Real-Time
Graphics Architecture

Kurt Akeley

Pat Hanrahan

http://www.graphics.stanford.edu/courses/cs448a-01-fall

Programmable Shading

Topics

Graphics hardware abstractions

Vertex programs

Fragment programs

Trends and observations

Readings

Required

1. M. Peercy, M. Olano, J. Airey, J. Ungar, Interactive multipass programmable shading, SIGGRAPH 2000
2. K. Proudfoot, B. Mark, S. Tvetkov, P. Hanrahan, A real-time programmable shading system for programmable graphics hardware, SIGGRAPH 2001

Recommended

1. M. Olano, A. Lastra, A shading language on graphics hardware: The PixelFlow shading system
2. M. McCool, The SMASH API
3. Quake Arena Shaders Manual

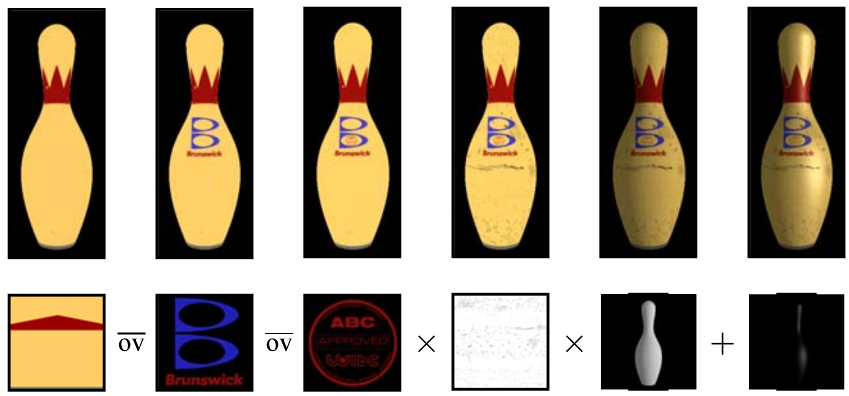
Readings

Background

1. R. Cook, Shade trees, SIGGRAPH 1984
2. K. Perlin, An image synthesizer, SIGGRAPH 1985
3. P. Hanrahan and J. Lawson, A language for shading and lighting calculations, SIGGRAPH 1990.
4. A. Apodaca and L. Gritz, Advanced RenderMan: Creating CGI for Motion Pictures, 2000

Graphics Hardware Abstractions

Multipass Rendering



Base $\overline{\text{ov}}$ Bruns $\overline{\text{ov}}$ Circle $\times$ Marks $\times$ Cd $+$ Cs

1. First pass uses ZLT mode (generate z-buffer)
2. Subsequent passes use ZEQ mode (draw front surface)

OpenGL as an Assembly Language

$$\text{fb} = \left[\text{fb} \left\{ \begin{array}{c} + \\ \times \\ \text{blend} \end{array} \right\} \right] \left\{ \begin{array}{c} \text{C} \\ \text{T} \\ \text{C} \times \text{T} \end{array} \right\} \quad \text{(render)}$$

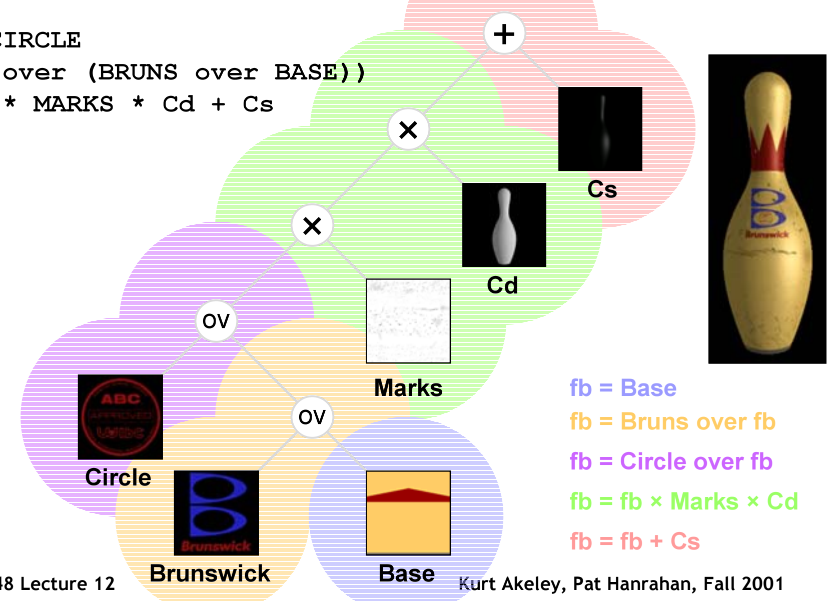
$$\text{T} = \text{fb} \quad \text{(save)}$$

