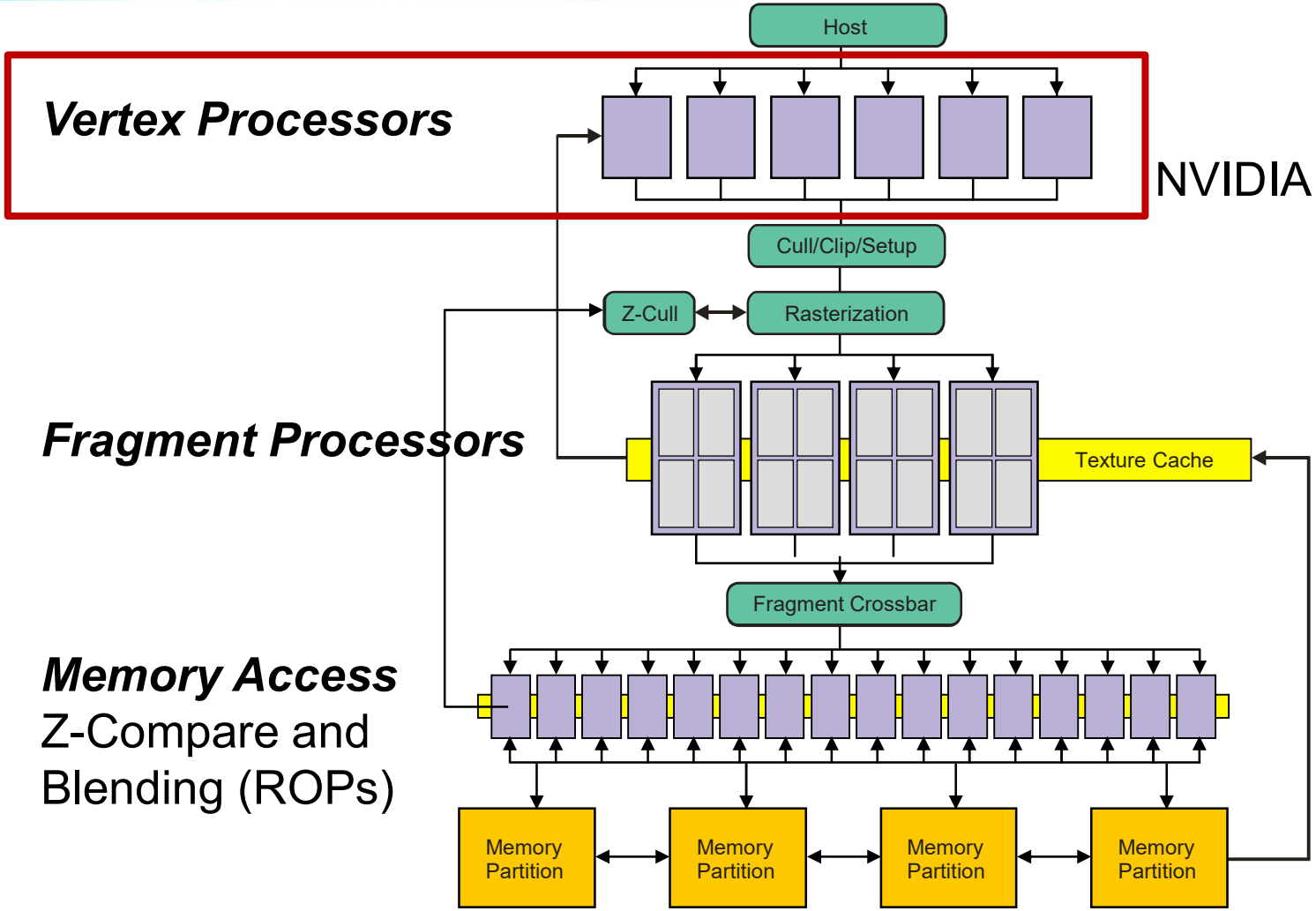

Draw
Input Assembler
Vertex Shader
Tessellation Control Shader
Tessellation Primitive Generator
Tessellation Evaluation Shader
Geometry Shader
Vertex Post-Processing
Rasterization
Early Per-Fragment Tests
Fragment Shader
Late Post-Fragment Tests
Blending
Indirect Buffer
Index Buffer
Vertex Buffer
Descriptor 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 1.1/1.2

- 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`)



