

# **CS 380 - GPU and GPGPU Programming**

## **Lecture 3: GPU Architecture 2**

Markus Hadwiger, KAUST

# Reading Assignment #2 (until Feb. 17)



Read (required):

- GLSL book, chapter 4 (*The OpenGL Programmable Pipeline*)
- 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](http://download.nvidia.com/developer/GPU_Gems_2/GPU_Gems2_ch30.pdf)

# Programming Assignments: Schedule



## Assignment #1:

- Querying the GPU (OpenGL and CUDA) due Feb 10

## Assignment #2:

- Phong shading and procedural texturing (GLSL) due Mar 3

----- Spring Break: Mar. 28 – Apr. 5 -----

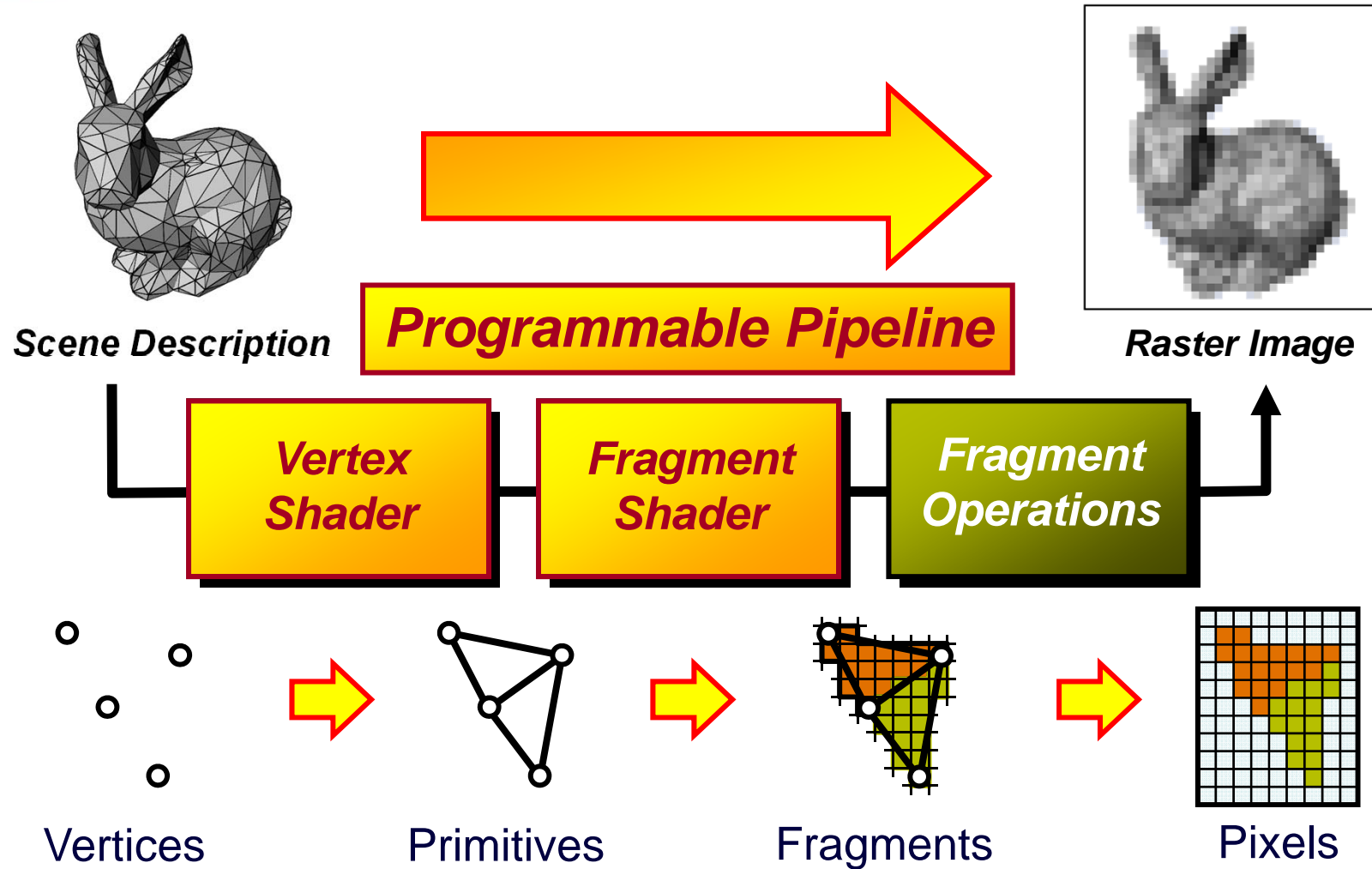
## Assignment #3:

- Image Processing with (a) GLSL, and (b) CUDA due Apr 7

## Assignment #4:

- Conjugate Gradient Linear Systems Solver (CUDA) due Apr 28

# Graphics Pipeline



# GPU Structure Before Unified Shaders

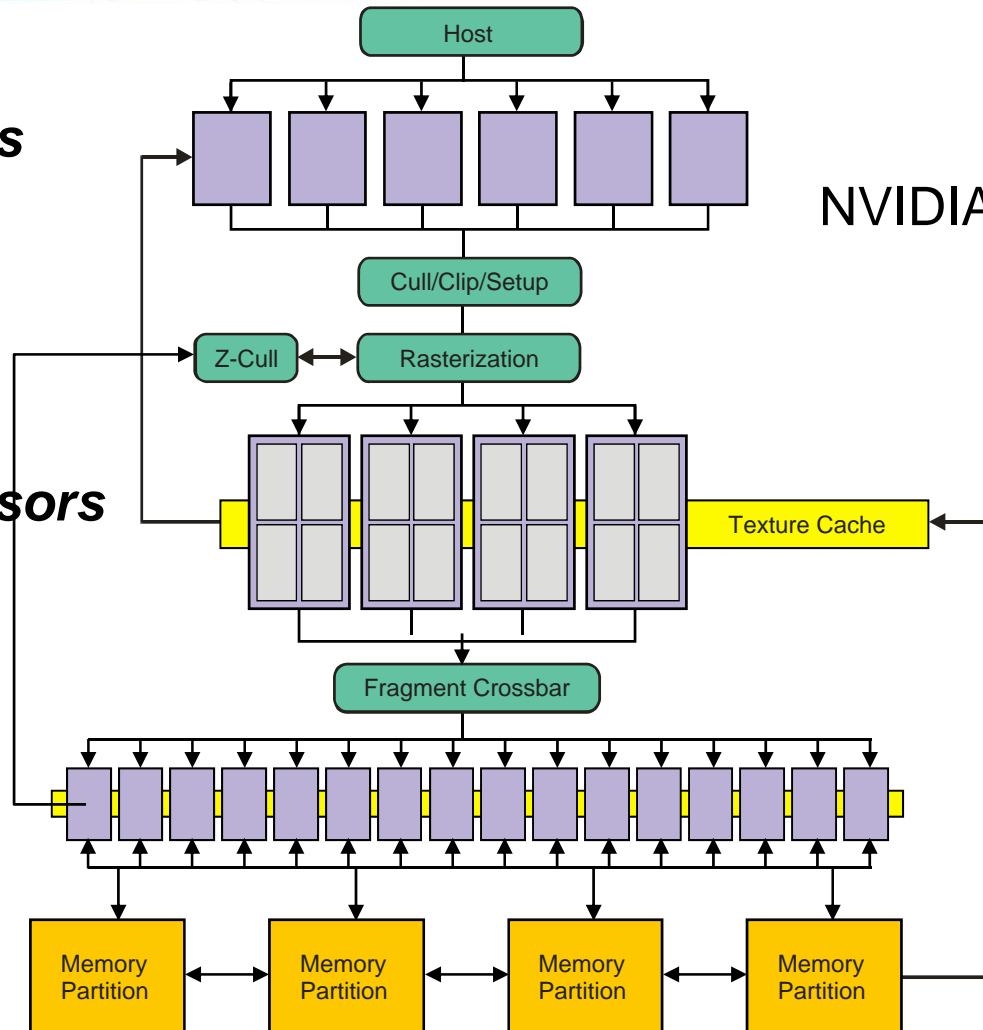


**Vertex Processors**

Example  
NVIDIA GeForce 6/7,  
2004, 2005

**Fragment Processors**

**Memory Access**  
Z-Compare and  
Blending



# 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

# Compile shader

1 unshaded fragment input record



```
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);
}
```



```
<diffuseShader>:
sample r0, v4, t0, s0
mul r3, v0, cb0[0]
madd r3, v1, cb0[1], r3
madd r3, v2, cb0[2], r3
clmp r3, r3, 1(0.0), 1(1.0)
mul o0, r0, r3
mul o1, r1, r3
mul o2, r2, r3
mov o3, 1(1.0)
```



1 shaded fragment output record



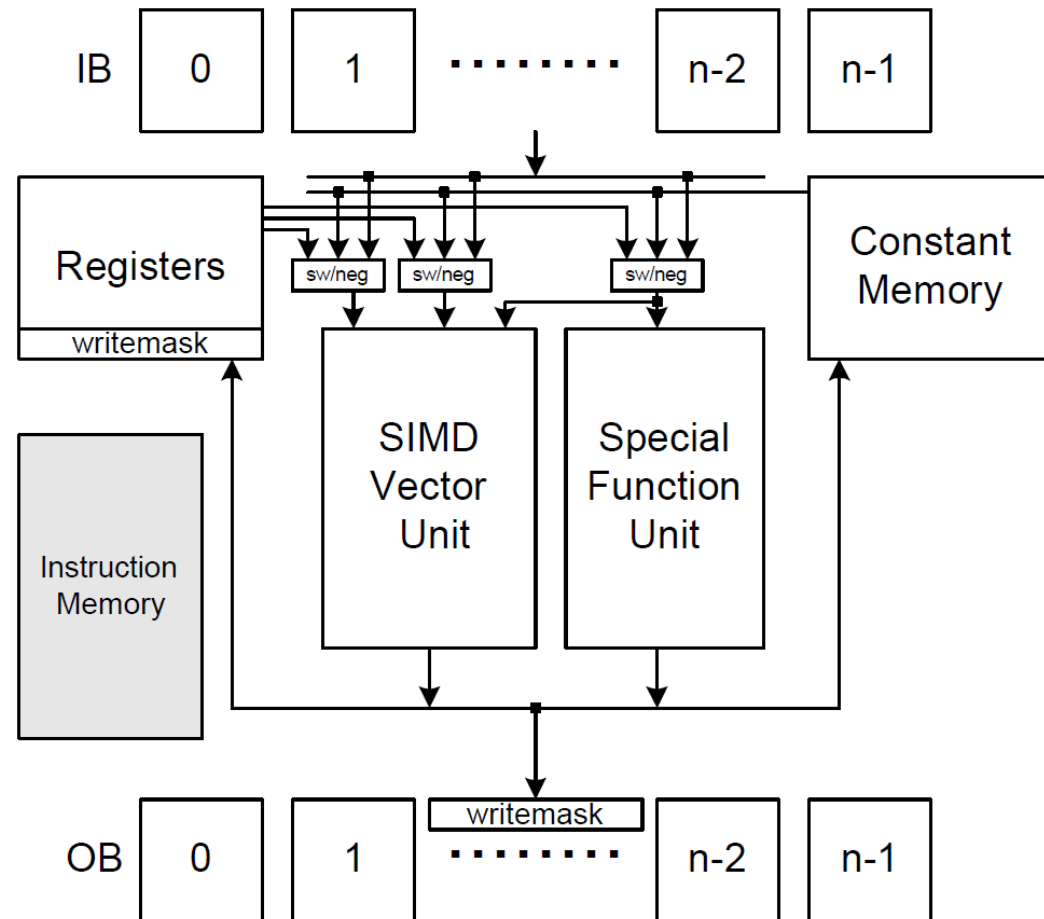
# Legacy Vertex Shading Unit (1)



