

Shaders

Ulf Borgenstam
Jonas Svensson
Chalmers University of Technology

OpenGL ARB Vertex Program Overview (1)

Traditional Graphics Pipeline:

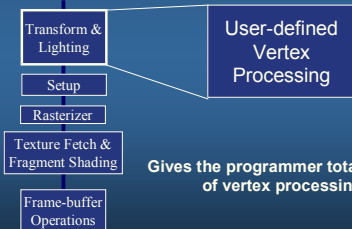


Each unit has specific function (usually with configurable "modes" of operation)

2

OpenGL ARB Vertex Program Overview (2)

Vertex Programming offers Programmable T&L unit:



Gives the programmer total control of vertex processing.

3

What is Vertex Programming? (1)

- Complete control of transform and lighting HW
- Complex vertex operations accelerated in HW
- Custom vertex lighting
- Custom skinning and blending
- Custom texture coordinate generation
- Custom texture matrix operations
- Custom vertex computations of your choice
- Offloading vertex computations frees up CPU

4

What is Vertex Programming? (2)

- Vertex Program
 - Assembly language interface to T&L unit
 - GPU instruction set to perform all vertex math
 - Input: arbitrary vertex attributes
 - Output: transformed vertex attributes
 - homogeneous clip space position (required)
 - colors (front/back, primary/secondary)
 - fog coordinates
 - texture coordinates
 - point size

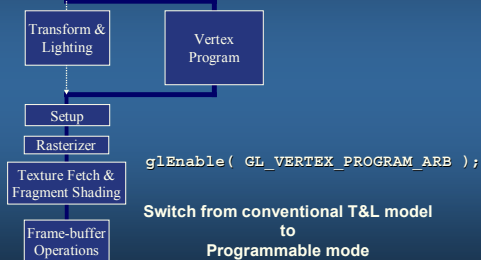
5

What is Vertex Programming? (3)

- Vertex Program
 - Does not generate or destroy vertices
 - 1 vertex in and 1 vertex out
 - No topological information provided
 - No edge, face, nor neighboring vertex info
 - Dynamically loadable
 - Exposed through NV_vertex_program and EXT_vertex_shader extensions and ARB_vertex_program

6

What is Vertex Programming? (4)



7

Specifically, what gets bypassed?

- Modelview and projection vertex transformations
- Vertex weighting/blending
- Normal transformation, rescaling, normalization
- Color material
- Per-vertex lighting
- Texture coordinate generation and texture matrix transformations
- Per-vertex point size and fog coordinate computations
- User-clip planes

8

What does NOT get bypassed?

- Clipping to the view frustum
- Perspective divide
- Viewport transformation
- Depth range transformation
- Front and back color selection (for two-sided)
- Clamping of primary and secondary colors to [0,1]
- Primitive assembly, setup, rasterization, blending

9

Creating a Vertex Program (1)

- Programs are arrays of GLubyte ("strings")
- Created/managed similar to texture objects
 - notion of a *program object*
 - `glGenProgramsARB(sizei n, uint *ids)`
 - `glBindProgramARB(enum target, uint id)`
 - `glProgramStringARB(enum target, enum format, sizei len, const ubyte *program)`

10

Creating a Vertex Program (2)

```
GLuint progid;

// Generate a program object handle.
glGenProgramsARB(1, &progid);

// Make the "current" program object progid.
glBindProgramARB(GL_VERTEX_PROGRAM_ARB, progid);

// Specify the program for the current object.
glProgramStringARB(GL_VERTEX_PROGRAM_ARB,
    GL_PROGRAM_FORMAT_ASCII_ARB,
    strlen(myString), myString);

// Check for errors and warnings...
```