GPU Structure *Before Later* Unified Shaders

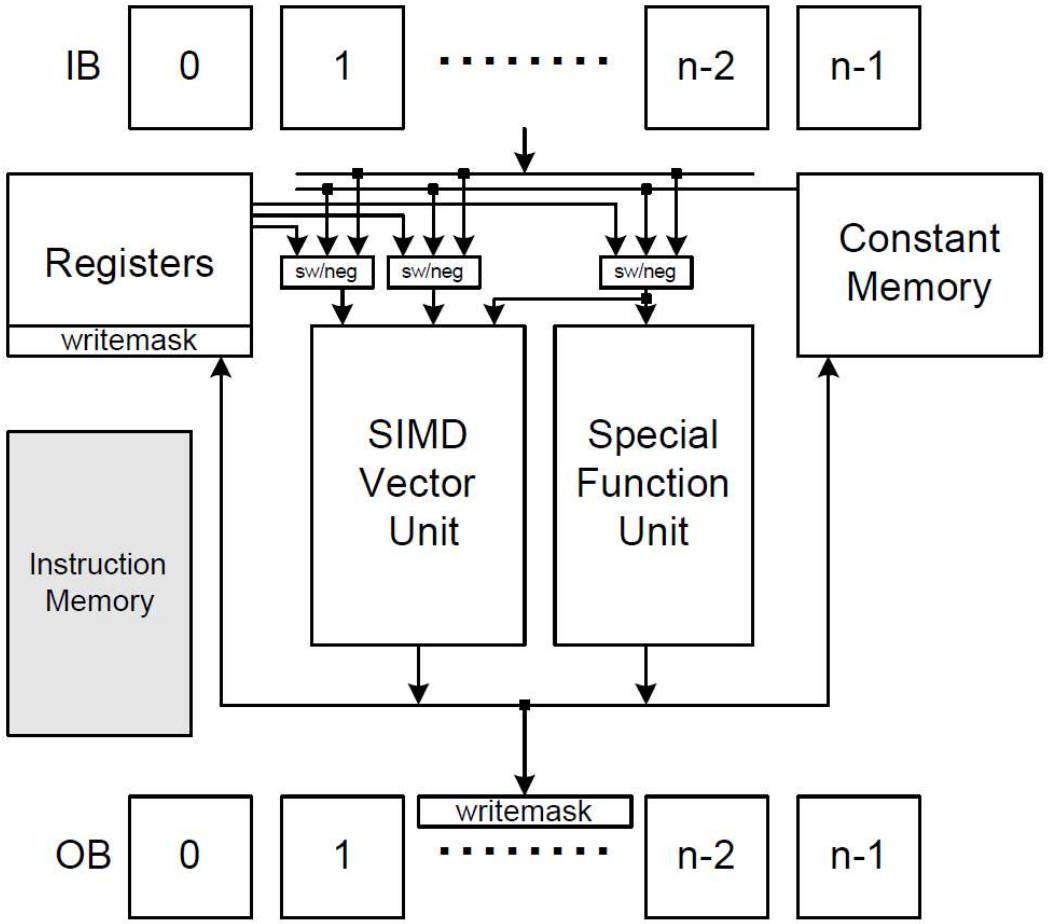


Example
NVIDIA Geforce 6/7,
2004, 2005

Legacy Vertex Shading Unit (1)

