Programmable Graphics Hardware

Ian Buck

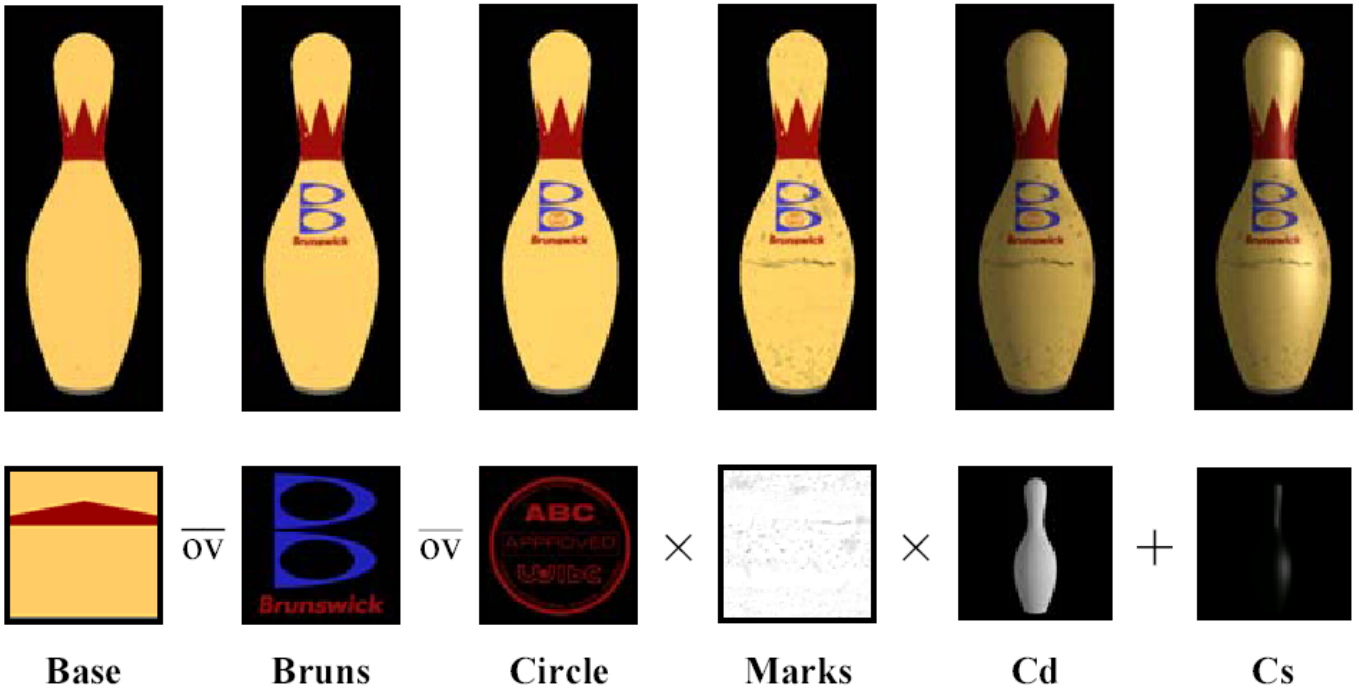
Computer Systems Laboratory
Stanford University

Outline

- Why programmable graphics hardware
- Vertex Programs
- Fragment Programs
- CG
- Trends

Why programmable graphics hardware?

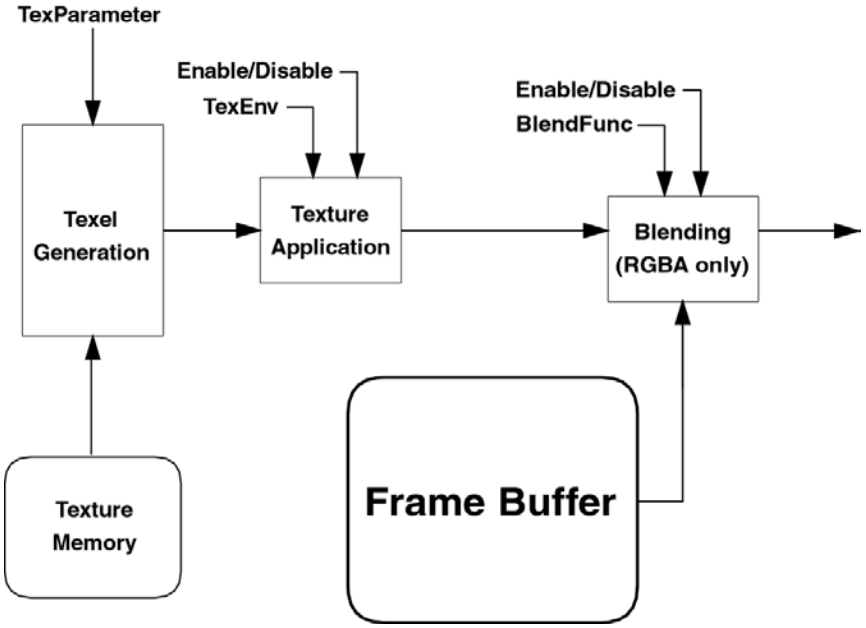
Multipass OGL



- Configuring OGL blending unit with multiple passes.

From Pat Hanrahan CS448
http://graphics.stanford.edu/courses/cs448a-01-fall/

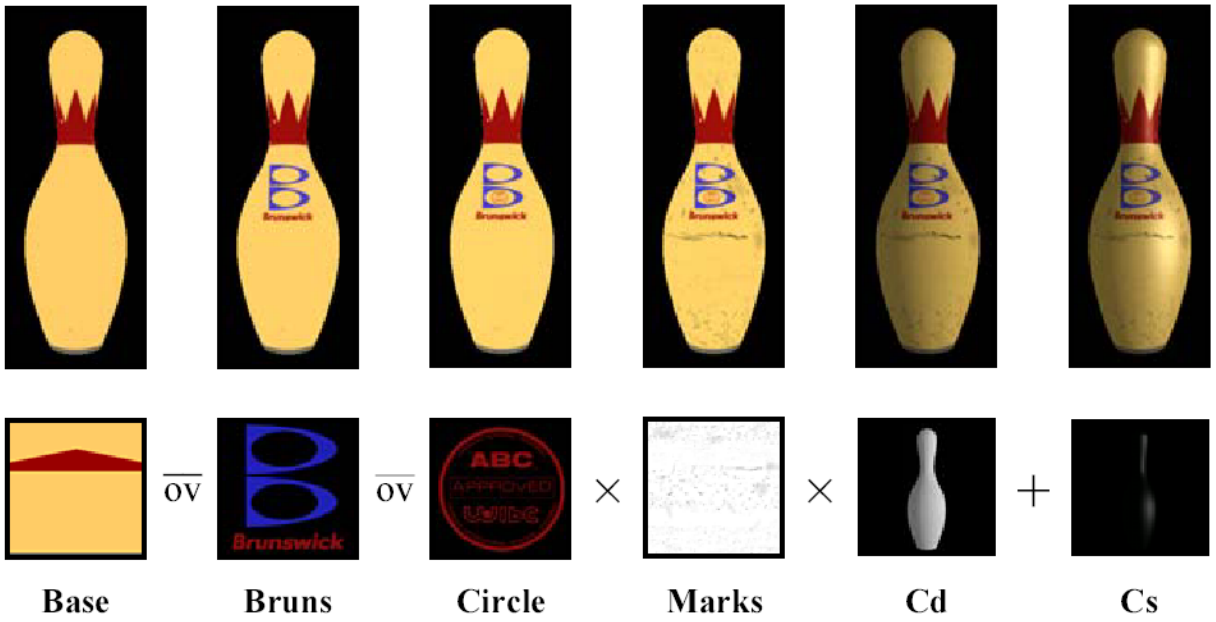
Programming OGL



- M. Peercy, M. Olano, J. Airey, J. Ungar, Interactive multipass programmable shading, SIGGRAPH 2000

