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
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
4. A. Apodaca and L. Gritz, Advanced RenderMan: Creating CGI for Motion Pictures, 2000

Graphics Hardware Abstractions
Multipass Rendering

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

OpenGL as an Assembly Language

\[
fb = \left[ \begin{array}{c}
\text{fb} \\
\text{blend}
\end{array} \right] + \left[ \begin{array}{c}
C \\
T
\end{array} \right]
\]

\[
T = \text{fb}
\]

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

\[
\text{fb} = \text{ brut } \times \text{ marks } \times \text{ cd } + \text{ cs }
\]

Mechanisms

\[
\text{if( predicate) body;}
\]

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

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

- Constant
- Per Primitive Group
- Per Vertex
- Per Fragment

Programmable Pipeline Abstraction

- Geometry w/ shader params
- Tessellation
- Rasterization
- Primitive Group Processing (unlimited instructions)
- Vertex Processing (unlimited instructions)
- Fragment Processing (unlimited instructions)

Conceptually one rendering pass
Vertex Programs

Graphics Pipeline

Application
  Command
  Geometry
  Rasterization
  Texture
  Fragment
  Display

Evaluators
  Transform
  Lighting
  Tex. Coords.
  Clipping

Evaluators
  Vertex Program

Clipping
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 Input -----> Parameters
               ^                        Up
               |                        |
           128 instructions
               |                        |
Vertex Program
               |                        |   96x4 read-only registers
               |                        |   12x4 read-write registers
               |                        |
Vertex Output

16x4 registers
128 instructions
15x4 registers
```
### Vertex Attributes

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

### Vertex Input Registers

<table>
<thead>
<tr>
<th>Attribute Register</th>
<th>Mnemonic Name</th>
<th>Typical Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>v[0]</td>
<td>v[OPOS]</td>
<td>object position</td>
</tr>
<tr>
<td>v[1]</td>
<td>v[WGHT]</td>
<td>vertex weight</td>
</tr>
<tr>
<td>v[3]</td>
<td>v[COL0]</td>
<td>primary color</td>
</tr>
<tr>
<td>v[6]</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>v[7]</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>v[8]</td>
<td>v[TEX0]</td>
<td>texture coordinate 0</td>
</tr>
<tr>
<td>v[9]</td>
<td>v[TEX1]</td>
<td>texture coordinate 1</td>
</tr>
</tbody>
</table>

Semantics defined by vertex program NOT parameter name!
### Vertex Output Registers

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

### Basic Instructions (VS1.0)

17 instructions

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

Vector Cross Product
\[
\begin{array}{c|ccc}
\# & i & j & k \\
\# & R0.x & R0.y & R0.z \\
\# & R1.x & R1.y & R1.z \\
\end{array}
\]
MUL R2, R0.zxyw, R1.yzxw; // swizzle
MAD R2, R0.yzxw, R1.zxyw, -R2; // negation

Vector Normalize
\[
\begin{array}{c}
\# \text{ R1} = (nx, ny, nz) \\
\# \text{ R0.xyz} = \text{normalize(R1)} \\
\# \text{ R0.w} = 1/\sqrt{nx*nx + ny*ny + nz*nz} \\
\end{array}
\]
DP3 R0.w, R1, R1;
RSQ R0.w, R0.w; // write-mask
MUL R0.xyz, R1, R0.w; // promotion

Simple Graphics Pipeline
\[
\begin{array}{c}
\# \text{ c[0-3]} = \text{Mat} ; \text{c[4-7]} = \text{Mat}^{-T} \\
\# \text{ c[32]} = L ; \text{c[33]} = H \\
\# \text{ c[35].x} = \text{Md} \ast \text{Ld} ; \text{c[35].y} = \text{Ma} \ast \text{La} \\
\# \text{ c[36]} = \text{Ms} ; \text{c[38].x} = s \\
\end{array}
\]
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

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Description</th>
<th>Component Interpretation</th>
<th>Range/Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>(v_0)</td>
<td>Diffuse color</td>
<td>((r,g,b,a))</td>
<td>0-1</td>
</tr>
<tr>
<td>(v_1)</td>
<td>Specular color</td>
<td>((r,g,b,a))</td>
<td>0-1</td>
</tr>
<tr>
<td>(t_0)</td>
<td>Texture coordinate set 0</td>
<td>((s,t,r,q))</td>
<td>-1..1</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(t_n)</td>
<td>Texture coordinate set n</td>
<td>((s,t,r,q))</td>
<td>-1..1</td>
</tr>
</tbody>
</table>

### Fragment Output Registers

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Description</th>
<th>Component Interpretation</th>
<th>Range/Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>(r_0)</td>
<td>Color</td>
<td>((r,g,b,a))</td>
<td>0..1</td>
</tr>
<tr>
<td>(r_{5,r})</td>
<td>Depth</td>
<td>((z))</td>
<td>0..1</td>
</tr>
</tbody>
</table>
Fragment Program Architecture

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)
Basic Instructions (PS1.4)

```plaintext
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)

```plaintext
texld t4, t5  // Texturing
...
dp3  t0.r, t0, t4  // Address calculations
dp3  t0.g, t1, t4
dp3  t0.b, t2, t4
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

Bumpy Environment Mapping
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
...

CS448 Lecture 12
Kurt Akeley, Pat Hanrahan, Fall 2001
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)

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 language for OpenGL 2.0

Hardware designed for compilability?

Virtual Graphics Pipeline

Meta-graphics pipeline?
- Reprogram pipeline
- Implement currently non-programmable stages
  - Rasterization and tessellation
  - 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?