fb = framebuffer (accumulator)
T = texture (memory)
C = triangle colors (interpolated shaded vertices)

Additional capabilities = new instructions
- **EXT_blend_subtract**
- **ARB_multitexture**

Compiling Expressions to Passes

```
(CIRCLE
  over (BRUNS over BASE))
  * MARKS * Cd + Cs
```



Mechanisms

```
if( predicate) body;
```

Conditionals (ala SIMD masks)

- Compute predicate in alpha
- Use alpha test to control setting stencil bit
- Nesting levels assigned to different stencil bits
- MinMax reduction tests if any stencil bits set
- Stencil bits control evaluation of body

Mechanisms

Implementing dependent textures

- Render texture coordinates to texture
- Render using 1st texture as coordinates for 2nd
- OpenGL pixel textures
  - Reconstructs, but doesn't antialias
  - Possible extension uvd-map

Textures as functions

- Univariate math functions stored in 1-D textures
- Multivariate functions: BRDFs, etc.

Extensions

Requires complete set of operators

- For example, OpenGL 1.2 Imaging Extension

Requires enhanced precision

- fp16 floating point format

Analysis: Multipass Abstraction

Advantages

- Layers on top of OpenGL (with few extensions)
- Hides complexity of multipass
- Relatively simple graphics pipeline

Analysis: Multipass Abstraction

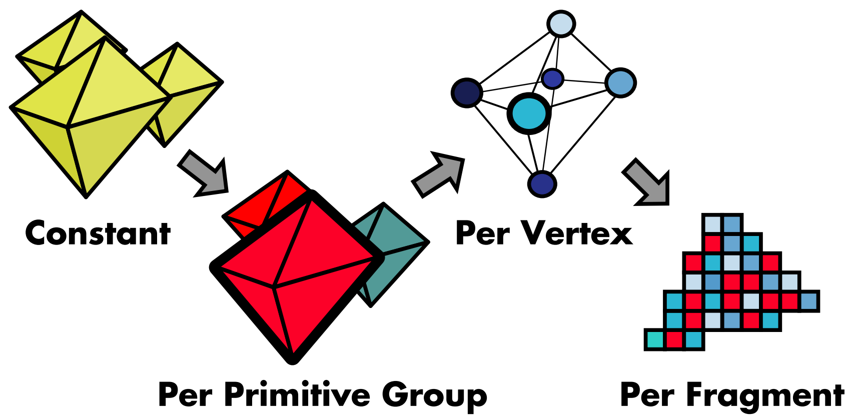
Limitations

- Monopolizes machine (may not mix well with other uses)
- Doesn't work for transparent surfaces
- Doesn't work with antialiasing
- Practically (2001) limited by precision and operators

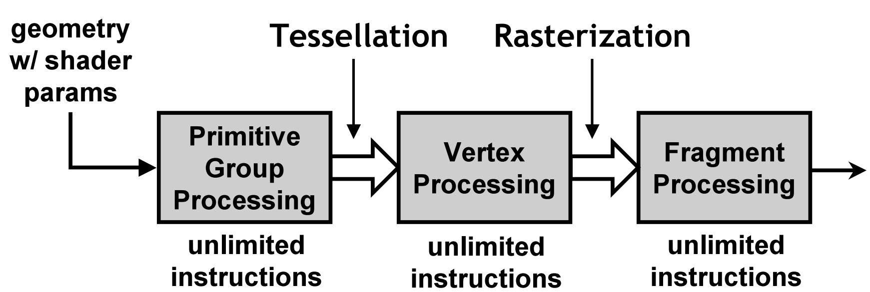
Performance limitations

- Render then copy to texture expensive
  - Requires screen-space bounding box
  - Ideally, render directly to texture, but …
  - F-buffer (fragment FIFO)
- Pushes all programmable vertex computations to fragment
  - Lots of passes