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	<i>x,y,z,w</i>
1	Vertex weights	glVertexWeightEXT	<i>w,0,0,1</i>
2	Normal	glNormal	
3	Primary color	glColor	<i>r,g,b,a</i>
4	Secondary color	glSecondaryColorEXT	<i>r,g,b,1</i>
5	Fog coordinate	glFogCoordEXT	<i>f,0,0,1</i>
6	-	-	-
7	-	-	-
8	Texture coord 0	glMultiTexCoordARB (GL_TEXTURE0...)	<i>s,t,r,q</i>
9	Texture coord 1	glMultiTexCoordARB (GL_TEXTURE1...)	<i>s,t,r,q</i>
10	Texture coord 2	glMultiTexCoordARB (GL_TEXTURE2...)	<i>s,t,r,q</i>
11	Texture coord 3	glMultiTexCoordARB (GL_TEXTURE3...)	<i>s,t,r,q</i>
12	Texture coord 4	glMultiTexCoordARB (GL_TEXTURE4...)	<i>s,t,r,q</i>
13	Texture coord 5	glMultiTexCoordARB (GL_TEXTURE5...)	<i>s,t,r,q</i>
14	Texture coord 6	glMultiTexCoordARB (GL_TEXTURE6...)	<i>s,t,r,q</i>
15	Texture coord 7	glMultiTexCoordARB (GL_TEXTURE7...)	<i>s,t,r,q</i>