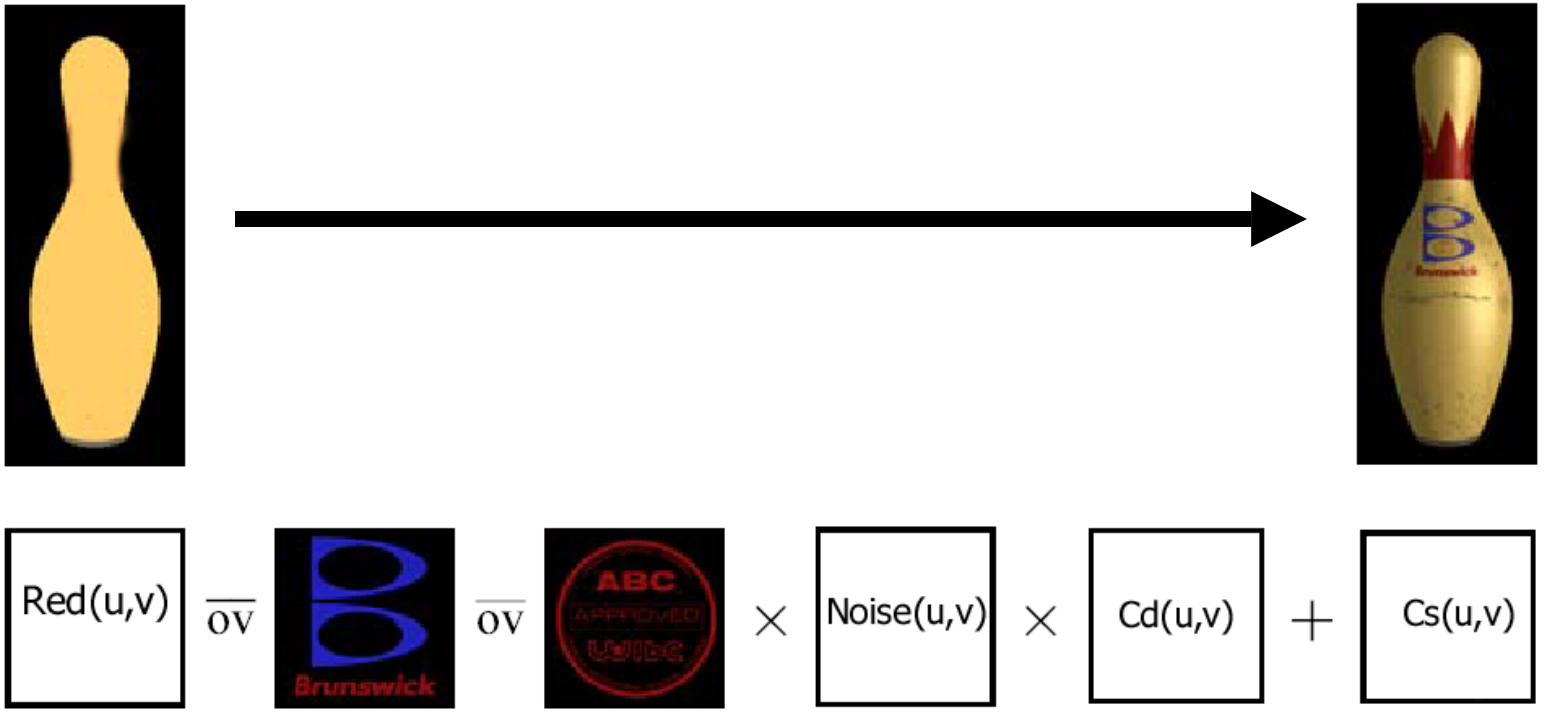
OpenGL Shading Downsides

- Limited instruction set
- Multipass consumes bandwidth
  - Load-Op-Store architecture
  - Excessive store and load of temporaries costly
- Bandwidth most costly resource in gfx hardware!



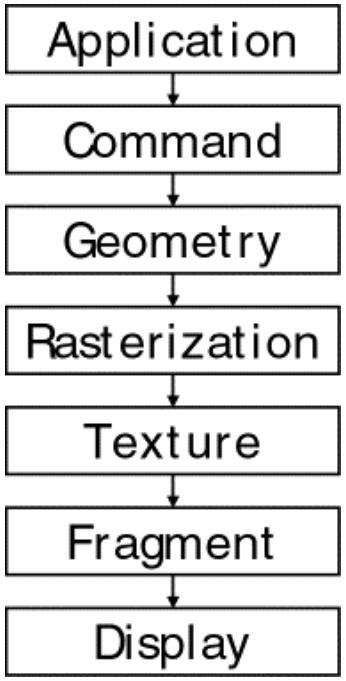
Programmable Graphics Hardware

- Goal: Render any effect in a single pass.
- Maximize computation. Minimize Bandwidth.

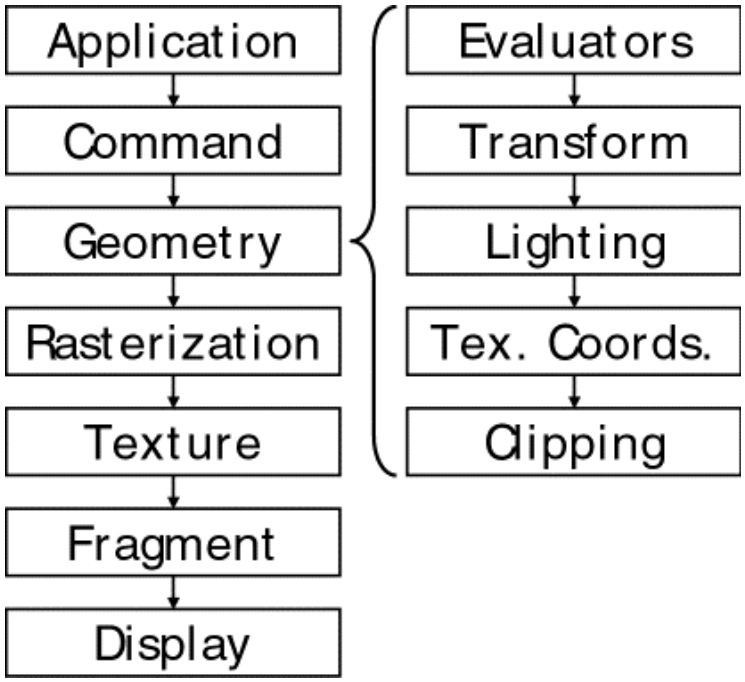


Vertex Programs

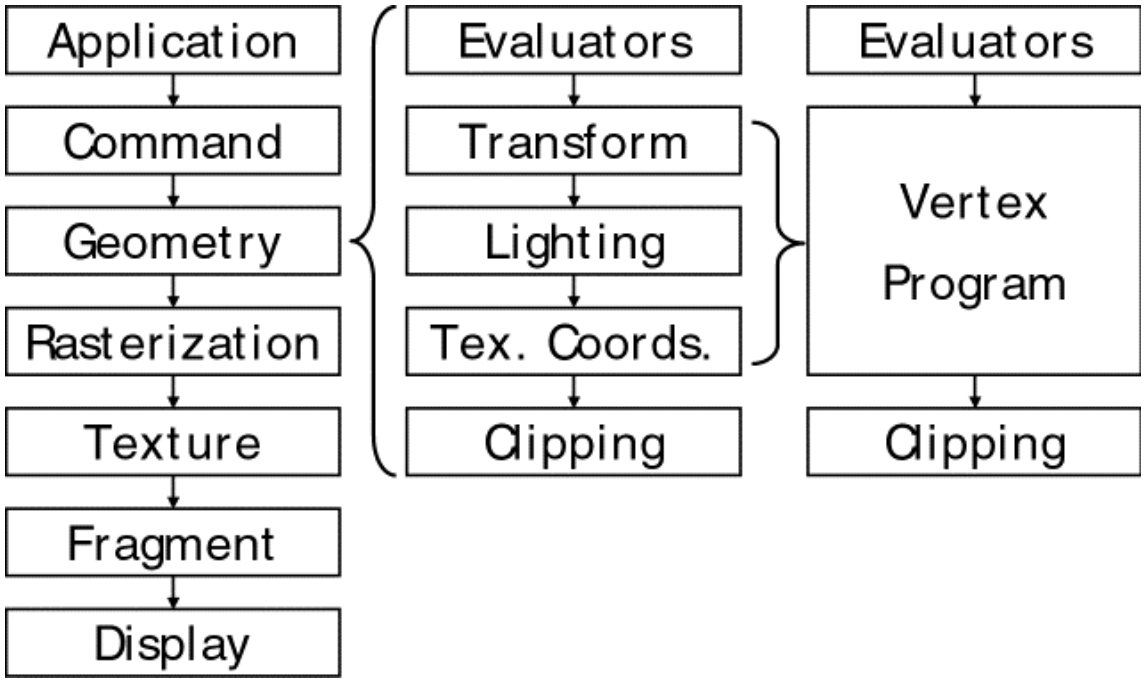
Programmable Vertex Pipeline



Programmable Vertex Pipeline



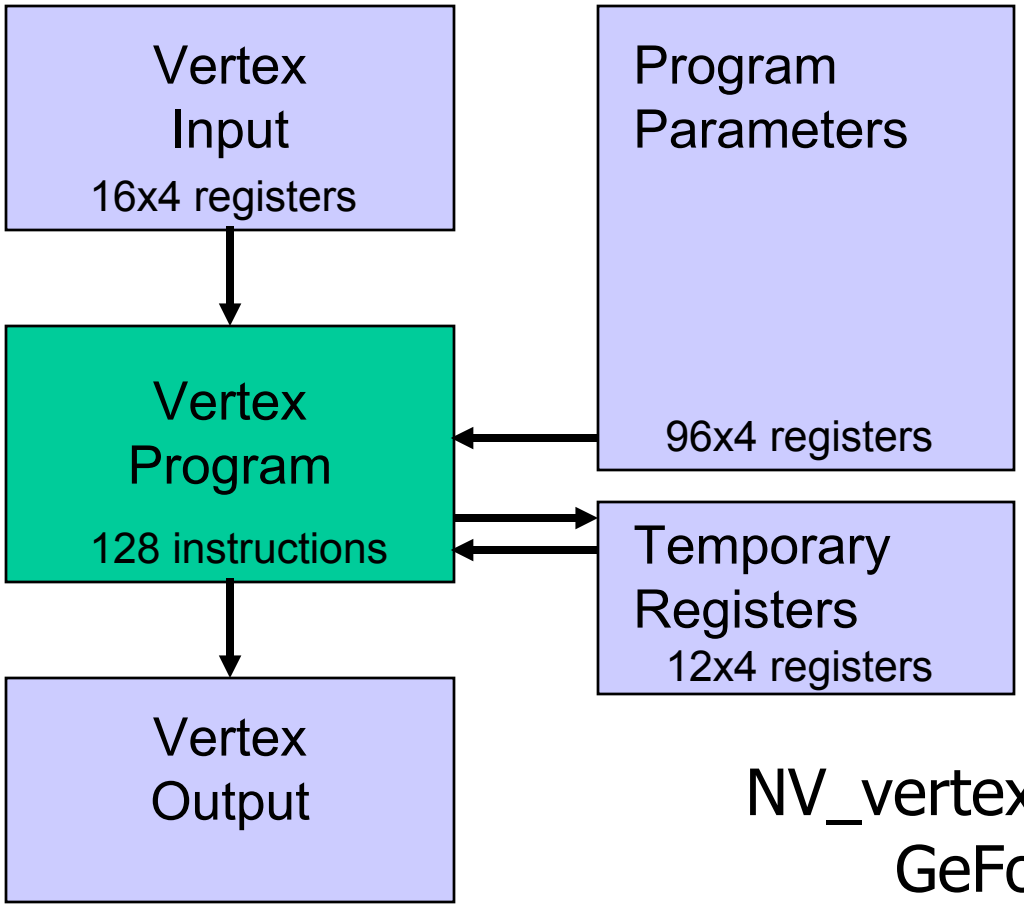
Programmable Vertex Pipeline



Vertex Programs

- Offers
  - Arbitrary fp32 per-vertex math
    - Position
    - Color
    - Texture Coord
- Missing
  - Dependencies
  - Clipping, tessellation
  - Assembly, evaluation
  - Rasterization