Multiple Computation Frequencies



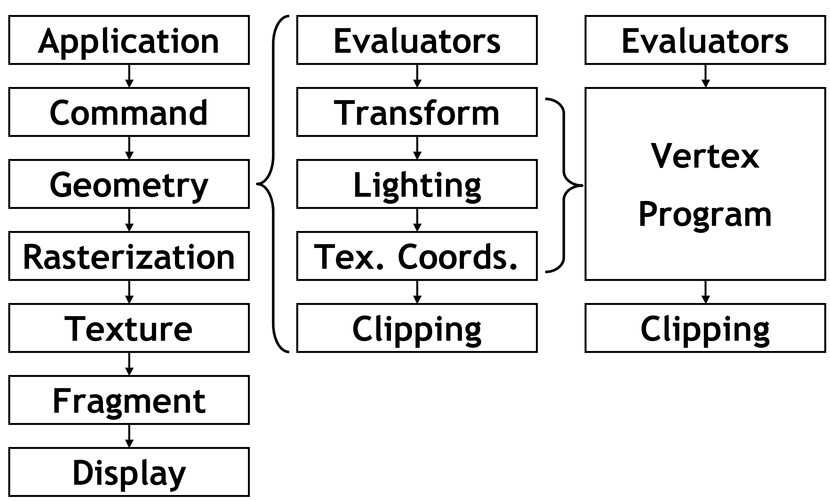
Programmable Pipeline Abstraction



Conceptually one rendering pass

Vertex Programs

Graphics Pipeline



Vertex Programs

Restrict processing to make it easier to parallelize

Restrictions

- Avoid dependencies and ordering constraints
- Avoid 1 to n expansions or n to 1 reductions

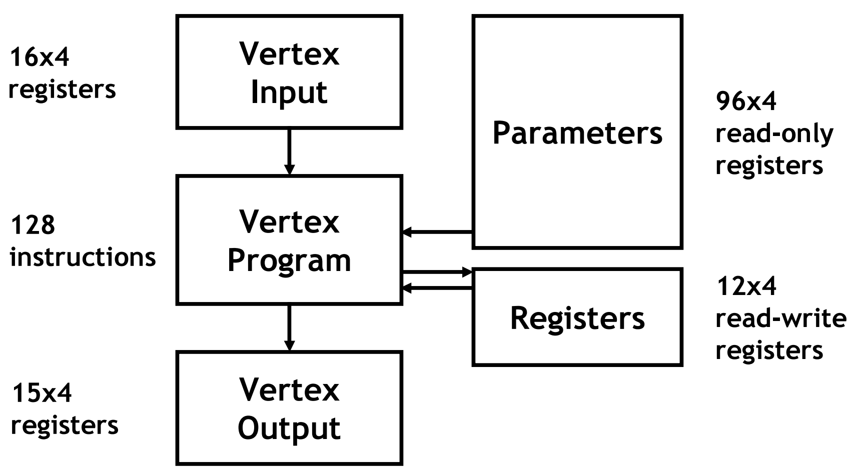
Restrictions create independent tasks

Disallows

- Primitive assembly (dependency)
- Evaluation and tesselation
- Clipping and culling

These tricker cases handled in special-purpose ways

Vertex Program Architecture



Vertex Attributes

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			
7			
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

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 vertex 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)

Basic Instructions (VS1.0)

17 instructions

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

Program Examples

Vector 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;      // swizzle
MAD R2, R0.yzxw, R1.zxyw, -R2; // negation
```

Vector 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;                // write-mask
MUL R0.xyz, R1, R0.w;          // promotion
```