Code examples

```

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

swizzling!

# Legacy Vertex Shading Unit (3)



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

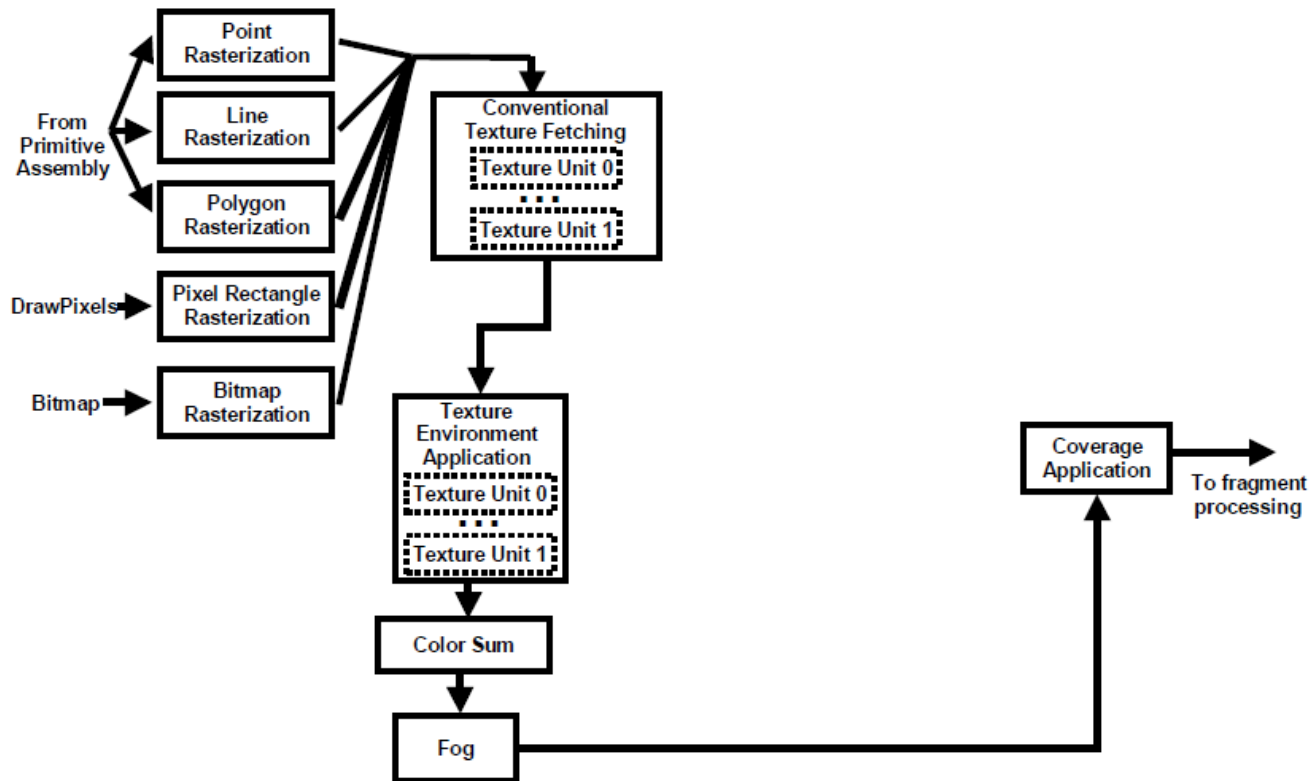
OpCode	Full Name	Description
MOV	Move	vector -> vector
MUL	Multiply	vector -> vector
ADD	Add	vector -> vector
MAD	Multiply and add	vector -> vector
DST	Distance	vector -> vector
MIN	Minimum	vector -> vector
MAX	Maximum	vector -> vector
SLT	Set on less than	vector -> vector
SGE	Set on greater or equal	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



NVIDIA Proprietary

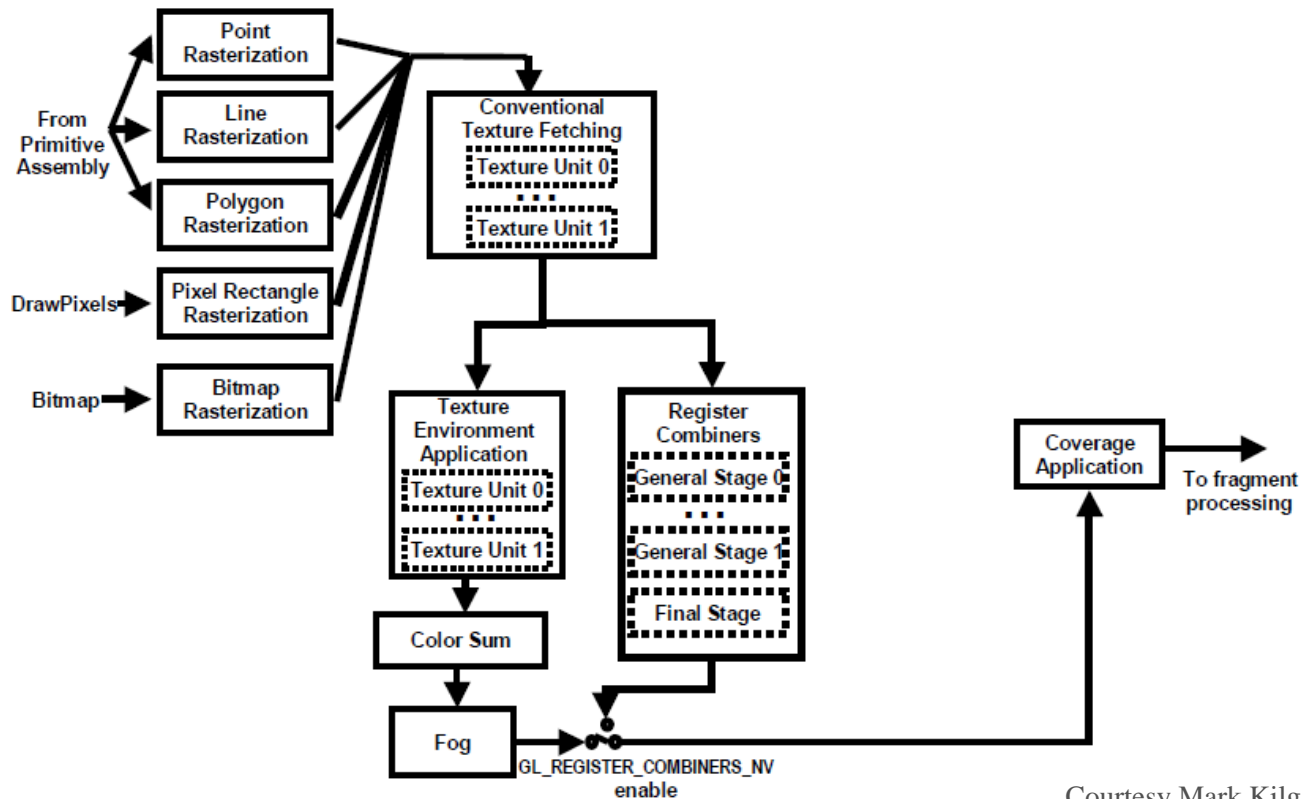
Courtesy Mark Kilgard

# Fast Forward to Programm. Fragment Shading



## NV10 OpenGL Fragment Texturing & Coloring

GeForce 256, 1999



NVIDIA Proprietary

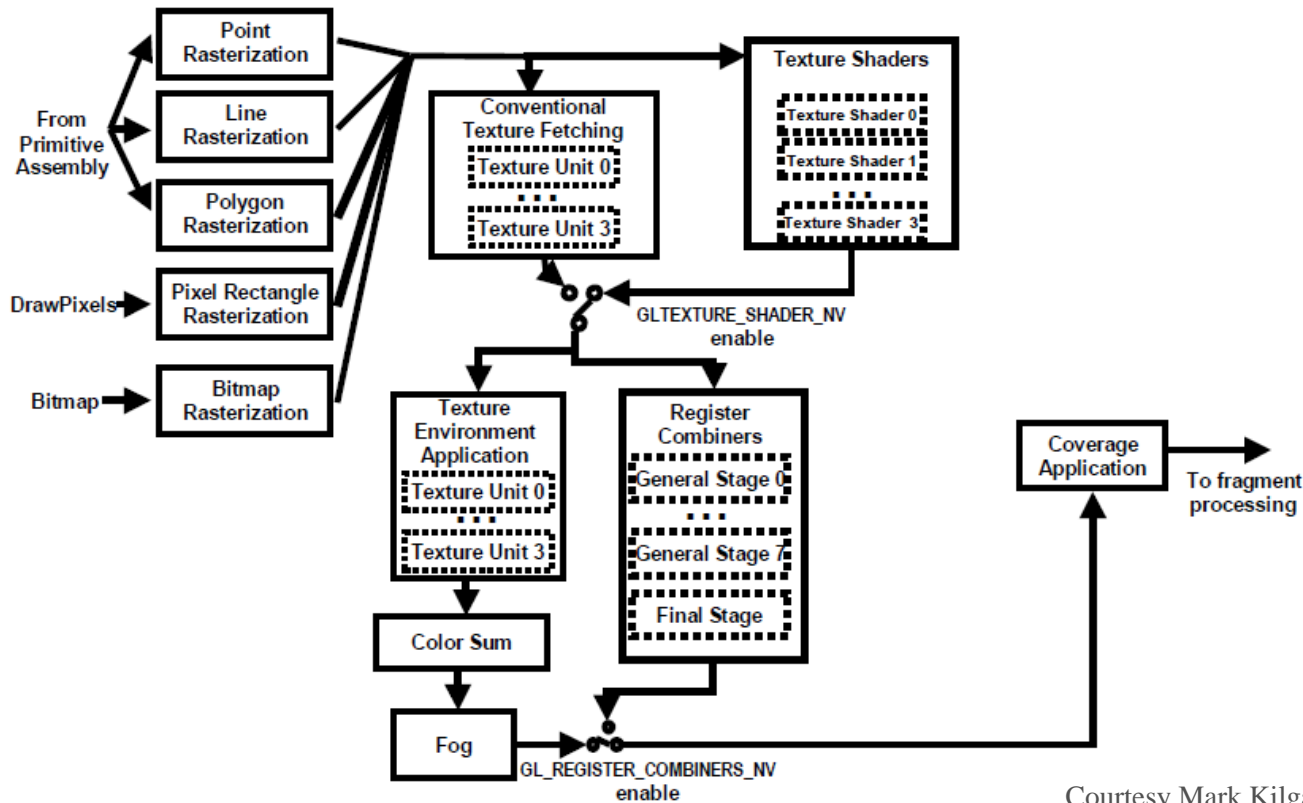
Courtesy Mark Kilgard