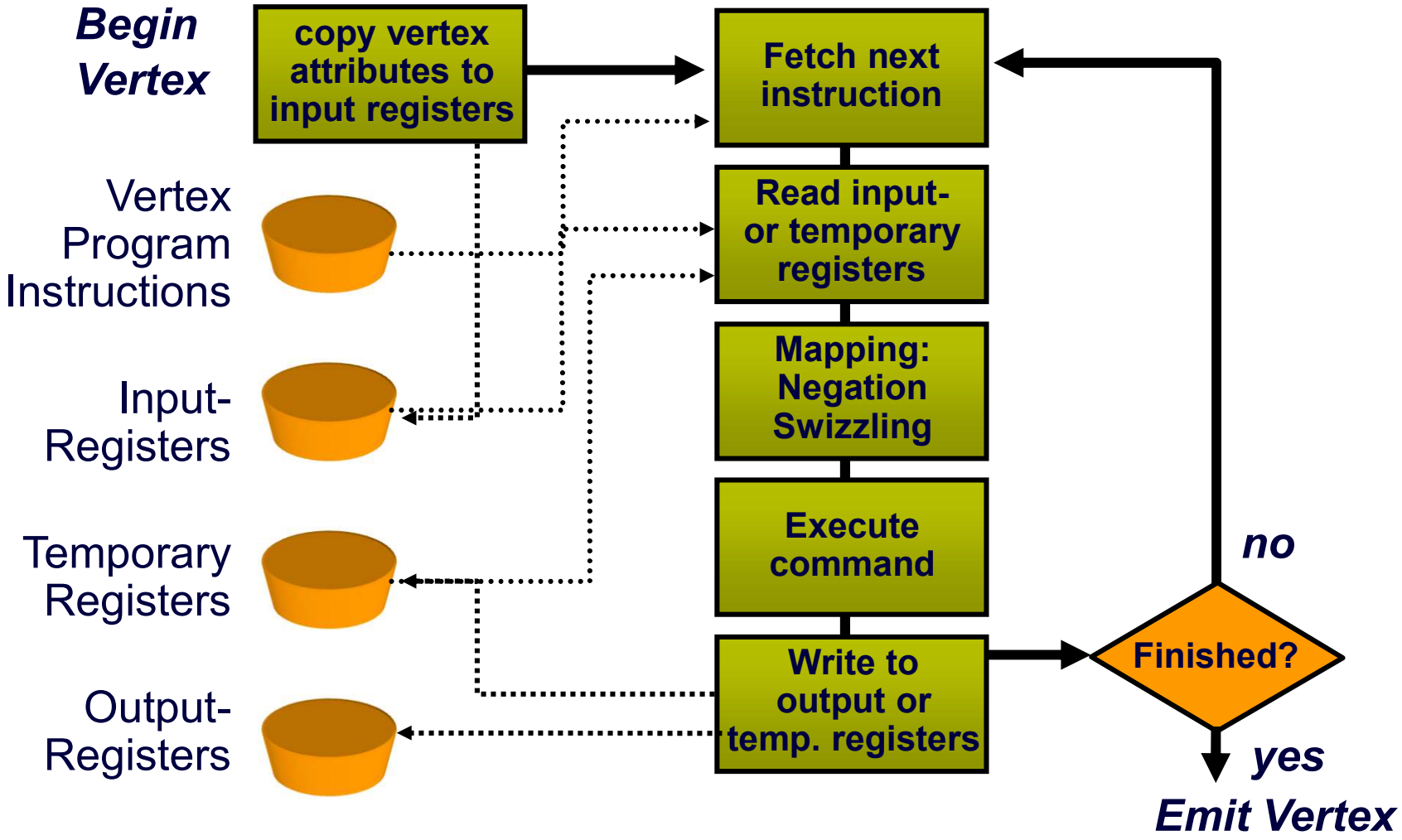
Geforce 3 (NV20), 2001

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



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

Vertex Processor



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;