Vertex Programming



NV_vertex_program
GeForce3

From:
http://developer.nvidia.com/view.asp?IO=vertex_programs

Vertex Inputs

Attribute Register	Conventional per-vertex Parameter	Conventional Command	Conventional Mapping
0	vertex position	glVertex	x,y,z,w
1	vertex weights	glVertexWeightEXT	w,0,0,1
2	normal	glNormal	x,y,z,1
3	Primary color	glColor	r,g,b,a
4	secondary color	glSecondaryColorEXT	r,g,b,1
5	Fog coordinate	glFogCoordEXT	fc,0,0,1
6	-	glVertexAttrib	-
7	-	glVertexAttrib	-
8	Texture coord 0	glMultiTexCoord	s,t,r,q
9	Texture coord 1	glMultiTexCoord	s,t,r,q
10	Texture coord 2	glMultiTexCoord	s,t,r,q
11	Texture coord 3	glMultiTexCoord	s,t,r,q
12	Texture coord 4	glMultiTexCoord	s,t,r,q
13	Texture coord 5	glMultiTexCoord	s,t,r,q
14	Texture coord 6	glMultiTexCoord	s,t,r,q
15	Texture coord 7	glMultiTexCoord	s,t,r,q

Semantics defined by program NOT parameter name!

Vertex Input Registers

Attribute Register	Mnemonic Name	Typical Meaning
v[0]	v[OPOS]	object position
v[1]	v[WGHT]	vertex weight
v[2]	v[NRML]	normal
v[3]	v[COL0]	primary color
v[4]	v[COL1]	secondary color
v[5]	v[FOGC]	fog coordinate
v[6]	-	-
v[7]	-	-
v[8]	v[TEX0]	texture coordinate 0
v[9]	v[TEX1]	texture coordinate 1
v[10]	v[TEX2]	texture coordinate 2
v[11]	v[TEX3]	texture coordinate 3
v[12]	v[TEX4]	texture coordinate 4
v[13]	v[TEX5]	texture coordinate 5
v[14]	v[TEX6]	texture coordinate 6
v[15]	v[TEX7]	texture coordinate 7

Semantics defined by program NOT parameter name!

Vertex Output Registers

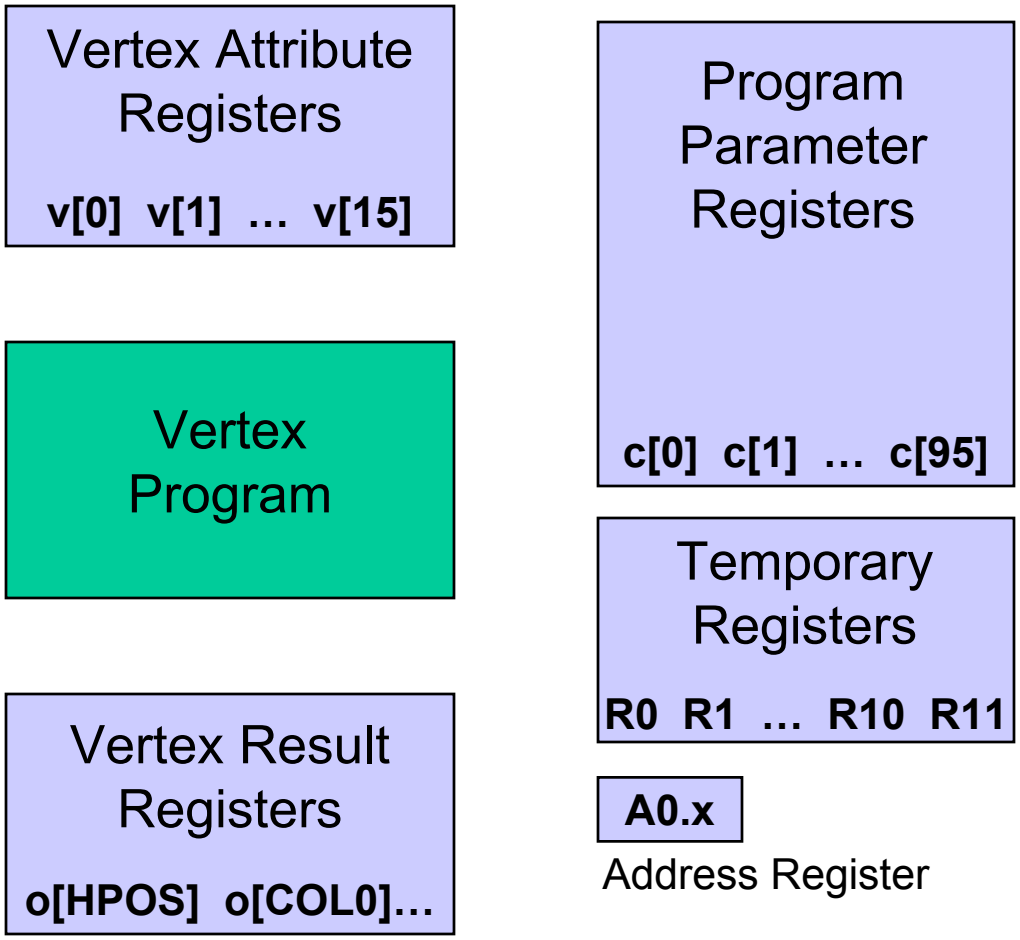
Register Name	Description	Component Interpretation
o[HPOS]	Homogeneous clip space position	(x,y,z,w)
o[COL0]	Primary color (front-facing)	(r,g,b,a)
o[COL1]	Secondary color (front-facing)	(r,g,b,a)
o[BFC0]	Back-facing primary color	(r,g,b,a)
o[BFC1]	Back-facing secondary color	(r,g,b,a)
o[FOGC]	Fog coordinate	(f,*,*,*)
o[PSIZ]	Point size	(p,*,*,*)
o[TEX0]	Texture coordinate set 0	(s,t,r,q)
o[TEX1]	Texture coordinate set 1	(s,t,r,q)
o[TEX2]	Texture coordinate set 2	(s,t,r,q)
o[TEX3]	Texture coordinate set 3	(s,t,r,q)
o[TEX4]	Texture coordinate set 4	(s,t,r,q)
o[TEX5]	Texture coordinate set 5	(s,t,r,q)
o[TEX6]	Texture coordinate set 6	(s,t,r,q)
o[TEX7]	Texture coordinate set 7	(s,t,r,q)

Semantics defined by down-stream pipeline stages.

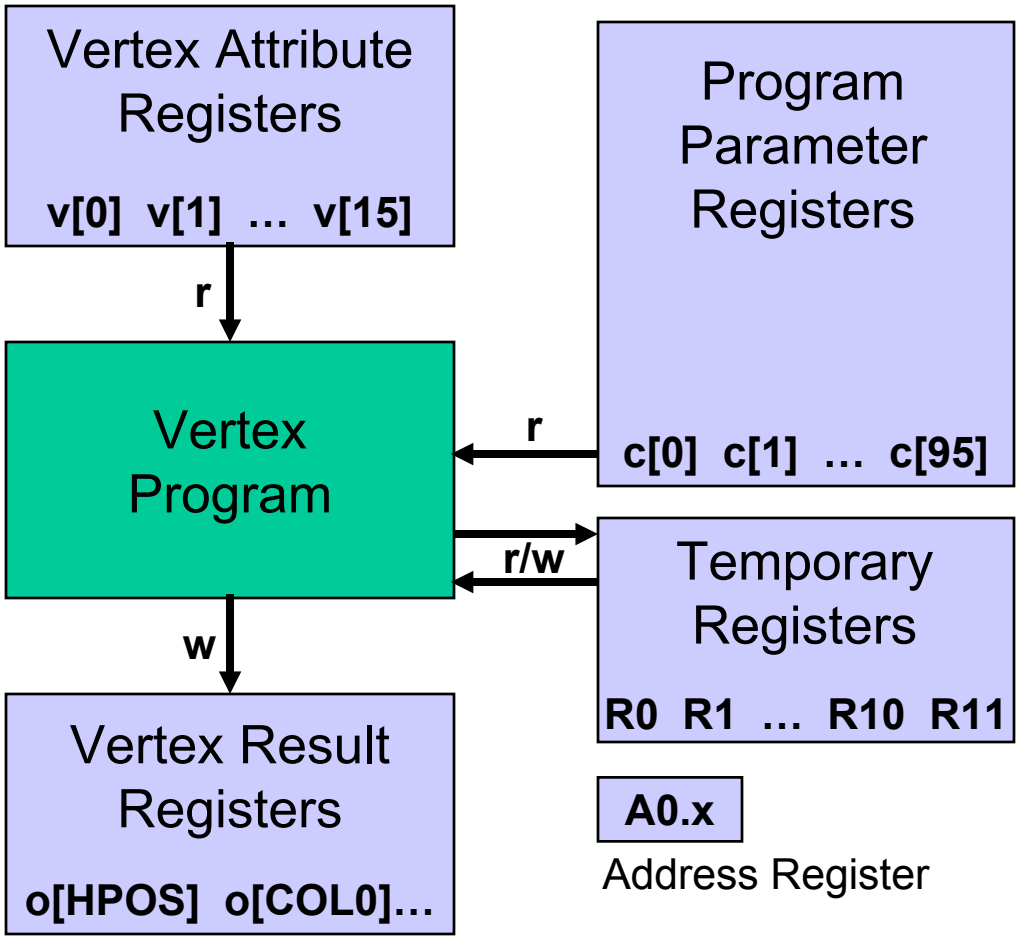
Vertex Constant Registers

- **Up to 96x4 per-block parameters**
- **Specified outside of glBegin/glEnd**
- **Store parameters such as matrices, lighting params, and constants required by vertex programs.**
- **Values specified with new commands**
  - glProgramParameter4fNV(GL_VERTEX_PROGRAM_NV, index, x, y, z, w)
  - glTrackMatrixNV(GL_VERTEX_PROGRAM_NV, 4, GL_MODELVIEW, GL_IDENTITY_NV);
- **Correspond to 96 registers (c[0] , … , c[95])**