Simple Graphics Pipeline

```
# c[0-3]   = Mat; c[4-7]   = Mat^{-T}
# c[32]    = L; c[33]    = H
# c[35].x = Md * Ld; c[35].y = Ma * La
# c[36]    = Ms; c[38].x = s
DP4   o[HPOS].x, c[0], v[OPOS];      # Transform 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];           # Transform normal.
DP3   R0.y, c[5], v[NRML];
DP3   R0.z, c[6], v[NRML];
DP3   R1.x, c[32], R0;               # R1.x = L DOT N
DP3   R1.y, c[33], R0;               # R1.y = H DOT N
MOV   R1.w, c[38].x;                 # R1.w = s
LIT   R2, R1;                        # Compute lighting
MAD   R3, c[35].x, R2.y, c[35].y;    # diffuse + ambient
MAD   o[COL0].xyz, c[36], R2.z, R3;  # + specular
END
```

LIT Instruction

LIT d, s

s.x = N • L
s.y = N • H
s.z = s **(-128<m<128)**

d.x = 1.0
d.y = CLAMP(N • L, 0, 1)
d.z = CLAMP(N • H, 0, 1)^{s}
d.w = 1.0

Summary

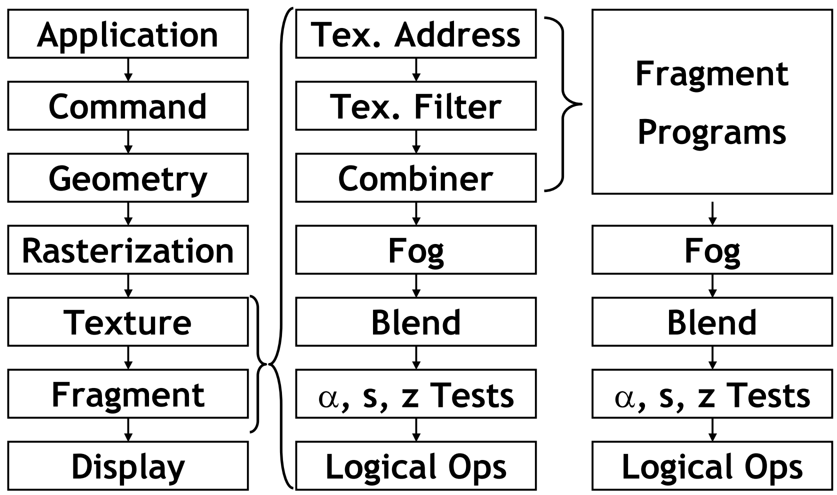
- 4-way SIMD (like SSE)
- Swizzle/negate on all sources
- Write-mask on all destinations
- DP3 and DP4 most common operations
- LIT implements Blinn lighting model
- Limited addressing mechanism
- *No branches or conditionals*

References

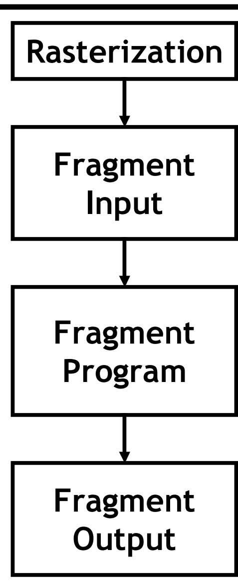
1. E. Lindholm, H. Moreton, M. Kilgard, A user-programmable vertex engine, SIGGRAPH 2001

Fragment Programs

Graphics Pipeline



Fragment Program Architecture (PS)



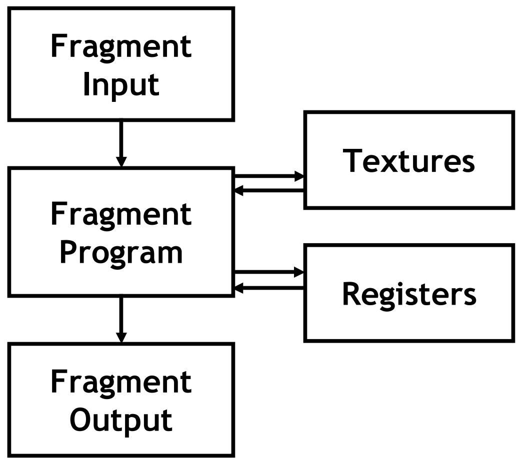
Fragment Input Registers

Register Name	Description	Component Interpretation	Range/ Precision
v0	Diffuse color	(r,g,b,a)	0-1
v1	Specular color	(r,g,b,a)	0-1
t0	Texture coordinate set 0	(s,t,r,q)	-1..1
…			
tn	Texture coordinate set n	(s,t,r,q)	-1..1