```

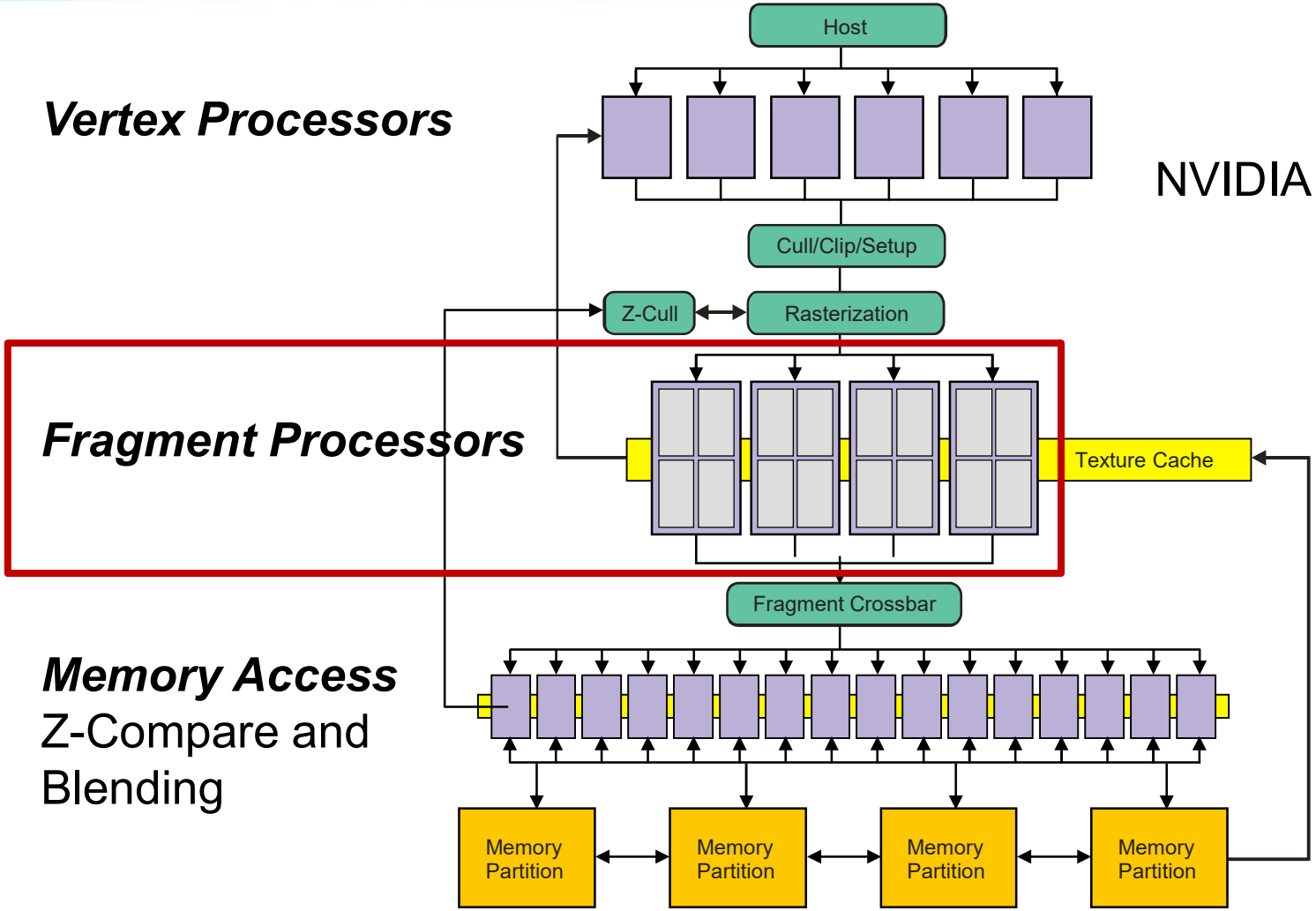
swizzling!

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

GPU Structure *Before Later* Unified Shaders

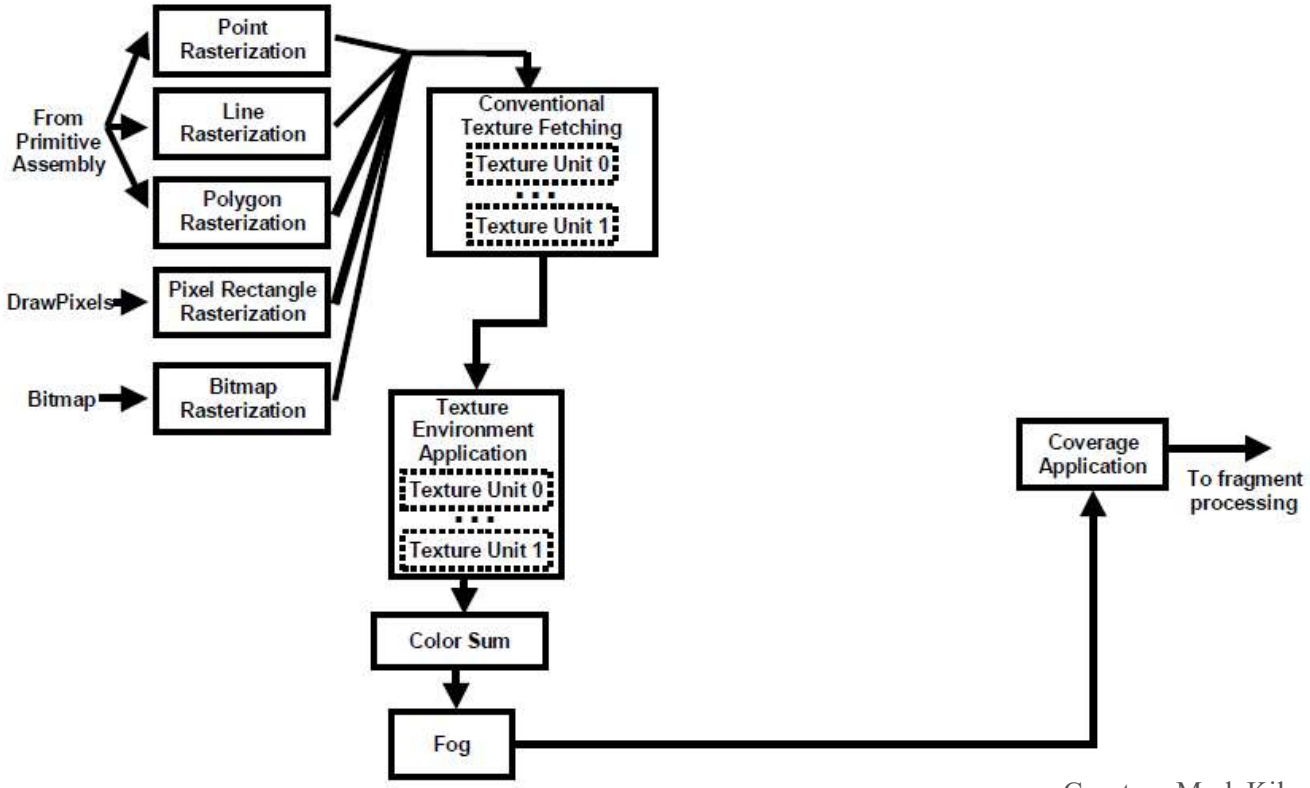


Example
NVIDIA Geforce 6/7,