# Fast Forward to Programm. Fragment Shading



## NV20 OpenGL Fragment Texturing & Coloring

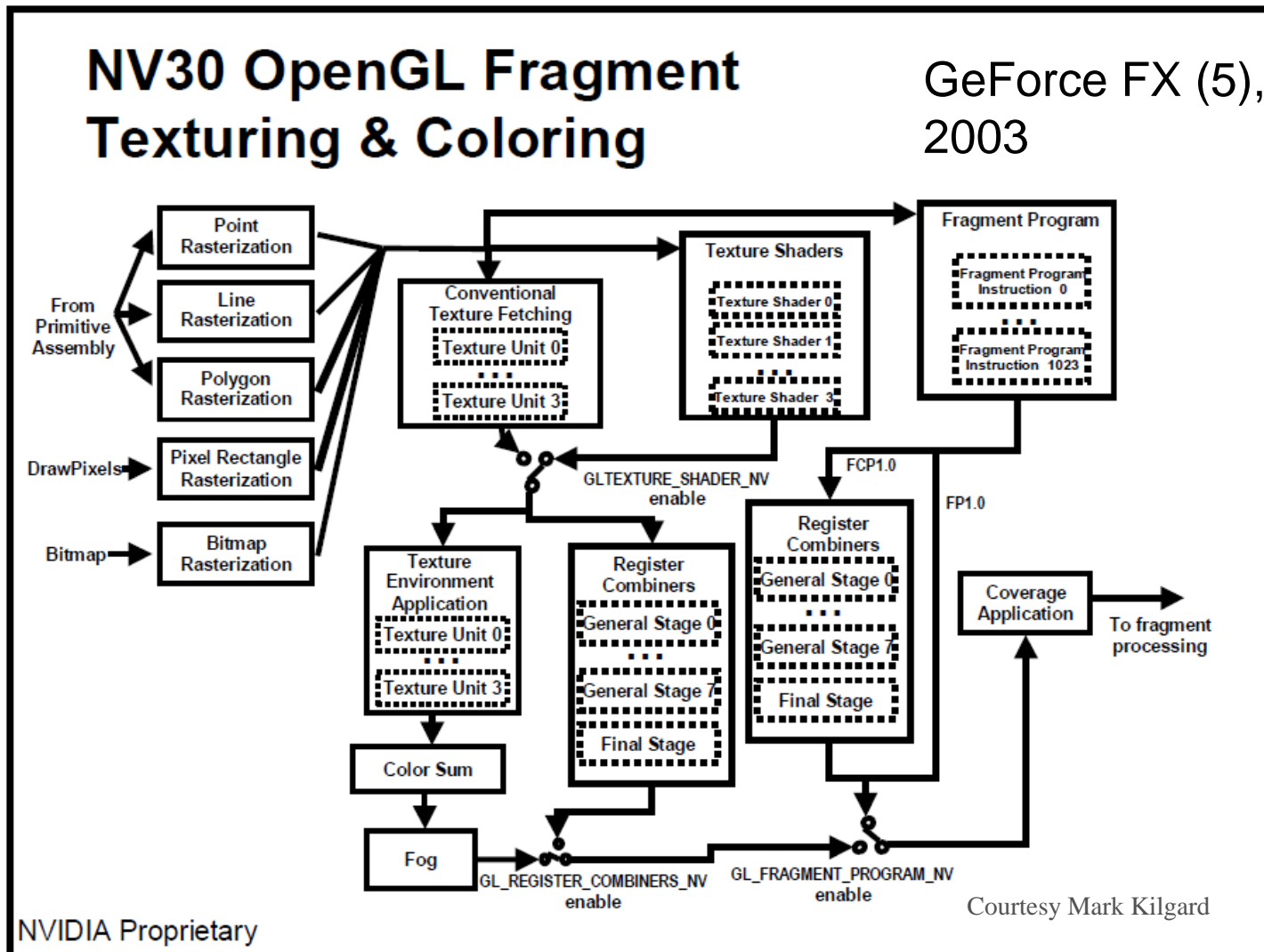
GeForce 3, 2001



NVIDIA Proprietary

Courtesy Mark Kilgard

# Fast Forward to Programm. Fragment Shading



# Legacy Fragment Shading Unit (1)



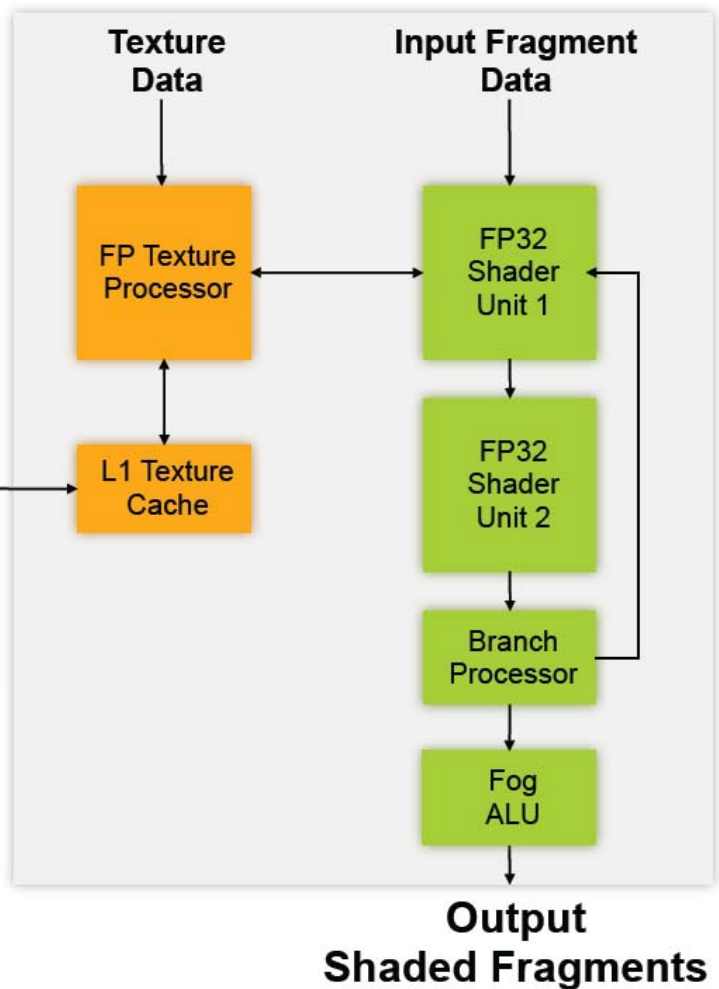
GeForce 6 (NV40), 2004

- dynamic branching

Texture Filter  
Bi / Tri / Aniso  
1 texture @ full speed  
4-tap filter @ full speed  
16:1 Aniso w/ Trilinear (128-tap)  
FP16 Texture Filtering

L2 Texture Cache

**SIMD Architecture**  
**Dual Issue / Co-Issue**  
**FP32 Computation**  
**Shader Model 3.0**



Shader Unit 1  
4 FP Ops / pixel  
Dual/Co-Issue  
Texture Address Calc  
Free fp16 normalize  
+ mini ALU

Shader Unit 2  
4 FP Ops / pixel  
Dual/Co-Issue  
+ mini ALU

# Legacy Fragment Shading Unit (2)



## Example code