Fragment Output Registers

Register Name	Description	Component Interpretation	Range/ Precision
r0	Color	(r,g,b,a)	0..1
r5.r	Depth	(z)	0..1

Fragment Program Architecture



CS448 Lecture 12 Kurt Akeley, Pat Hanrahan, Fall 2001

Fragment Registers and Textures

Constants c0, …, cn

- High precision, range [-1,1]

Registers r0, …, rn

- High precision, extended range

Textures s0, …, sn (future, for now same as t#)

- Different types
- Different dimensionality (1D, 2D, 3D)

CS448 Lecture 12 Kurt Akeley, Pat Hanrahan, Fall 2001

Basic Instructions (PS1.4)

```
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
tex   t0
texld d, t0
texld d, t0_dw;
```

… other operators …

Phases (PS1.4)

`texld t4, t5` …	Texturing

`dp3 t0.r, t0, t4`
`dp3 t0.g, t1, t4`
`dp3 t0.b, t2, t4`
	Address calculations

`phase`	

`texld t0, t0`
`texld t1, t1`
`texld t2, t5`
…
	Dependent texturing

`mul t0, t0, t2` `mad t0, t0, t2.a, t1`	Color calculations

ATI Radeon 8500

Range [-8,8]

Texture coordinates = 6

Textures = 6

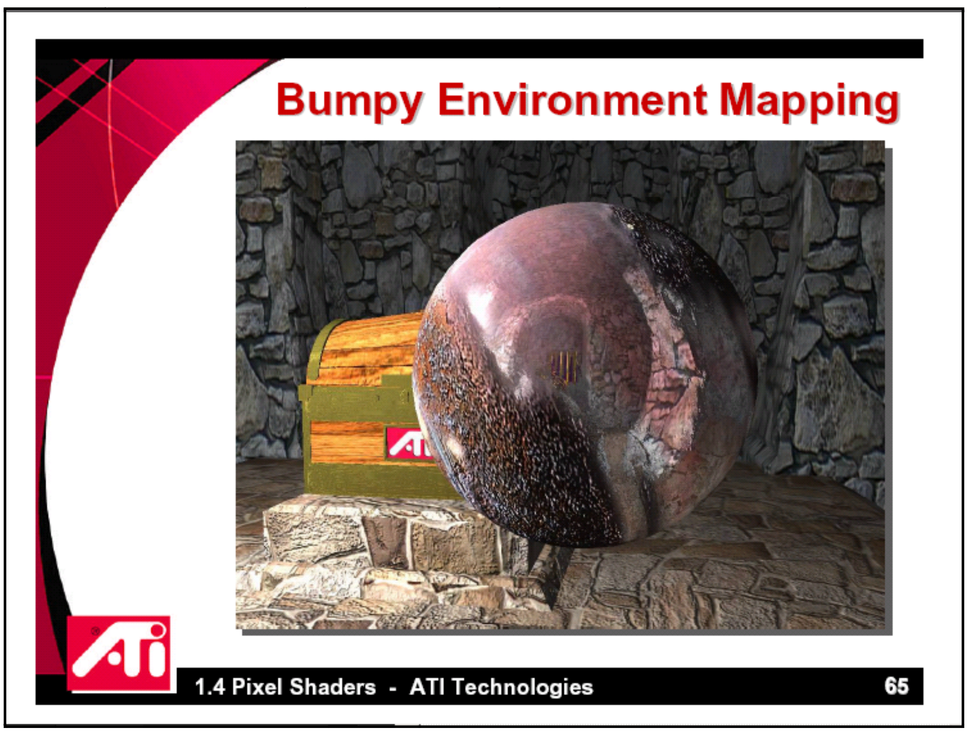
Texturing stages = 6 lookups each

Registers = 6

Constants = 8

Two stages: 1 level of dependency in textures

Addressing and color stages = 8 instructions each



Reflective Bump Mapping

```
Stage 0: Texture
Texld     r0, t0           // Lookup N'
Texld     r1, t4           // Normalize E
Texcrd    r4,rgb, t1       // 1st row of M
Texcrd    r2.rgb, t2       // 2nd row of M
Texcrd    r3.rgb, t3       // 3rd row of M
Texcrd    r5.rgb, t5       // World space L
…
```