2004, 2005

Fast Forward to Programm. Fragment Shading

Core OpenGL Fragment Texturing & Coloring

< 1999



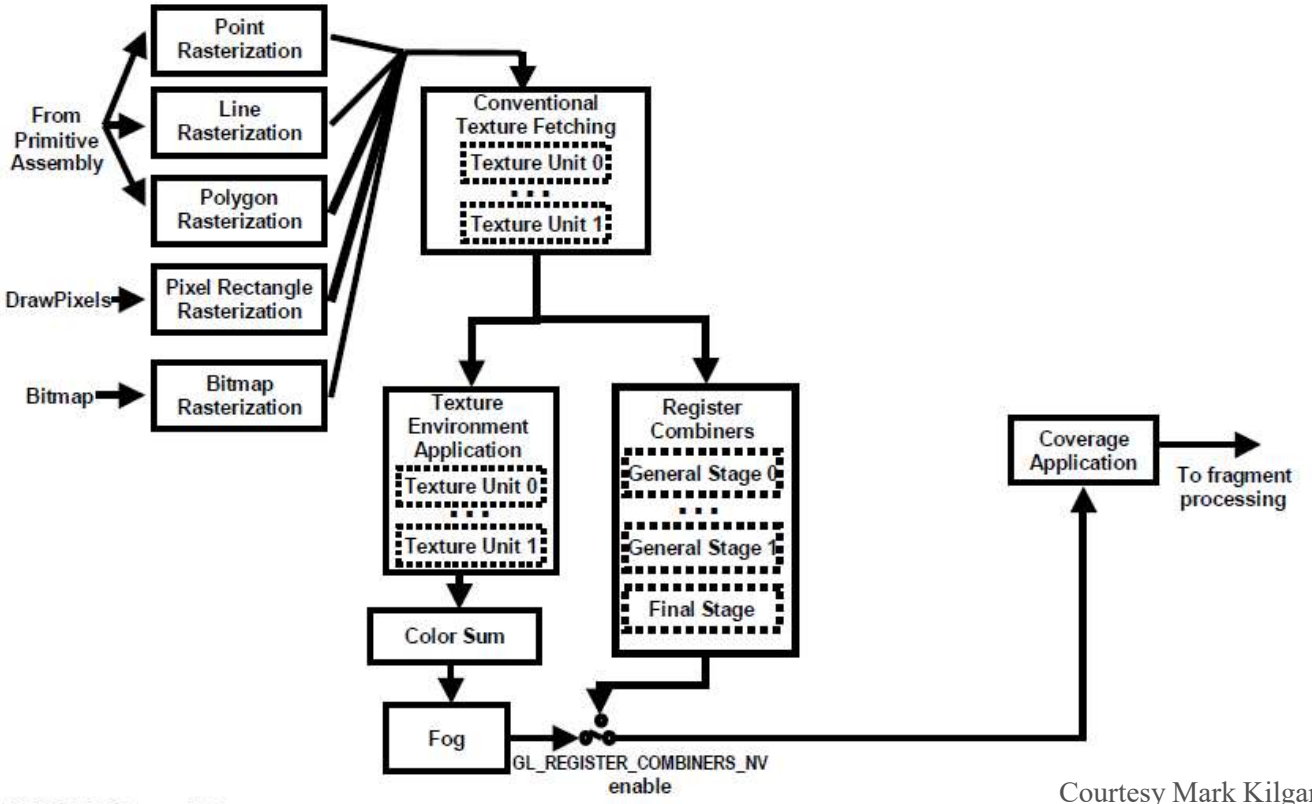
Courtesy Mark Kilgard

NVIDIA Proprietary

Fast Forward to Programm. Fragment Shading