11

Creating a Vertex Program (3)

```
// Check for errors and warnings...
if (GL_INVALID_OPERATION == glGetError())
{
    // Find the error position
    GLuint errPos;
    glGetIntegerv(GL_PROGRAM_ERROR_POSITION_ARB,
        &errPos);

    // Print implementation-dependent program
    // errors and warnings string.
    GLubyte *errString;
    glGetString(GL_PROGRAM_ERROR_STRING_ARB,
        &errString);

    fprintf(stderr, "error at position: %d\n%s\n",
        errPos, errString);
}
```

12

Creating a Vertex Program (4)

When finished with a program object, delete it

```
// Delete the program object.  
glDeleteProgramsARB(1, &progid);
```

13

Specifying Program Parameters

- Three types

- Vertex Attributes – specifiable per-vertex
- Program Local Parameters
- Program Environment Parameters

Program Parameters modifiable outside of a Begin/End block

14

Specifying Program Local Parameters

- Each program object has an array of ($N \geq 96$) four- component floating point vectors
 - Store program-specific parameters required by the program
- Values specified with new commands
 - `glProgramLocalParameter4fARB(GL_VERTEX_PROGRAM_ARB, index, x, y, z, w)`
 - `glProgramLocalParameter4fvARB(GL_VERTEX_PROGRAM_ARB, index, params)`
- Correspond to 96+ local parameter registers

15

Specifying Program Environment Parameters

- Shared array of ($N \geq 96$) four-component registers accessible by any vertex program
 - Store parameters common to a set of program objects (i.e. Modelview matrix, MVP matrix)
- Values specified with new commands
 - `glProgramEnvParameter4fARB(GL_VERTEX_PROGRAM_ARB, index, x, y, z, w)`
 - `glProgramEnvParameter4fvARB(GL_VERTEX_PROGRAM_ARB, index, params)`
- Correspond to 96+ environment registers

16

Program Environment and Program Local Registers

- Program environment registers
 - access using: `program.env[i]`
 - `i` in `[0, GL_MAX_PROGRAM_ENV_PARAMETERS_ARB-1]`
- Program local registers
 - access using: `program.local[i]`
 - `i` in `[0, GL_MAX_PROGRAM_LOCAL_PARAMETERS_ARB-1]`

17

Vertex Attribute Registers

Attribute Register	Components	Underlying State
vertex.position	(x,y,z,w)	object position
vertex.weight	(w,w,w,w)	vertex weights 0-3
vertex.weight[n]	(w,w,w,w)	vertex weights n-n+3
vertex.normal	(x,y,z,1)	normal
vertex.color	(r,g,b,a)	primary color
vertex.color.primary	(r,g,b,a)	primary color
vertex.color.secondary	(r,g,b,a)	secondary color
vertex.fogcoord	(f,0,0,1)	fog coordinate
vertex.texcoord	(s,t,r,q)	texture coordinate, unit 0
vertex.texcoord[n]	(s,t,r,q)	texture coordinate, unit n
vertex.matrixindex	(i,i,i,i)	vertex matrix indices 0-3
vertex.matrixindex[n]	(i,i,i,i)	vertex matrix indices n-n+3
vertex.attrib[n]	(x,y,z,w)	generic vertex attribute n

Semantics defined by program, NOT parameter name

18

Vertex Result Registers

Result Register	Components	Description
result.position	(x,y,z,w)	position in clip coordinates
result.color	(r,g,b,a)	front-facing, primary color
result.color.primary	(r,g,b,a)	front-facing, primary color
result.color.secondary	(r,g,b,a)	front-facing, secondary color
result.color.front	(r,g,b,a)	front-facing, primary color
result.color.front.primary	(r,g,b,a)	front-facing, primary color
result.color.front.secondary	(r,g,b,a)	front-facing, secondary color
result.color.back	(r,g,b,a)	back-facing, primary color
result.color.back.primary	(r,g,b,a)	back-facing, primary color
result.color.back.secondary	(r,g,b,a)	back-facing, secondary color
result.fogcoord	(f,*,*,*)	fog coordinate
result.pointsize	(s,*,*,*)	point size
result.texcoord	(s,t,r,q)	texture coordinate, unit 0
result.texcoord[n]	(s,t,r,q)	texture coordinate, unit n