The Register Set



Vertex Program Register Access



Assembly Language

17 Basic Instructions

- **ARL**
- **MOV**
- **MUL**
- **ADD**
- **MAD**
- **RCP**
- **RSQ**
- **DP3**
- **DP4**
- **DST**
- **MIN**
- **MAX**
- **SLT**
- **SGE**
- **EXP**
- **LOG**
- **LIT**

Instructions

- **ARL: Address register load**
- **MOV: Move**
- **MUL: Multiply**
- **ADD: Add**
- **MAD: Multiply and Add**
- **RCP: Scalar Reciprocal**
- **RSQ: Scalar Reciprocal Square Root**
- **DP3: 3 Component Dot Product**
- **DP4: 4 Component Dot Product**
- **DST: Distance Attenuation Vector**
- **MIN: Minimum**
- **MAX: Maximum**
- **SLT: Set Less Than**
- **SGE: Set Greater Than**
- **EXP: Scalar Exponential base 2**
- **LOG: Scalar Logarithm base 2**
- **LIT: Lighting Vector**

The Instruction Set

ADD Example:

```
ADD     R1, R2, -R3;
```

before

	R1	R2	R3
x	0.0	7.0	2.0
y	0.0	3.0	2.1
z	0.0	6.0	5.0
w	0.0	2.0	7.0

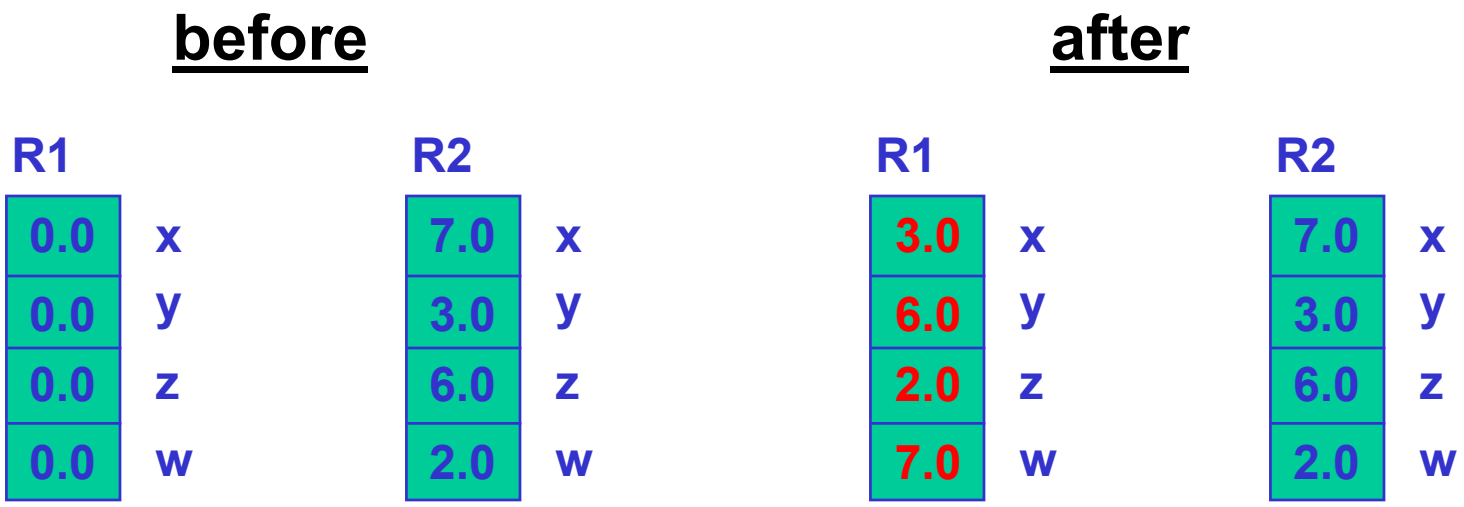
after

	R1	R2	R3
x	5.0	7.0	2.0
y	0.9	3.0	2.1
z	1.0	6.0	5.0
w	-5.0	2.0	7.0

Vertex Programming

Source registers can be “swizzled":

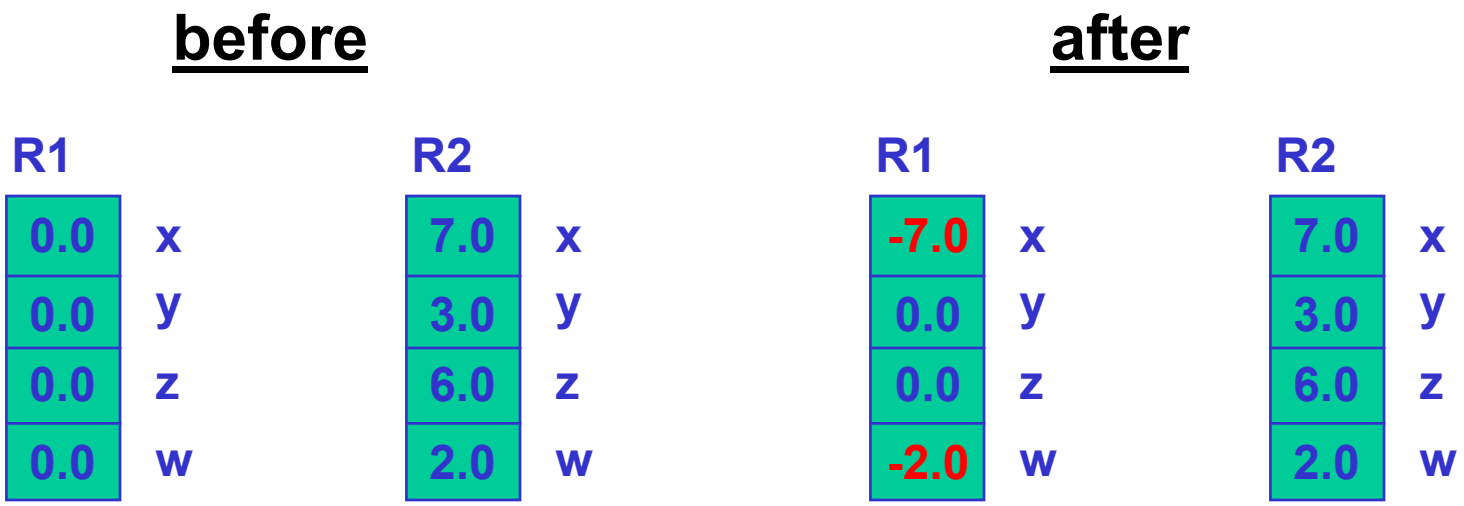
```
MOV     R1, R2.yzwx;
```



Vertex Programming

Destination register masking:

```
MOV     R1.xw, -R2;
```



Example Programs

3-Component Normalize

```
#
# R1 = (nx,ny,nz)
#
# R0.xyz = normalize(R1)
# R0.w   = 1/sqrt(nx*nx + ny*ny + nz*nz)
#
DP3 R0.w, R1, R1;
RSQ R0.w, R0.w;
MUL R0.xyz, R1, R0.w;
```

Example Programs

3-Component Cross Product

```
#
# Cross product |  i      j      k    | into R2.
#               | R0.x  R0.y  R0.z |
#               | R1.x  R1.y  R1.z |
#
MUL R2, R0.zxyw, R1.yzxw;
MAD R2, R0.yzxw, R1.zxyw, -R2;
```