NV10 OpenGL Fragment Texturing & Coloring

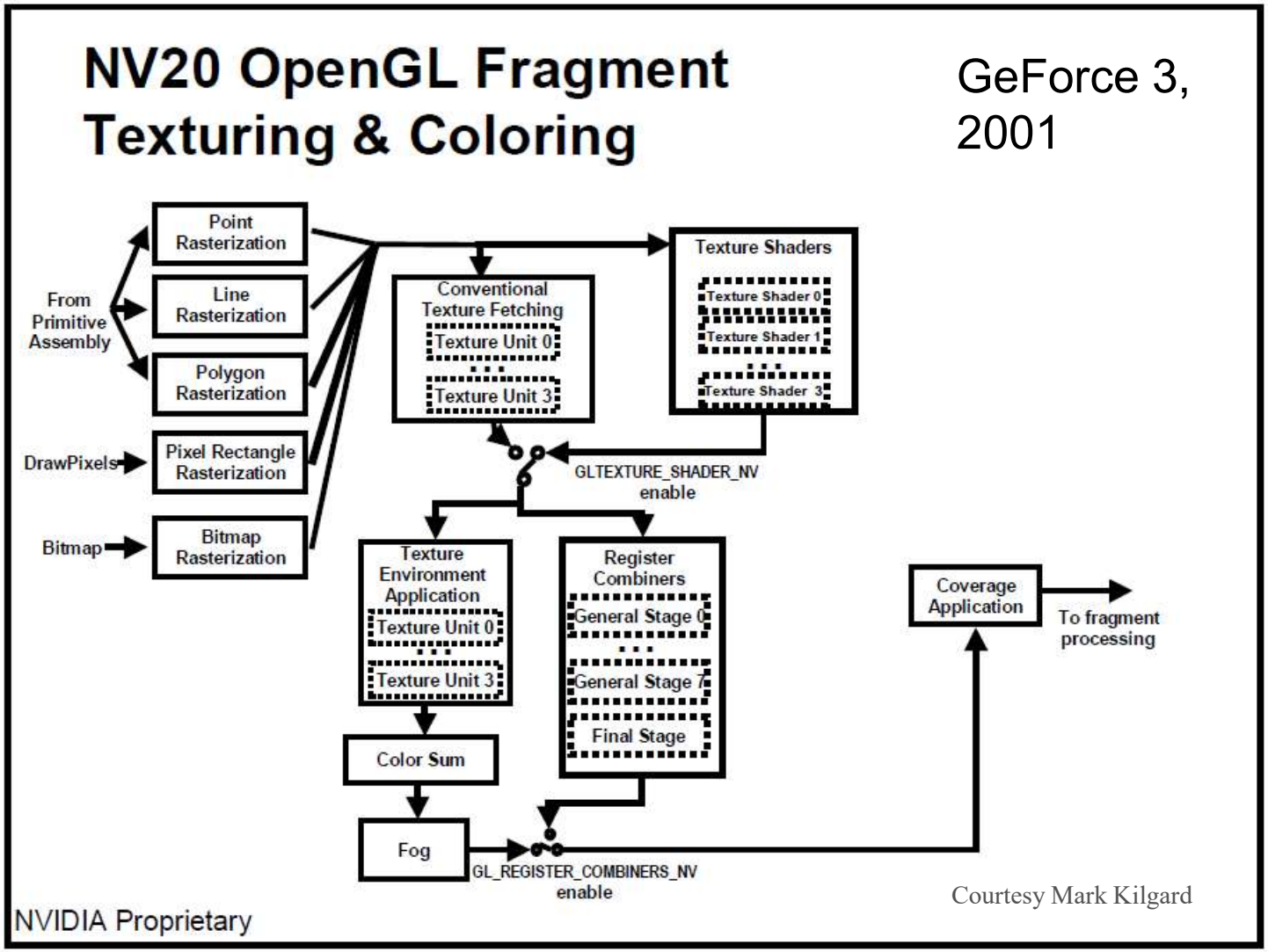
GeForce 256, 1999



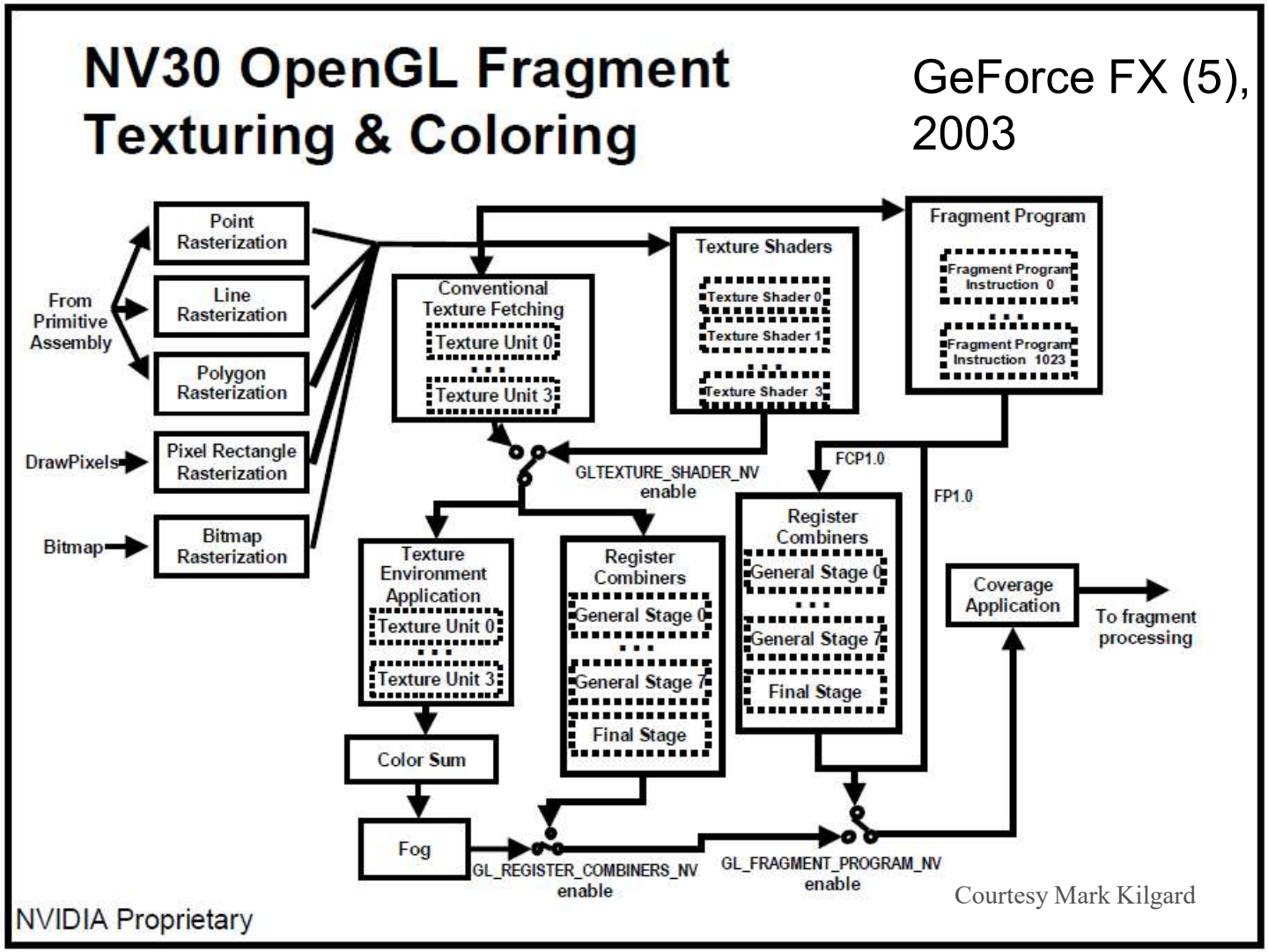
NVIDIA Proprietary

Courtesy Mark Kilgard

Fast Forward to Programm. Fragment Shading