Semantics defined by down-stream pipeline stages

19

Temporary Variables

- Four-component floating-point vectors used to store intermediate computations
- Temporary variables declared before first use
 - TEMP flag;
 - TEMP tmp1, tmp2, tmp3;
- Number of temporary variables limited to `GL_MAX_PROGRAM_TEMPORARIES_ARB`

20

Program Parameter Variable Bindings

- Explicit Constant Binding
- Single Declaration
 - PARAM a = {1.0, 2.0, 3.0, 4.0}; (1.0, 2.0, 3.0, 4.0)
 - PARAM b = {3.0}; (3.0, 0.0, 0.0, 1.0)
 - PARAM c = {1.0, 2.0}; (1.0, 2.0, 0.0, 1.0)
 - PARAM d = {1.0, 2.0, 3.0}; (1.0, 2.0, 3.0, 1.0)
 - PARAM e = 3.0; (3.0, 3.0, 3.0, 3.0)
- Array Declaration
 - PARAM arr[2] = { {1.0, 2.0, 3.0, 4.0}, {5.0, 6.0, 7.0, 8.0} }; (1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0)

21

Program Parameter Variable Bindings

- Implicit Constant Binding
 - ADD a, b, {1.0, 2.0, 3.0, 4.0}; (1.0, 2.0, 3.0, 4.0)
 - ADD a, b, {3.0}; (3.0, 0.0, 0.0, 1.0)
 - ADD a, b, {1.0, 2.0}; (1.0, 2.0, 0.0, 1.0)
 - ADD a, b, {1.0, 2.0, 3.0}; (1.0, 2.0, 3.0, 1.0)
 - ADD a, b, 3.0; (3.0, 3.0, 3.0, 3.0)
- Number of program parameter variables (explicit+implicit) limited to `GL_MAX_PROGRAM_PARAMETERS_ARB`

22

Program Parameter Variable Bindings

- Program Environment/Local Parameter Binding
 - PARAM a = program.local[8];
 - PARAM b = program.env[9];
 - PARAM arr[2] = program.local[4..5];
 - PARAM mat[4] = program.env[0..3];
- Essentially creates a “Reference”

23

Program Parameter Variable Bindings

Binding	Components	Underlying GL state
state.material.ambient	(r,g,b,a)	front ambient material color
state.material.diffuse	(r,g,b,a)	front diffuse material color
state.material.specular	(r,g,b,a)	front specular material color
state.material.emission	(r,g,b,a)	front emissive material color
state.material.shininess	(s,0,0,1)	front material shininess
state.material.front.ambient	(r,g,b,a)	front ambient material color
state.material.front.diffuse	(r,g,b,a)	front diffuse material color
state.material.front.specular	(r,g,b,a)	front specular material color
state.material.front.emission	(r,g,b,a)	front emissive material color
state.material.front.shininess	(s,0,0,1)	front material shininess
state.material.back.ambient	(r,g,b,a)	back ambient material color
state.material.back.diffuse	(r,g,b,a)	back diffuse material color
state.material.back.specular	(r,g,b,a)	back specular material color
state.material.back.emission	(r,g,b,a)	back emissive material color
state.material.back.shininess	(s,0,0,1)	back material shininess

24

Vertex Programming Assembly Language

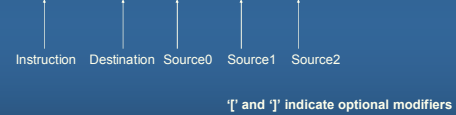
- Powerful SIMD instruction set
- Four operations simultaneously
- 27 instructions
- Operate on scalar or 4-vector input
- Result in a vector or replicated scalar output

25

Assembly Language

Instruction Format:

Opcode dst, [-]s0 [, [-]s1 [, [-]s2]; #comment



Examples:

```
MOV R1, R2;
MAD R1, R2, R3, -R4;
```

26

Assembly Language

Source registers can be negated:

```
MOV R1, -R2;
```

before		after	
R1	R2	R1	R2
0.0 x	7.0 x	-7.0 x	7.0 x
0.0 y	3.0 y	-3.0 y	3.0 y
0.0 z	6.0 z	-6.0 z	6.0 z
0.0 w	2.0 w	-2.0 w	2.0 w

27

Assembly Language

Source registers can be "swizzled":

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

before		after	
R1	R2	R1	R2
0.0 x	7.0 x	3.0 x	7.0 x
0.0 y	3.0 y	6.0 y	3.0 y
0.0 z	6.0 z	2.0 z	6.0 z
0.0 w	2.0 w	7.0 w	2.0 w

28

Assembly Language

Destination register can mask which components are written to...

- R1 ⇒ write all components
- R1.x ⇒ write only x component
- R1.xw ⇒ write only x, w components

29

Vertex Programming Assembly Language

Destination register masking:

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

before		after	
R1	R2	R1	R2
0.0 x	7.0 x	-7.0 x	7.0 x
0.0 y	3.0 y	0.0 y	3.0 y
0.0 z	6.0 z	0.0 z	6.0 z
0.0 w	2.0 w	-2.0 w	2.0 w

30

Vertex Programming Assembly Language

- ABS • EX2 • MAD • RSQ
- ADD • EXP • MAX • SGE
- ARL • FLR • MIN • SLT
- DP3 • FRC • MOV • SUB
- DP4 • LG2 • MUL • SWZ
- DPH • LIT • POW • XPD
- DST • LOG • RCP

31

Example Program

Simple Transform to CLIP space

```
!!ARBv1.0
ATTRIB pos = vertex.position;
PARAM mat[4] = { state.matrix.mvp };

# Transform by concatenation of the
# MODELVIEW and PROJECTION matrices.
DP4 result.position.x, mat[0], pos;
DP4 result.position.y, mat[1], pos;
DP4 result.position.z, mat[2], pos;
DP4 result.position.w, mat[3], pos;

# Pass the primary color through w/o lighting.
MOV result.color, vertex.color;

END
```

32

Querying Implementation-specific Limits

- **Max number of instructions**
`glGetProgramivARB(GL_VERTEX_PROGRAM_ARB,
GL_MAX_PROGRAM_INSTRUCTIONS, &maxInsts);`
- **Max number of temporaries**
`glGetProgramivARB(GL_VERTEX_PROGRAM_ARB,
GL_MAX_PROGRAM_TEMPORARIES, &maxTemps);`
- **Max number of program parameter bindings**
`glGetProgramivARB(GL_VERTEX_PROGRAM_ARB,
GL_MAX_PROGRAM_PARAMETERS, &maxParams);`
- **Others (including native limits)**

Query current program resource usage by removing "MAX_"

33

Generic and Conventional Attribute Mappings

Conventional Attribute	Generic Attribute
vertex.position	vertex.attrib[0]
vertex.weight	vertex.attrib[1]
vertex.weight[0]	vertex.attrib[1]
vertex.normal	vertex.attrib[2]
vertex.color	vertex.attrib[3]
vertex.color.primary	vertex.attrib[3]
vertex.color.secondary	vertex.attrib[4]
vertex.fogcoord	vertex.attrib[5]
vertex.texcoord	vertex.attrib[8]
vertex.texcoord[0]	vertex.attrib[8]
vertex.texcoord[1]	vertex.attrib[9]
vertex.texcoord[2]	vertex.attrib[10]
vertex.texcoord[3]	vertex.attrib[11]
vertex.texcoord[4]	vertex.attrib[12]
vertex.texcoord[5]	vertex.attrib[13]
vertex.texcoord[6]	vertex.attrib[14]
vertex.texcoord[7]	vertex.attrib[15]

In practice, probably use either conventional or generic not both

34

Wrap-Up

- Increased programmability
 - Customizable engine for transform, lighting, texture coordinate generation, and more.
- Vendor extensions available for dynamic branching
 - Will show up as ARBv2 soon?