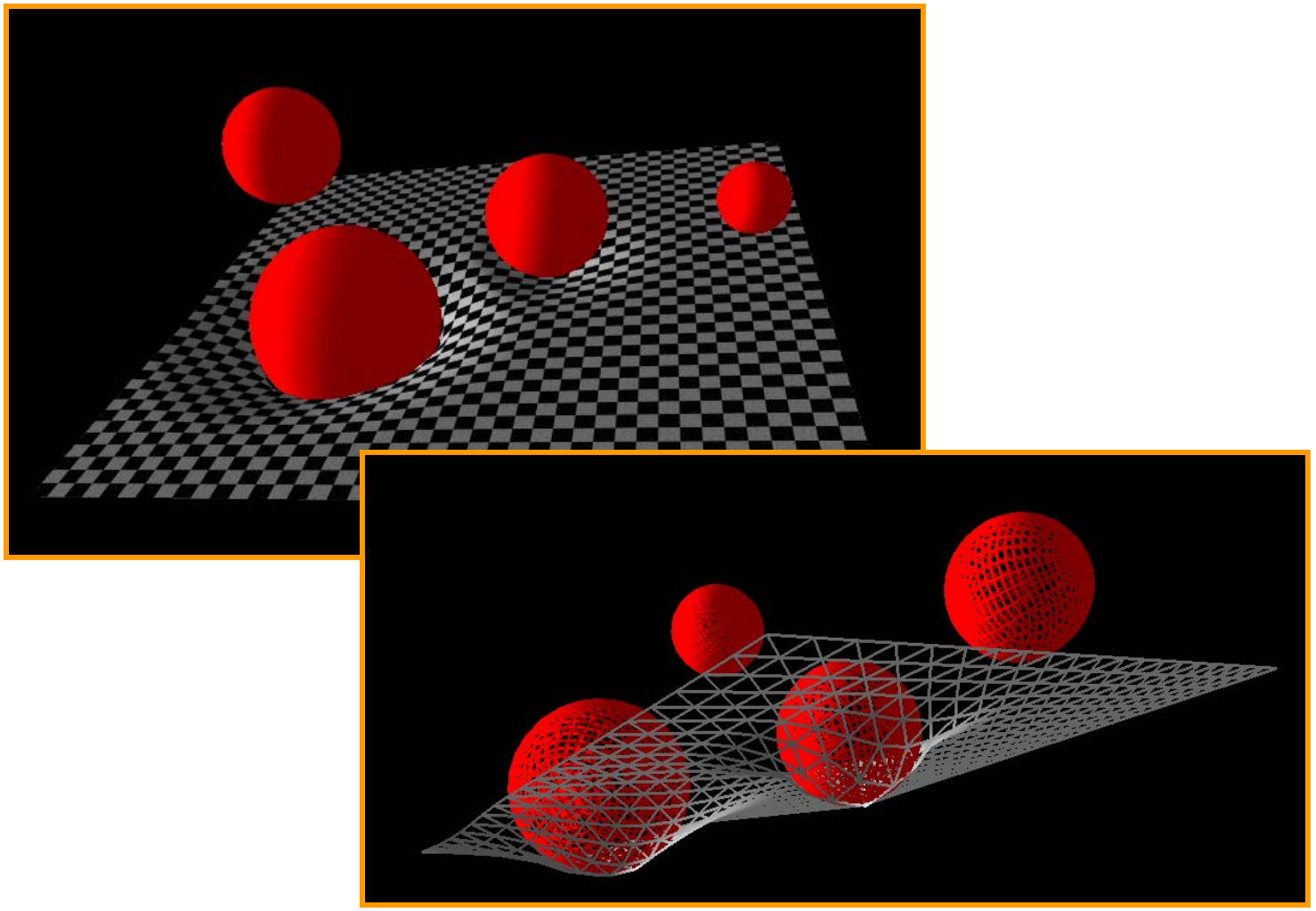
Example Programs

Simple Specular and Diffuse Lighting

```
!!VP1.0
#
# c[0-3]  = modelview projection (composite) matrix
# c[4-7]  = modelview inverse transpose
# c[32]   = eye-space light direction
# c[33]   = constant eye-space half-angle vector (infinite viewer)
# c[35].x = pre-multiplied monochromatic diffuse light color & diffuse mat.
# c[35].y = pre-multiplied monochromatic ambient light color & diffuse mat.
# c[36]   = specular color
# c[38].x = specular power
# outputs homogenous position and color
#
DP4   o[HPOS].x, c[0], v[OPOS];      # Compute position.
DP4   o[HPOS].y, c[1], v[OPOS];
DP4   o[HPOS].z, c[2], v[OPOS];
DP4   o[HPOS].w, c[3], v[OPOS];
DP3   R0.x, c[4], v[NRML];           # Compute normal.
DP3   R0.y, c[5], v[NRML];
DP3   R0.z, c[6], v[NRML];           # R0 = N' = transformed normal
DP3   R1.x, c[32], R0;               # R1.x = Ldir DOT N'
DP3   R1.y, c[33], R0;               # R1.y = H DOT N'
MOV   R1.w, c[38].x;                 # R1.w = specular power
LIT   R2, R1;                        # Compute lighting values
MAD   R3, c[35].x, R2.y, c[35].y;    # diffuse + ambient
MAD   o[COL0].xyz, c[36], R2.z, R3;  # + specular
END
```

Example Programs

- **Dynamic displacements of surfaces by objects**



Example Programs

- Custom transform, lighting, and skinning

Vertex Programs

- Hardware
  - GeForce3
  - Radeon 8500
- API
  - DirectX 8.1 “Vertex Shader”
  - OGL: NV_vertex_program

Details

- No Jumps or Branches
- Performance
  - Smaller the program, faster the execution
  - CPU vs GPU tradeoff
    - Is 2.8GHz P4 faster than GeForce4 TI?
  - Cheap to switch programs

Next Generation

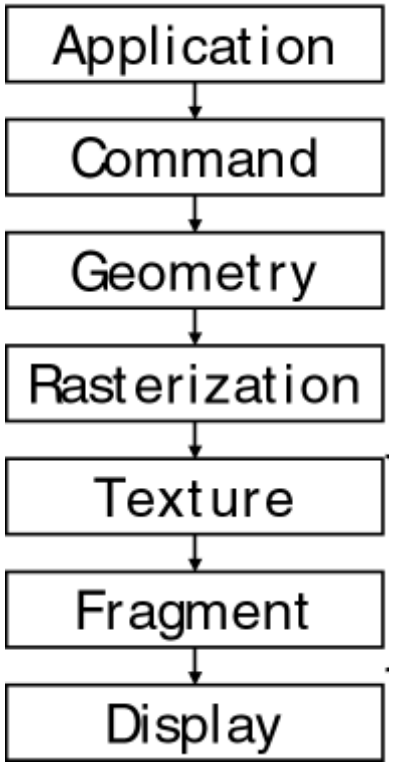
- Available in NV30 “CineFX” & Radeon 9700
- Branching supported
  - BRA
- Subroutines
  - CAL, RET
- 256 instructions
  - 64K ops per vertex
- New Math Instructions
  - COS, SIN, EX2, LG2
- More Info:
  - http://developer.nvidia.com/docs/IO/3121/ATT/CineFX_1-final.pdf
  - NV_vertex_program2

Other stuff

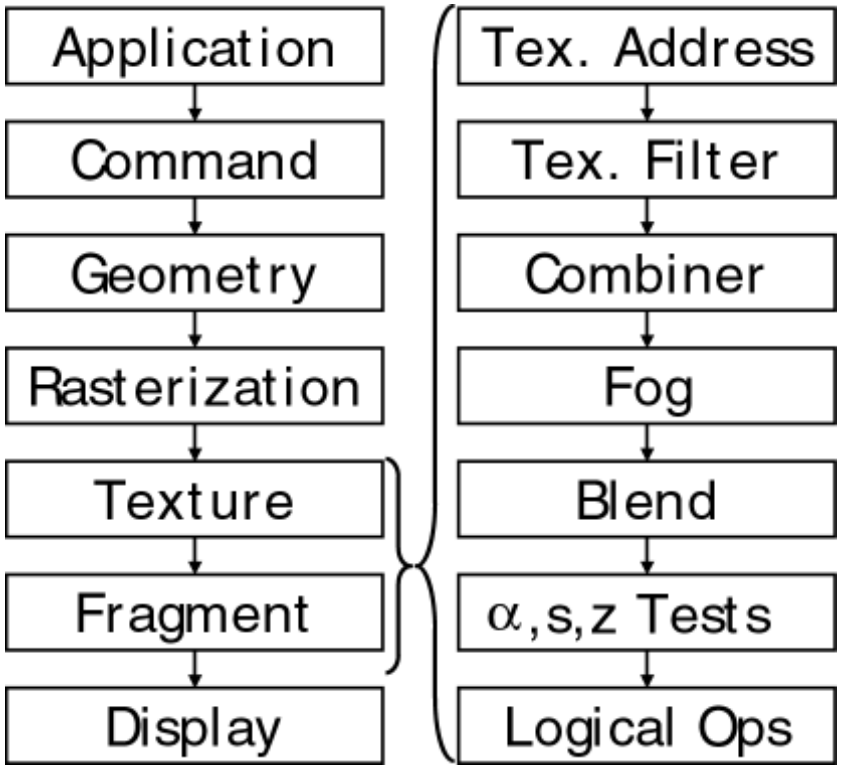
- DirectX 8.1 Vertex Shaders
  - http://developer.nvidia.com/view.asp?IO=vstovp
- Vertex State Programs
- C interface
- Specification
  - http://developer.nvidia.com/view.asp?PAGE=opengl_specs