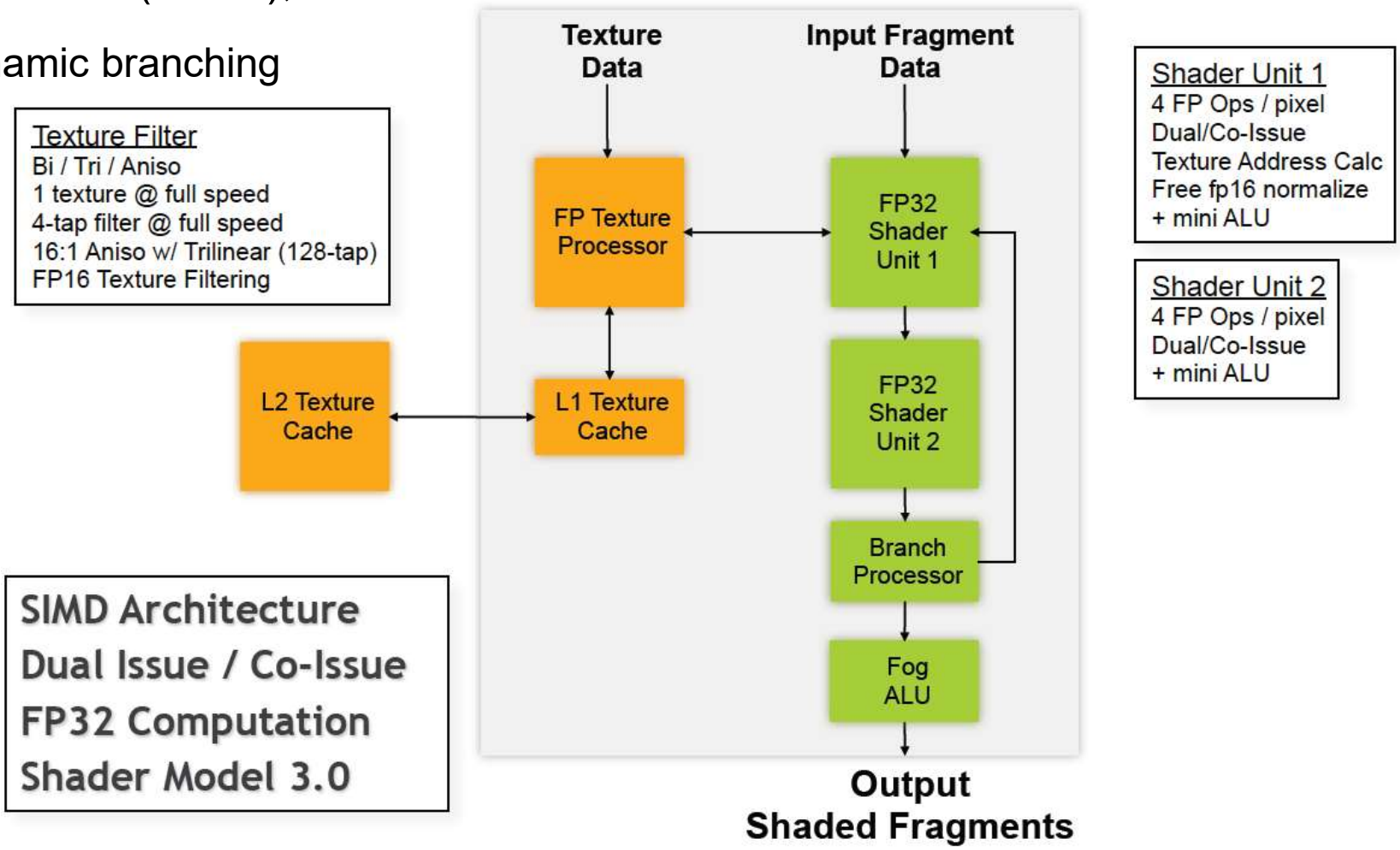
Fast Forward to Programm. Fragment Shading



Legacy Fragment Shading Unit (1)

GeForce 6 (NV40), 2004

- dynamic branching

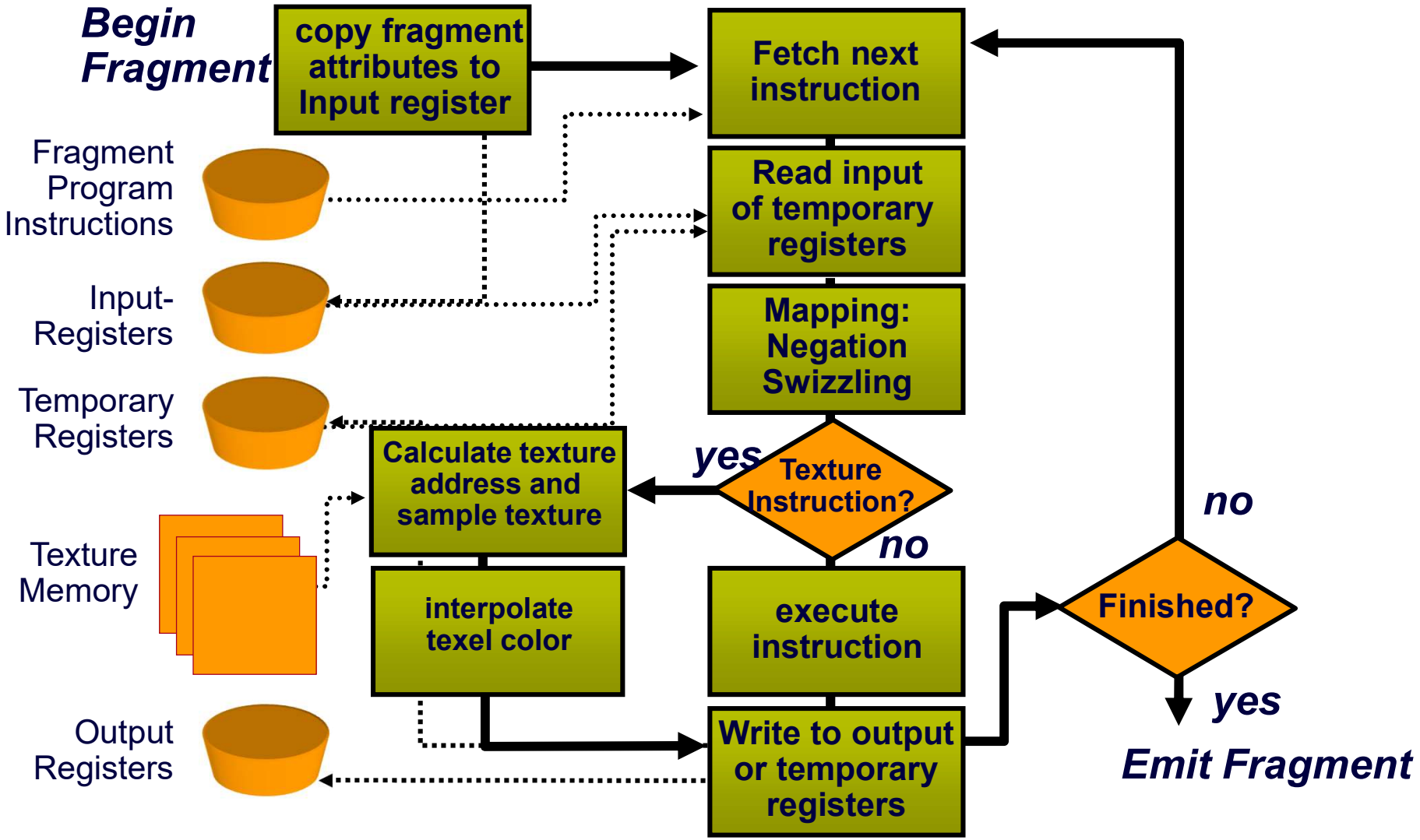


Legacy Fragment Shading Unit (2)

Example code