Reflective Bump Mapping

```
Stage 1: Texture Addresses
dp3       r4.r, r4, r0.bx2    // M N'
dp3       r4.g, r2, r0.bx2
dp3       r4.b, r2, r0.bx2
dp3_x2    r3.rgb, r4, r1_bx2  // 2 (N.E)
mul       r3.rgb, r4, r3      // 2N (N.E)
dp3       r2.rgb, r4, r4      // N.N
mad       r2.rgb, -r1_bx2, r2, r3 // R
phase
…
```

Reflective Bump Mapping

```
Stage 3: Dependent Textures
Texld     r2, r2          // cubemap(R)
Texld     r3, t0          // Cd(st)
Texld     r4, r4          // cubemap(N)
Texld     r5, t0          // Cs(st)
…
```


Reflective Bump Mapping

```
Stage 1: Final color
mul       r1.rgb, r5, r2
mad       r0.rgb, r3, r4_x2, r1
…
```

DX9 PS2.0 Proposal

Extended range and precision

Vertex program-like instruction set (no cond.)

Color interpolants = 2

Texture coordinates = 8

Textures = 16 (separate from texture coordinates)

Dependent textures = 4 levels with no phases

Registers = 16

Constants = 32

Addressing operations = 32

Mathematical operations = 64

References

ATI

1. J. Mitchell, EGDC 2001 Conference Presentation
2. DX8.1 Presentation at Meltdown 2001

NVIDIA – Very different architecture!

1. NVIDIA Texture Shader Presentation
2. NVIDIA Register Combiner Presentation

Microsoft

1. DX9.0 Presentation at Meltdown 2001

Analysis: Programmable Pipeline

Support for multiple computational frequencies

- Natural model of the graphics pipeline

Better match to current hardware

- Current chips programmable
- OpenGL and DX8 have exposed programmability

Reduced off-chip bandwidth

- More ops per pass means fewer passes
- Limited AGP bandwidth
- Limited FB and texture memory bandwidth

Downside: system much more complicated

Hardware Resource Constraints

Inputs and outputs

- Host to vertex
- Framebuffer

Constants

Interpolants

Textures

Dependent textures (levels of dependency)

Colors

Registers

Instructions

Virtualization

Use multipass … recalling previous caveats

Key: Save and restore registers

- Exact copy (no precision reduction)
- Equivalent texture formats
- Multiple outputs

Trends

Transition in Graphics Systems

Past

- Fixed-function pipelines
- Feature-based interfaces

Present

- Limited programmability
- Assembly-language interfaces

Future

- General programmability

More Generality

"All processors aspire to be general-purpose"

- Tim Van Hook, Keynote, Graphics Hardware 2001

Quickly

- General-purpose instruction set
  - Full set of arithmetic operators
  - Clean orthogonal design
  - Conditional branches
- High-precision data types
- Convergence between vertex and fragment programs

Shading Languages

Hardware is difficult to use

- Programming in low-level assembly language
- Coordinating multiple programs

Hardware level is non-portable

- Rapidly varying APIs as hardware evolves
- Variation between vendors

Shading languages

- Proven technology in the movie industry
- Stanford Real-Time Shading Language
- 3DLabs has proposed a shading languge for OpenGL 2.0

Hardware designed for compilability?

Virtual Graphics Pipeline

Meta-graphics pipeline?

- Reprogram pipeline
- Implement currently non-programmable stages
  - Rasterization and tesselation
  - Built-in fragment operations, e.g. alpha test
- Introduce new programmable stages

Built-in hardware functions

- Rasterization and texturing …

Simplify?

Specialize?

General Stream Processor

General-purpose data parallel computer

- Collections
  - Sets and lists
  - Arrays
  - Graphs and meshes
- Operators
  - Map (apply function)
  - Filter
  - Gather, scatter, permute
  - Expansion
  - Reduction

Implications for computing?