Pixel Shaders

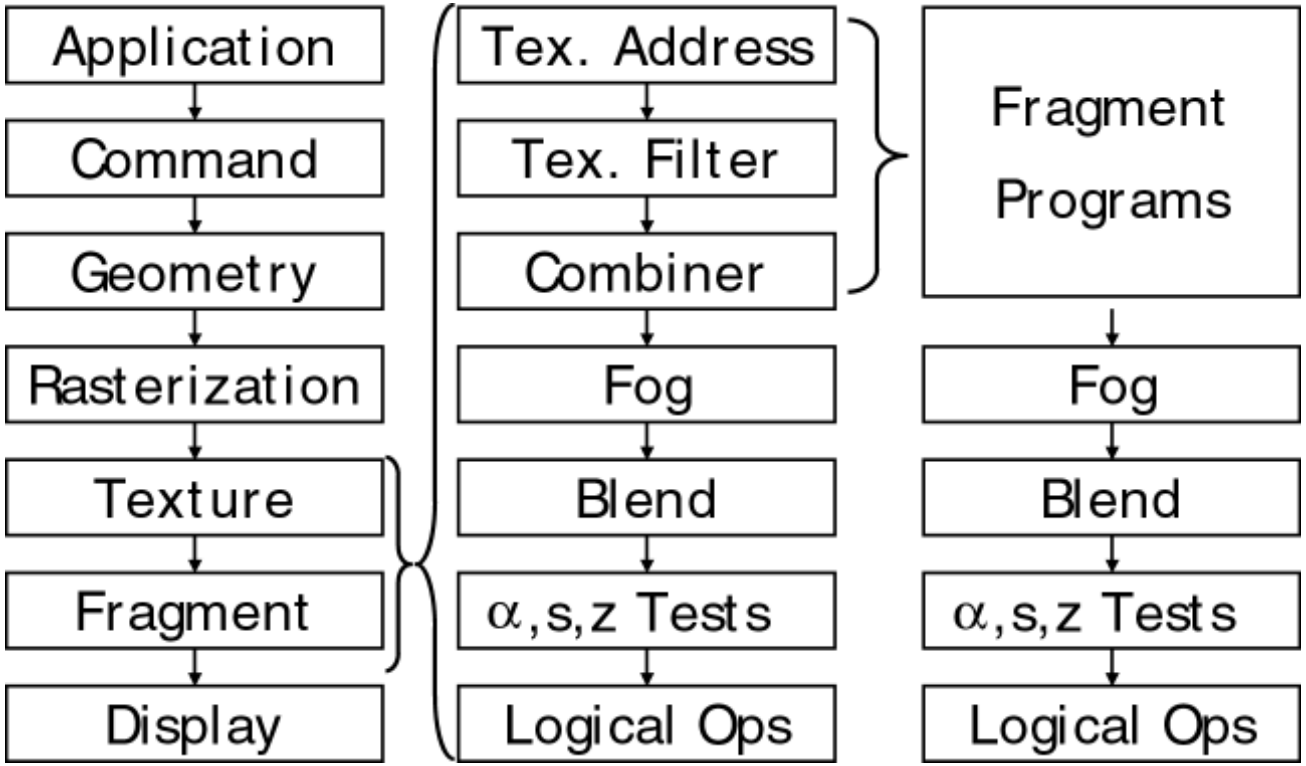
Fragment Pipeline



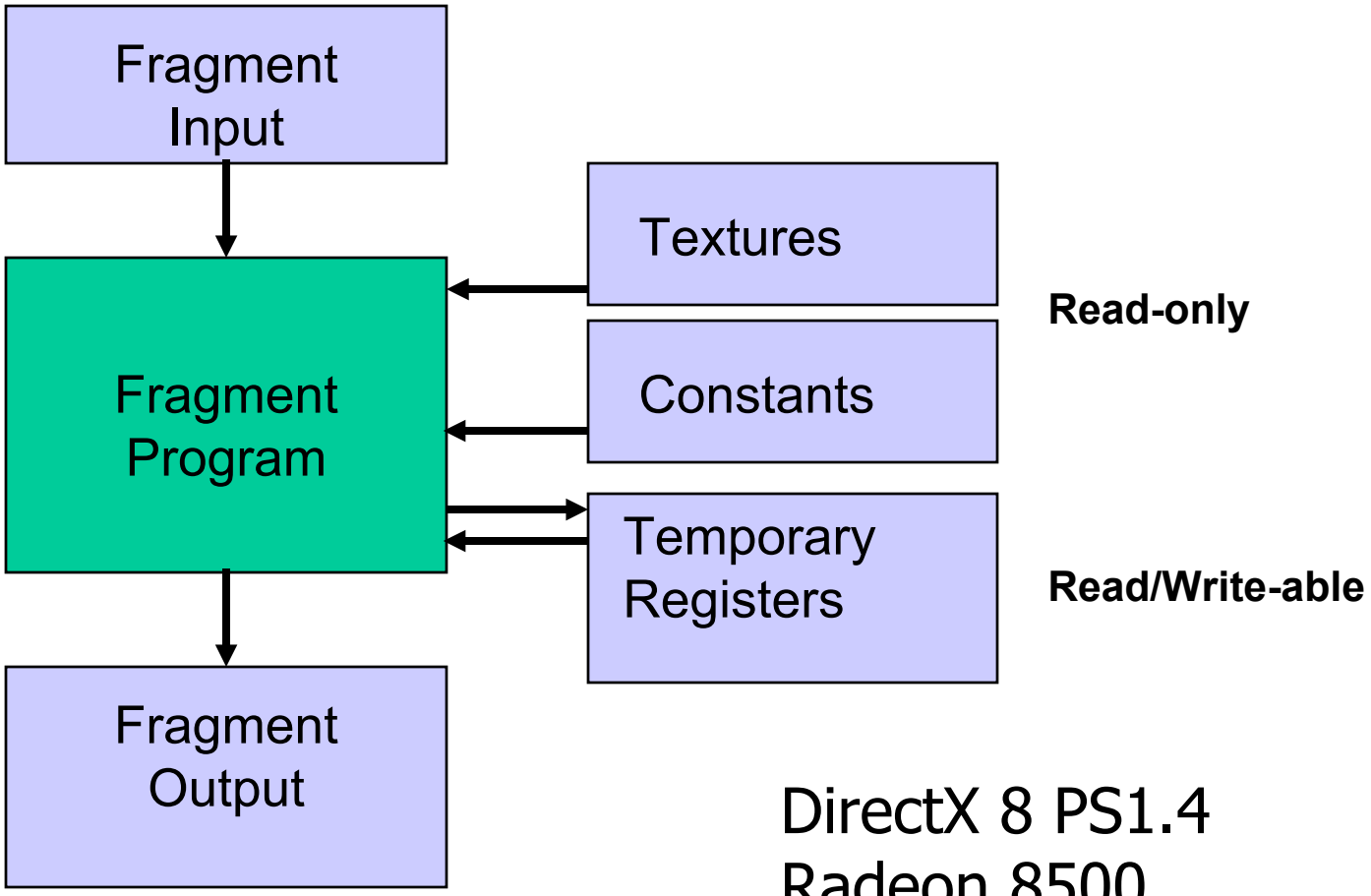
Fragment Pipeline



Fragment Pipeline



DX8 Fragment Programming



DirectX 8 PS1.4
Radeon 8500

PS1.4 Input Registers

Attribute Register	Components	Typical Meaning
v0	r,g,b,a	Diffuse Color
v1	r,g,b,a	Specular Color
t0	s,t,r,q	TexCoord 0
t1	s,t,r,q	TexCoord 1
t2	s,t,r,q	TexCoord 2
t3	s,t,r,q	TexCoord 3
t4	s,t,r,q	TexCoord 4
t5	s,t,r,q	TexCoord 5

Semantics defined by program NOT parameter name!

PS1.4 Registers

- Constants c0,…, c7
  - [-1..1]
  - SetPixelShaderConstant()
  - def c3, -1.0, 0.0, 0.0, 1.0
- Registers r0,…,r5
  - Extended range

PS1.4 Output Registers

Attribute Register	Range	Output
r0	0..1	Color
r5.r	0..1	Depth

PS1.4 Instructions

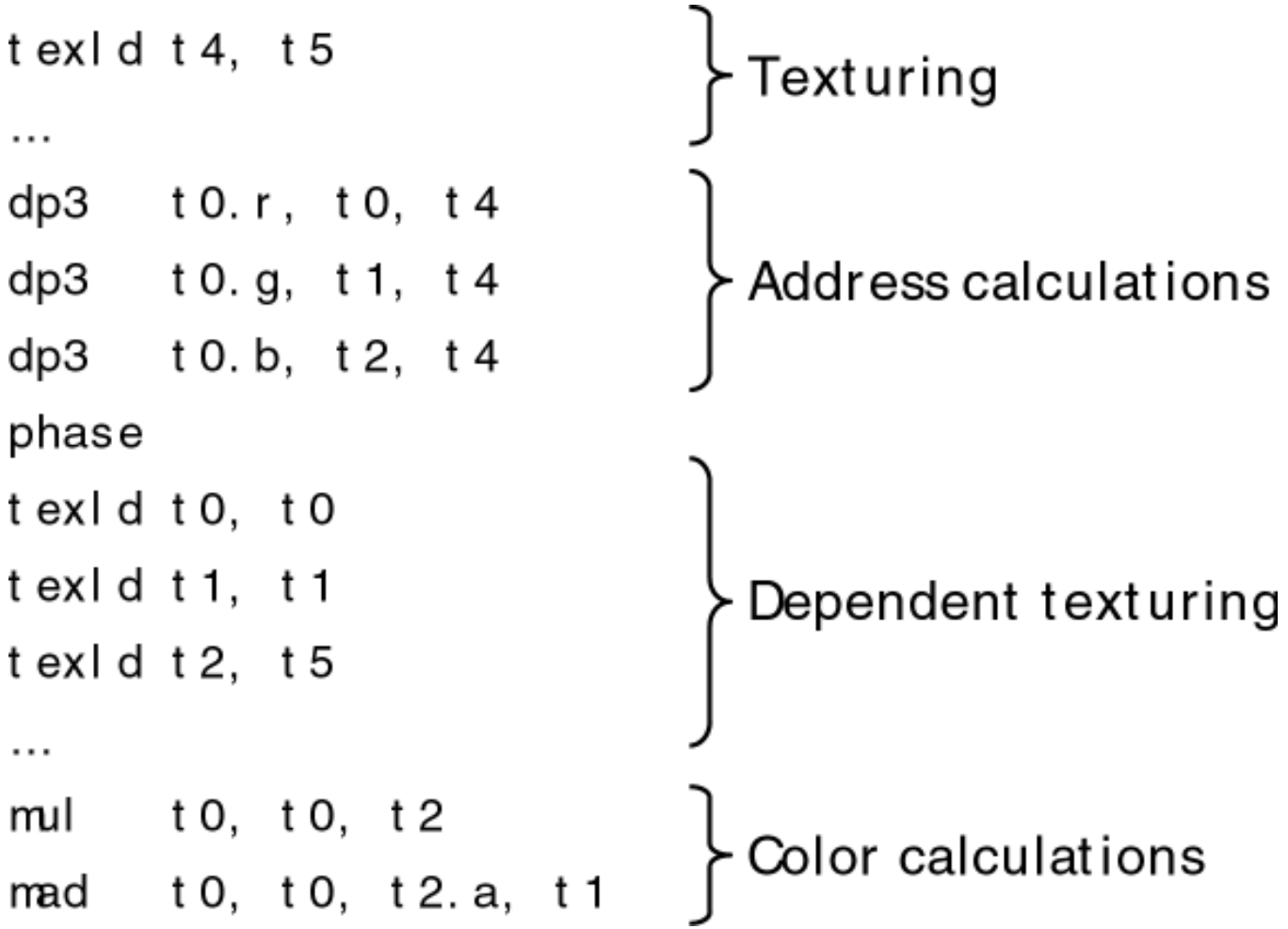
Basic

```
add d, s0, s1
sub d, s0, s1
mul d, s0, s1
mad d, s0, s1, s2  // s0 + s1 * s2
lrp d, s0, s1, s2  // s2 + s0 *(s1 - s2)
cnd d, s0, s1, s2  // (s2 > 0.5) ? s1 : s2
cmp d, s0, s1, s2  // (s2 >= 0.0) ? s1 : s2
dp3 d, s0, s1
dp4 d, s0, s1
```