```
!!ARBfp1.0

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

TEMP pos_wi n, temp;

TEX pos_wi n.z, uni t_tc, texture[ 1 ], 2D;

ADD pos_wi n.w, constants.y, -pos_wi n.z;
KIL pos_wi n.w;

MOV resul t.col or.w, pos_wi n.z;

MOV pos_wi n.xyw, uni t_tc;
MAD pos_wi n.xyz, pos_wi n, constants.a, -constants.b;

DP4 temp.w, mvp_inv[ 3 ], pos_wi n;
RCP temp.w, temp.w;

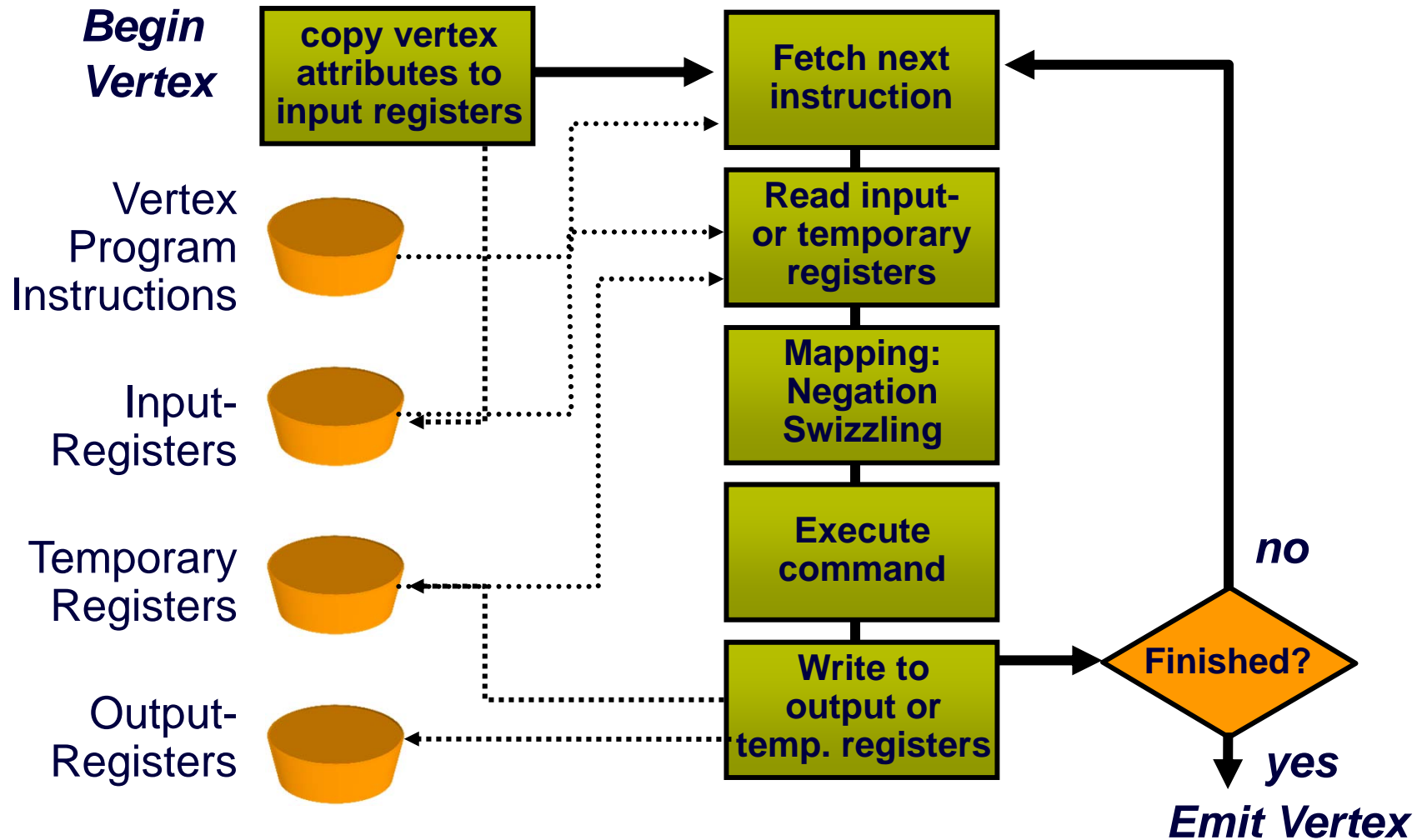
MUL pos_wi n, pos_wi n, temp.w;