```
!!ARBfp1.0

ATTRIB unit_tc = fragment.texcoord[ 0 ];
PARAM  mvp_inv[]  = { state.matrix.mvp.inverse };
PARAM  constants  = {0, 0.999, 1, 2};

TEMP pos_win, temp;

TEX pos_win.z, unit_tc, texture[ 1 ], 2D;

ADD pos_win.w, constants.y, -pos_win.z;
KIL pos_win.w;

MOV result.color.w, pos_win.z;

MOV pos_win.xyw, unit_tc;
MAD pos_win.xyz, pos_win, constants.a, -constants.b;

DP4 temp.w, mvp_inv[ 3 ], pos_win;
RCP temp.w, temp.w;

MUL pos_win, pos_win, temp.w;

DP4 result.color.x, mvp_inv[ 0 ], pos_win;
DP4 result.color.y, mvp_inv[ 1 ], pos_win;
DP4 result.color.z, mvp_inv[ 2 ], pos_win;

END
```

Fragment Processor



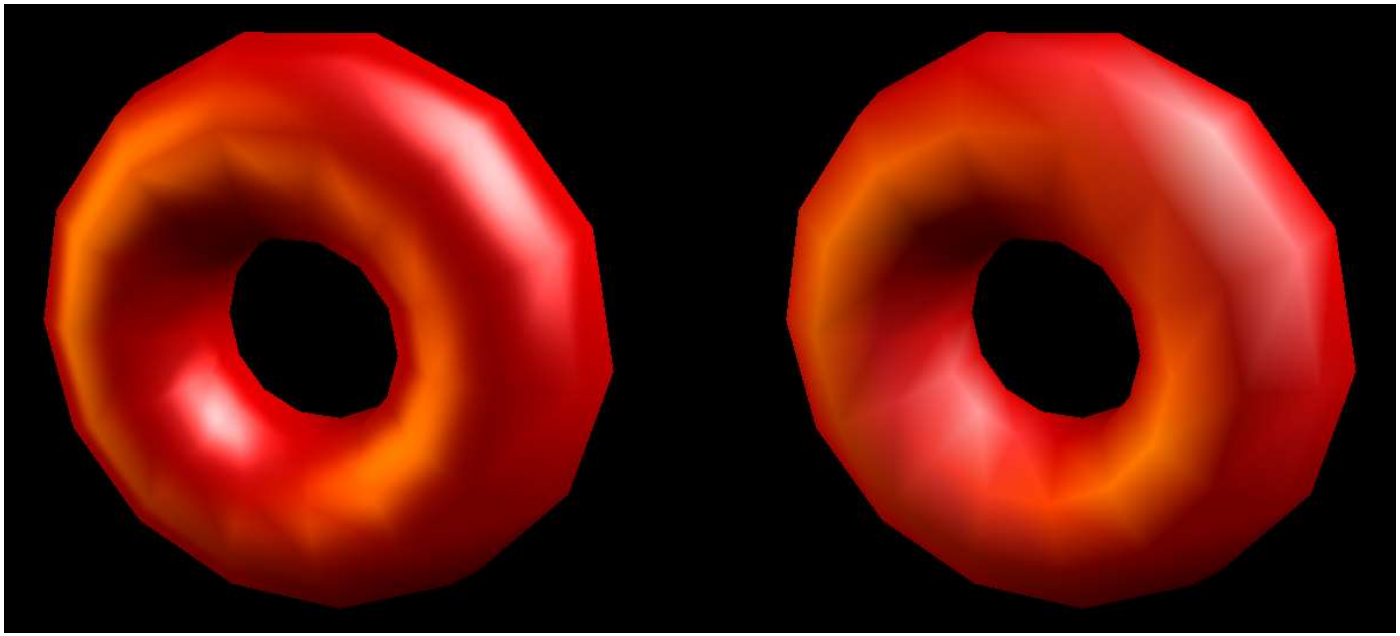
Per-Pixel(Fragment) Lighting

Simulating smooth surfaces by calculating illumination for each fragment

Example: specular highlights (Phong illumination/shading)

fragment shader! Phong shading: per-fragment evaluation

Gouraud shading: linear interpolation from vertices **vertex shader!**



Per-Pixel Phong Lighting (Cg)

```
void main(float4 position  : TEXCOORD0,
          float3 normal    : TEXCOORD1,

      out float4 oColor     : COLOR,

  uniform float3 ambientCol,
  uniform float3 lightCol,
  uniform float3 lightPos,
  uniform float3 eyePos,
  uniform float3 Ka,
  uniform float3 Kd,
  uniform float3 Ks,
  uniform float  shiny)
{
```

Per-Pixel Phong Lighting (Cg)

```
  float3 P = position.xyz;
  float3 N = normal;
  float3 V = normalize(eyePosition - P);
  float3 H = normalize(L + V);

  float3 ambient = Ka * ambientCol;

  float3 L         = normalize(lightPos - P);
  float  diffLight = max(dot(L, N), 0);
  float3 diffuse   = Kd * lightCol * diffLight;

  float specLight = pow(max(dot(H, N), 0), shiny);
  float3 specular = Ks * lightCol * specLight;

  oColor.xyz = ambient + diffuse + specular;
  oColor.w = 1;
}
```

Thank you.