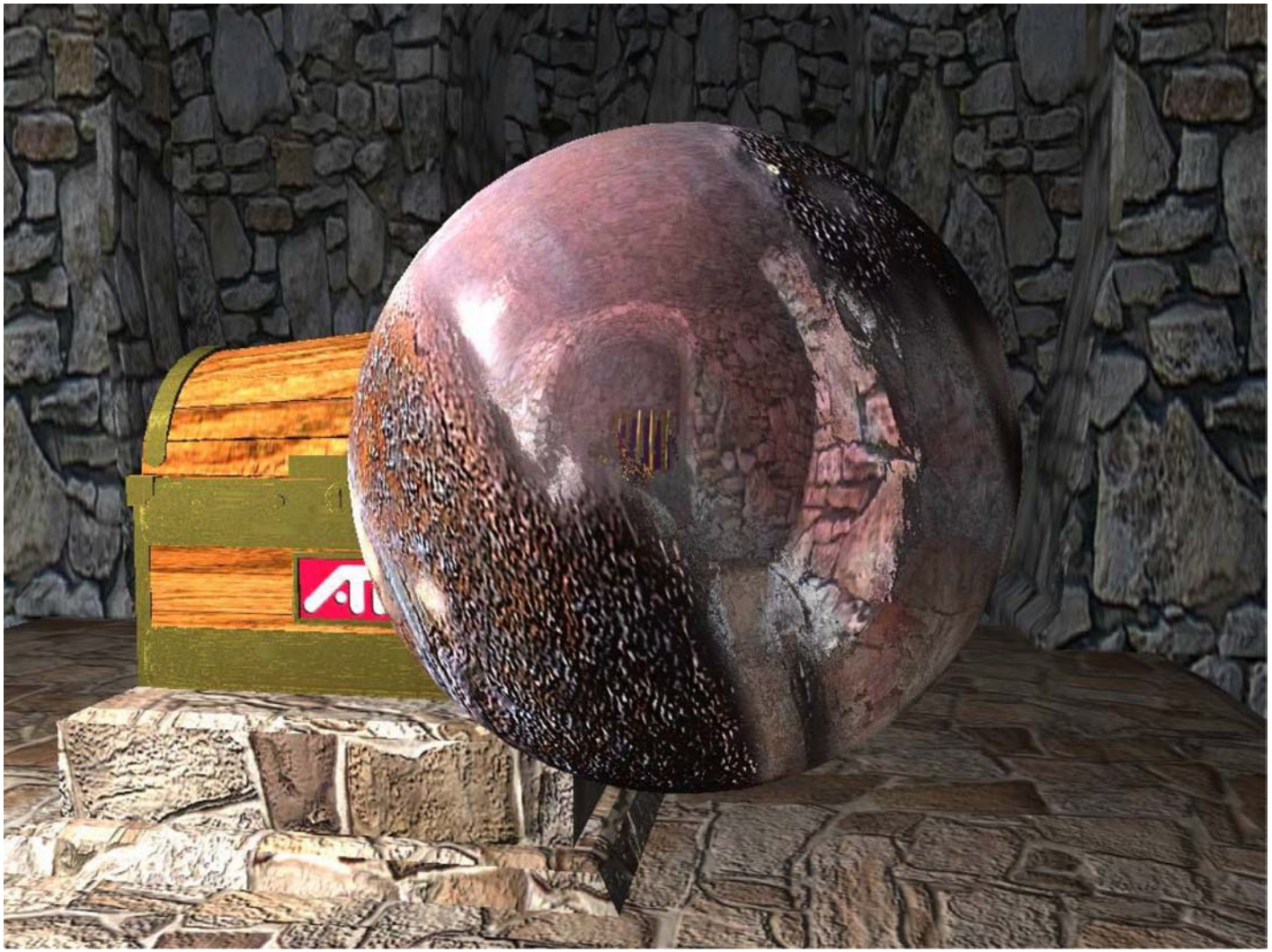
Texturing

```
texld r0, r2      // Sample texture 0 with r2 coords
texcrd r0, t0     // Move coord t0 to r0
texkill t0        // Kill fragment if u,v, or w = 0
texdepth r5       // Use r5.r / r5.g for z test
```

Phases (PS1.4)


texld t4, t5
...
dp3 t0.r, t0, t4
dp3 t0.g, t1, t4
dp3 t0.b, t2, t4
phase
texld t0, t0
texld t1, t1
texld t2, t5
...
mul t0, t0, t2
mad t0, t0, t2.a, t1
Texturing
Address calculations
Dependent texturing
Color calculations

Bumpy Enviroment Mapping



Bumped Environment Mapping

```
ps.1.4
def c0, 1,1,1,1

texld   r0, t0              ; Look up normal map
texcrd  r1.xyz, t4          ; Eye vector
texcrd  r4.xyz, t1          ; 1st row of environment matrix
texcrd  r2.xyz, t2          ; 2st row of environment matrix
texcrd  r3.xyz, t3          ; 3rd row of environment matrix
```

Bumped Environment Mapping

```
ps.1.4
def c0, 1,1,1,1

texld   r0, t0                ; Look up normal map
texcrd  r1.xyz, t4            ; Eye vector
texcrd  r4.xyz, t1            ; 1st row of environment matrix
texcrd  r2.xyz, t2            ; 2st row of environment matrix
texcrd  r3.xyz, t3            ; 3rd row of environment matrix

dp3     r4.x, r4, r0_bx2      ; N.x = 1st row of matrix multiply
dp3     r4.y, r2, r0_bx2      ; N.y = 2nd row of matrix multiply
dp3     r4.z, r3, r0_bx2      ; N.z = 3rd row of matrix multiply
dp3_x2  r3.xyz, r4, r1        ; 2(N.Eye)
mul     r3.xyz, r4, r3        ; 2N(N.Eye)
dp3     r2.xyz, r4, r4        ; N.N
mad     r2.xyz, -r1, r2, r3 ; 2N(N.Eye) - Eye(N.N)

phase
```

Environment Mapping

```
phase

texld   r2, r2                  ; Sample cubic reflection map
texld   r3, t0                  ; Sample base map with gloss in alpha
texld   r4, r4                  ; Sample cubic diffuse map
```

Environment Mapping

```
phase

texld   r2, r2                ; Sample cubic reflection map
texld   r3, t0                ; Sample base map with gloss in alpha
texld   r4, r4                ; Sample cubic diffuse map

mul     r1.rgb, r3.a, r2      ; Specular = Gloss * Reflection
mad     r0.rgb, r3, r4, r1    ; Base*Diffuse + Specular
mov     r0.a, c0.a            ; Put 1.0 in alpha
```