DP4 resul t.col or.x, mvp_inv[ 0 ], pos_wi n;
DP4 resul t.col or.y, mvp_inv[ 1 ], pos_wi n;
DP4 resul t.col or.z, mvp_inv[ 2 ], pos_wi n;

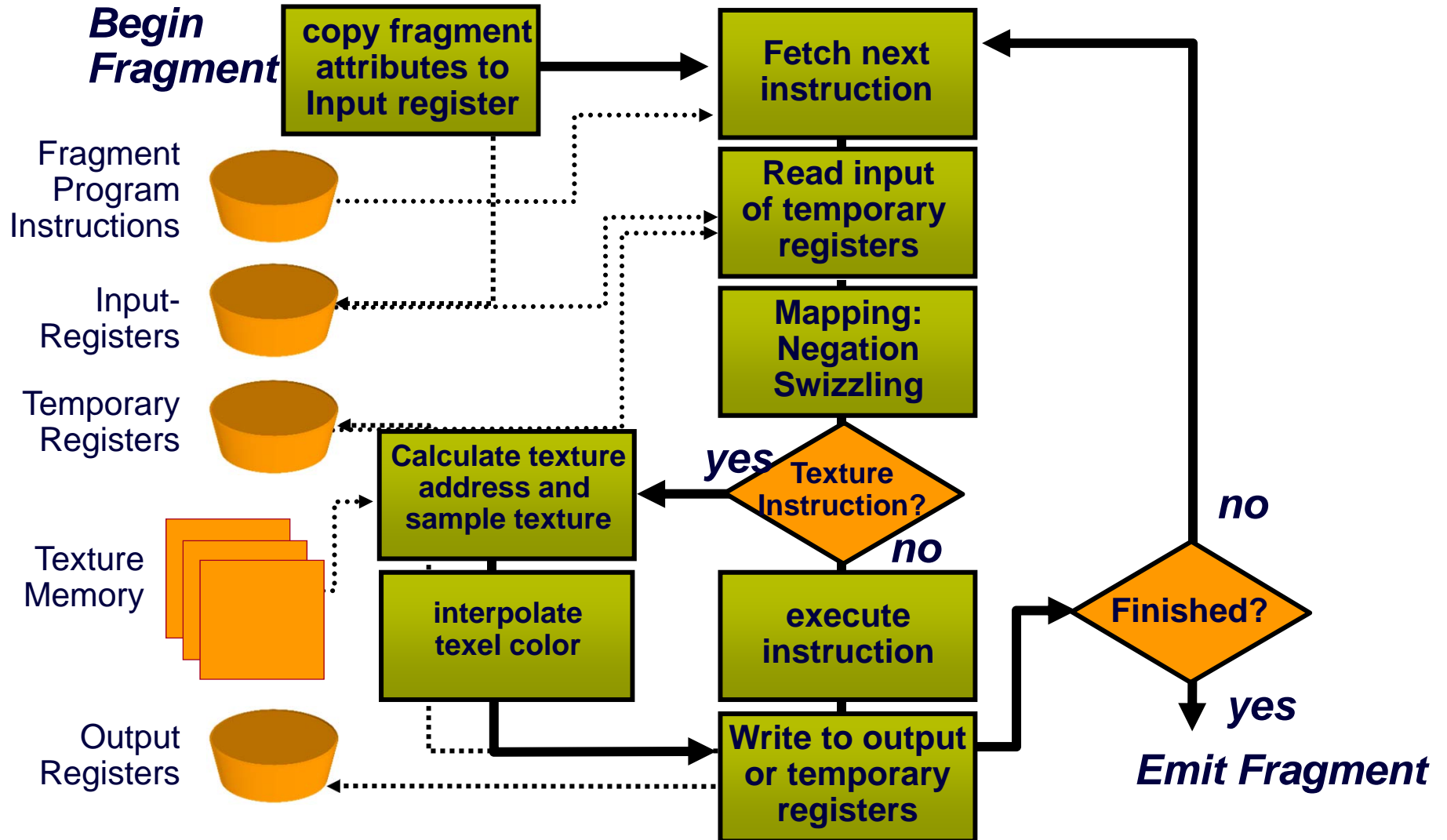
END
```



# Vertex Processor



# Fragment Processor



# 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;
}
```