Volume Rendering

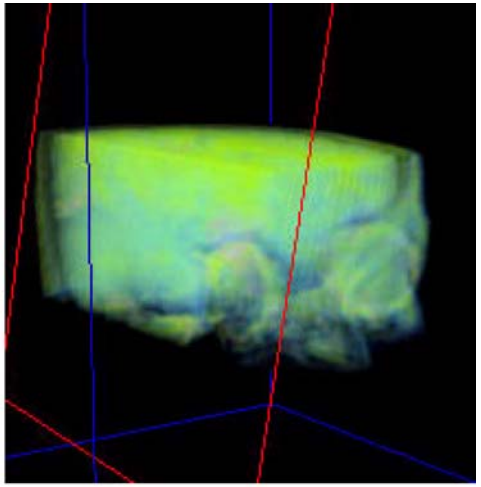
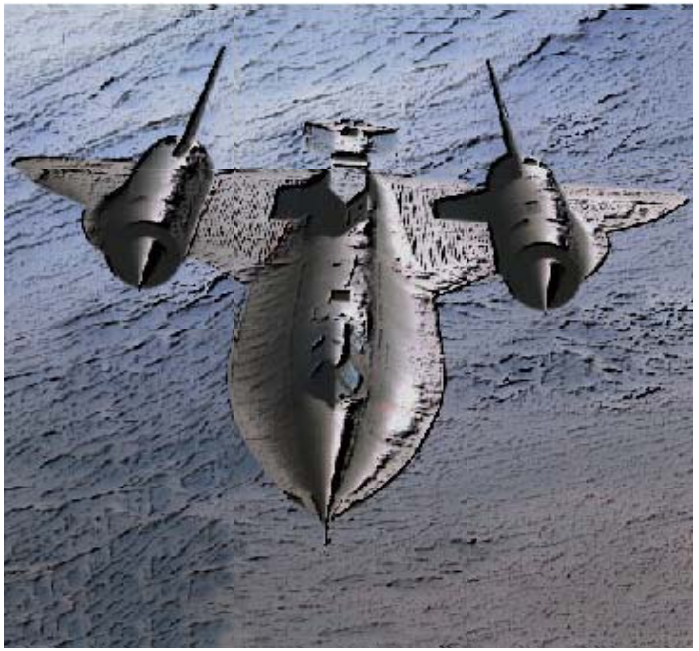
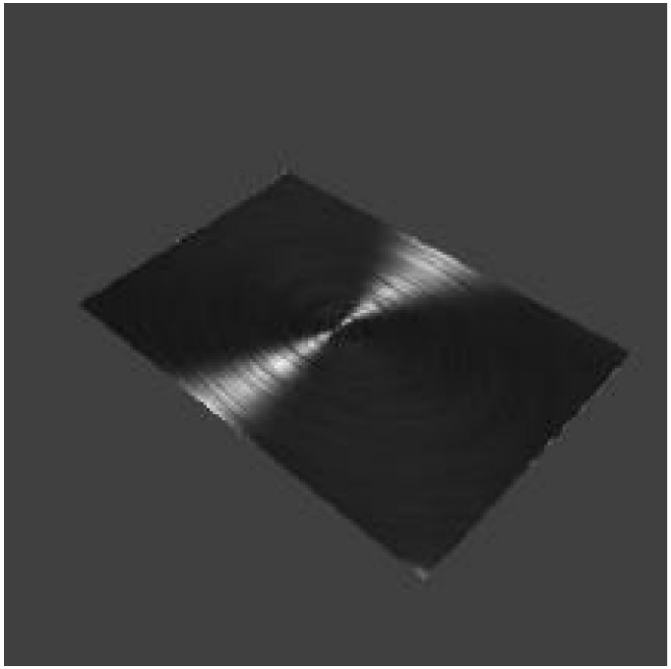


Image Processing



Edge Detection

Materials



Anisotropic lighting

Next Generation

- NV_fragment_program & DirectX 9 PS2.0
- Radeon 9700 & NV30
- Features
  - No more phases
  - Generalized texture lookups
  - Long programs
  - Floating point

Next Generation

Next Generation

```
ps.2.0

def c0, 2.0f, -1.0f, 0.5f, 0.5f
def c1, 1.0f, 1.0f, 0.1f, 0.0f

dcl t0.xyzw
dcl t1.xyzw
dcl t2.xyzw
dcl t3.xyzw
dcl t4.xyzw
dcl t6.xyzw
dcl t7.xyzw

dcl_volume s0
dcl_2d     s1

texld r3,  t0, s0
texld r4,  t1, s0
texld r5,  t2, s0
texld r6, t3, s0
texld r7, t4, s0

mov r3.y, r4.x
mov r3.z, r5.x
mov r6.y, r7.x
mad r6, r6, c0.x, c0.y
mad r3, r3, c0.x, c0.y
mad r7, c3.w, r3, t0
mad r7, c4.w, r6, r7

dp2add r0, r7, r7, c1.w
rsq r0, r0.x
rcp r0, r0.x
mul r0, r0, c2.w

texld r0, r0, s1

mov r1, c3
lrp r2, r0.x, c2, r1
sub r4, c4, t6
dp3 r5.w, r4, r4
rsq r5.w, r5.w
mul r4, r4, r5.w
dp3 r6.w, t7, t7
rsq r6.w, r6.w
mul r5, t7, r6.w
dp3 r3.w, t6, t6
rsq r3.w, r3.w
mul r3, -t6, r3.w
add r6, r3, r5
dp3 r6.w, r6, r6
rsq r6.w, r6.w

mul r6, r6, r6.w
dp3_sat r6, r5, r6
mad r0.z, r0.z, c5.z, c5.w
pow r6, r6.x, r0.z
dp3 r5, r4, r5
mad_sat r5, r5, c0.z, c0.z
mul r6, r6, r0.y
mad r2, r5, r2, r6
mov oC0, r2
```

Next Generation

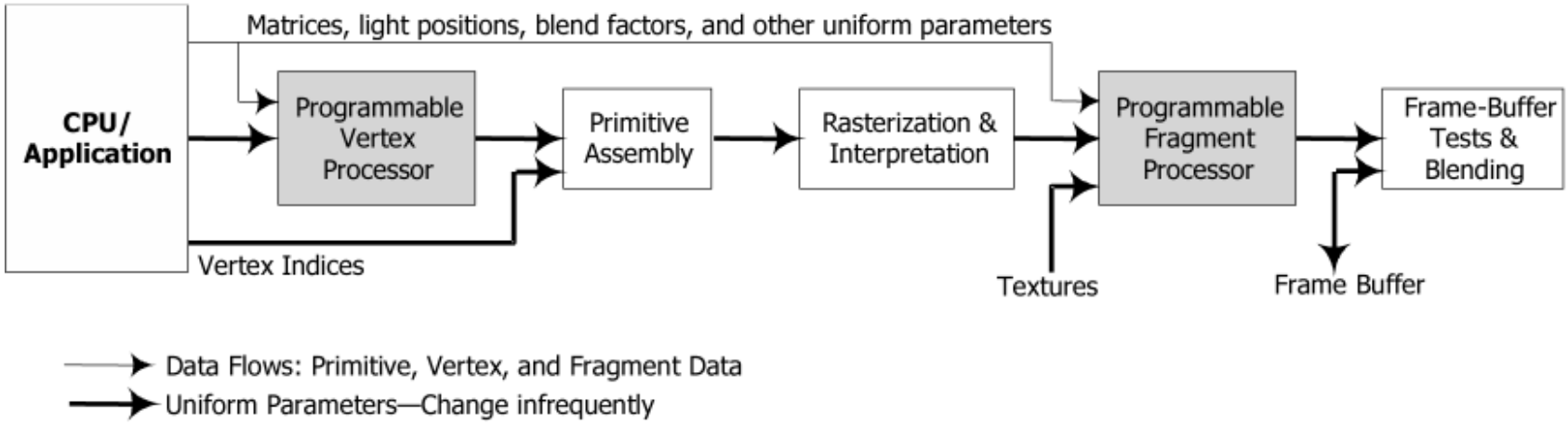
Other Stuff

- Legacy
  - Register Combiners
  - Texture Shaders
  - ATI_fragment_shader
- DX9 & NV_fragment_program coming soon!
  - http://www.ati.com/developer/techpapers.html
  - http://developer.nvidia.com/

CG: C for Graphics

CG

- C for Graphics
- Generalized C-like programming environment
- Compiles to
  - NV_fragment_program
  - DX8 vertex and pixel shaders
  - NV_vertex_program[1,2]



CG General

- Types
  - Scalar: float, half, fixed, bool (no int)
  - Vector: float4, half2, float4x4, …
- Operators
  - C standard operators (a * b)
  - Swizzles and masking: v.xyyw
  - Component-wise vector operations
- Standard Library
  - Basic: pow,sin,floor,dot,lerp,…
  - Special: transpose,noise,…

CG Compiling

- Compiling
  - Offline

```
cgc -profile dx8vs -o foo.vs foo.cg
```

  - Runtime

```
cgGLLoadProgram()
```

- Binding variables
  - Static

```
float4 lightpos : c0
```

  - Runtime

```
cgGLBindUniform4f()
```

CG Vertex Shader

```
// define inputs from application
struct appin
{
    float4 Position : ATTR0;
    float4 Normal   : ATTR2;
};

// define outputs from vertex shader
struct vertout
{
    float4 HPosition    : HPOS;
    float4 Color0       : COL0;
};
```

CG Vertex Shader

```
vertout main(appin IN,
             uniform float4x4 ModelViewProj,
             uniform float4x4 ModelViewIT,
             uniform float4 LightVec)
{
    vertout OUT;

    OUT.HPosition = mul(ModelViewProj, IN.Position);

    // transform normal from model-space to view-space
    float4 normal = normalize(mul(ModelViewIT, IN.Normal).xyzz);

    // store normalized light vector
    float4 light = normalize(LightVec);

    // calculate half angle vector
    float4 eye = float4(0.0, 0.0, 1.0, 1.0);
    float4 half = normalize(light + eye);
```

CG Vertex Shader

```
    // calculate diffuse component
    float diffuse = dot(normal, light);

    // calculate specular component
    float specular = dot(normal, half);
    specular = pow(specular, 32);

    // blue diffuse material
    float4 diffuseMaterial = float4(0.0, 0.0, 1.0, 1.0);

    // white specular material
    float4 specularMaterial = float4(1.0, 1.0, 1.0, 1.0);

    // output final vertex color
    OUT.Color0 = diffuse * diffuseMaterial + specular *
                         specularMaterial;

    return OUT;
}
```

CG Resources

- Main Page
  - http://developer.nvidia.com/view.asp?PAGE=cg_main
- Tutorials
  - http://developer.nvidia.com/view.asp?IO=cg_tutorials
- Open Source Parser
  - http://developer.nvidia.com/view.asp?IO=cg_compiler_code
- CG Effects Browser
  - http://download.nvidia.com/developer/cg/Cg_Browser.exe

Trends

Trends

- "All processors aspire to be general-purpose"
  - Tim van Hook, Keynote, Graphics Hardware 2001

- Unified vertex and fragment shading
- Goal toward movie quality production shading
- Non-graphics related processing