# Per-Pixel(Fragment) Lighting

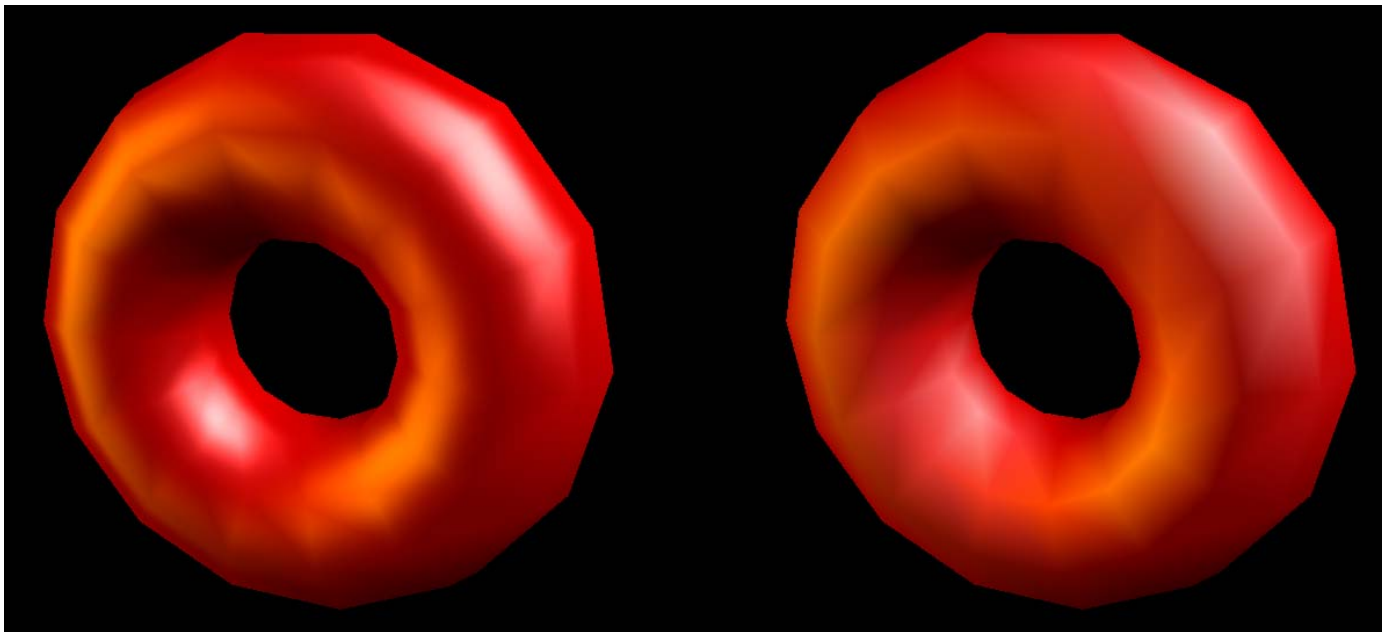


Simulating smooth surfaces by calculating illumination for each fragment

Example: specular highlights (Phong illumination/shading)

Phong shading:  
per-fragment evaluation

Gouraud shading:  
linear interpolation from vertices





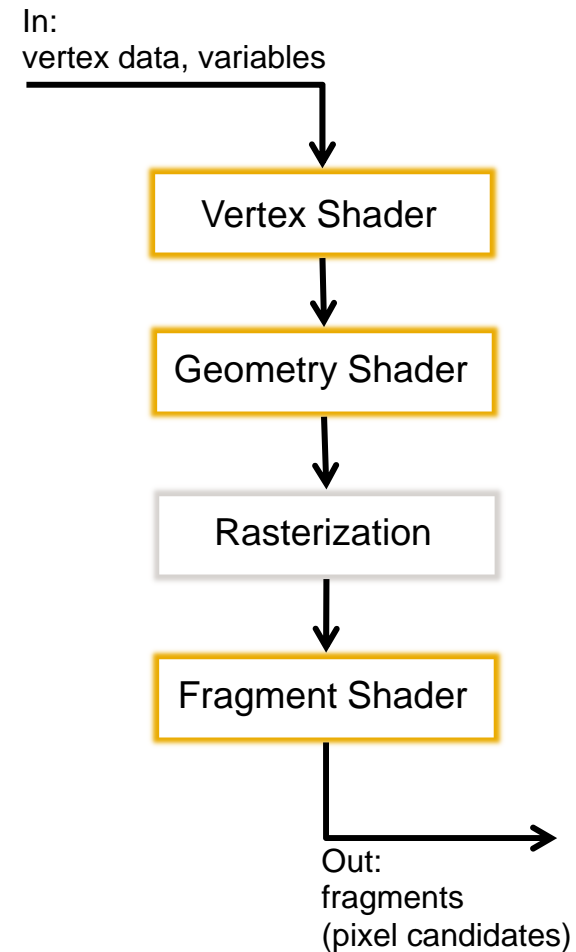
# Assignment 1 – Presentations

- Sign up for a presentation slot
  - Sign-up sheet on Peter's office door
  - Feb. 13 2pm-4pm

# Assignment 2 – Shaders – Su., 03.03.



- What you'll learn
  - Basic graphics pipeline
  - How to use vertex shaders
  - How to use fragment shaders
- What you'll pick up on the way
  - GLSL
  - Phong lighting and shading



# Assignment 2 – Shaders – Su., 03.03.



- What's already there
  - Different models
  - Shader setup
    - Loading
    - Compiling
    - Variables
  - Gouraud shading





# Assignment 2 – Shaders – Su., 03.03.



- What you'll have to do
  - Per fragment lighting (instead of per vertex)
  - Procedural texturing “OpenGL Shading Language” book (11.1-11.3)
  - Optional: (11.4) procedural bump mapping
- Where to start
  - readme.txt
  - Understand the framework code (how to edit shaders, etc.)
  - Look for TODOs

Thank you.