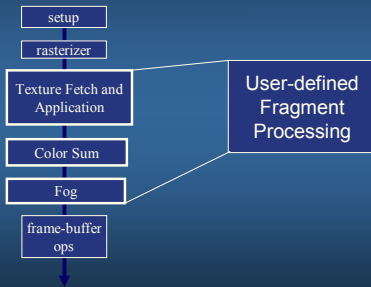
35

OpenGL ARB Fragment Program Overview

- ARB standard for fragment-level programmability
- Derived from ARB_vertex_program
- Replaces
 - Texture blending
 - Color Sum
 - Fog

36

Fragment Processing Pipe



37

Preserves Backend Operations

- Alpha Test
- Stencil and depth test
- Blending
- Coverage application

38

Programming Model

- ASM based similar to ARB_vertex_program
- Rich SIMD instruction set
- Requires resource management
 - Texture accesses
 - Interpolators
 - Temporaries
 - Constants

39

Scalar Instructions

- COS – cosine
- EX2 – base 2 exponential
- LG2 – base 2 logarithm
- RCP – reciprocal
- RSQ – reciprocal square root
- SIN – sine
- SCS – sine and cosine
- POW – power

40

New Scalar Instructions

- COS – cosine
- SIN – sine
- SCS – sine and cosine

41

Removed Scalar Instructions

- EXP – Partial Precision EX2
- LOG – Partial Precision LG2

42

Standard Arithmetic Ops

- ABS – absolute value
- FLR – floor
- FRC – fraction component
- SUB – subtract
- XPD – cross product
- CMP – compare
- LRP – linearly interpolate
- MAD – multiply accumulate
- MOV – move
- ADD – add
- DP3 – three component dot product
- DP4 – four component dot product

43

Special Vector Ops

- LIT – compute lighting
- DPH – homogeneous dot product
- DST – compute distance vector

44

New Instructions

- LRP – Linearly Interpolate
- CMP - Compare

45

Texture Instructions

- TEX
- TXP
- TXB
- KIL

46

Standard Texture Fetch

```
TEX <dest>, <src>, texture[n],  
  <type>;
```

- Does not divide by q

47

Extended Texture Fetches

- TXP
 - Same syntax as TEX
 - Divides first three components by the fourth
- TXB
 - Adds the fourth component to the computed LOD

48

Killing pixels

- KIL instruction
 - Terminates shader program if any component is less than 0

49

Identifying Limits

- Standard Resource limits
 - Number of temps etc
- Texture Indirections

50

Shared API

```
glGenProgramsARB(num, &id);
glBindProgramARB(GL_FRAGMENT_PROGRAM_ARB,
id);
glProgramStringARB(
GL_FRAGMENT_PROGRAM_ARB,
GL_PROGRAM_FORMAT_ASCII_ARB, length,
string);
```

51

Simple Shader Example

```
!!ARBfp1.0

TEMP temp; #temporary
ATTRIB tex0 = fragment.texcoord[0];
ATTRIB col0 = fragment.color;

PARAM pink = { 1.0, 0.4, 0.4, 1.0};

OUTPUT out = result.color;

TEX temp, tex0, texture[0], 2D; #Fetch texture

MOV out, temp; #replace

#MUL out, col0, temp; #modulate
#MUL out, temp, pink; #modulate with constant color
```

52

Phong Lighting

```
#compute half angle vector
ADD spec.rgb, view, lVec;
DP3 spec.a, spec, spec;
RSQ spec.a, spec.a;
MUL spec.rgb, spec, spec.a;

#compute specular intensity
DP3_SAT spec.a, spec, tmp;
LG2 spec.a, spec.a;
MUL spec.a, spec.a, const.w;
EX2 spec.a, spec.a;

#compute diffuse illum
DP3_SAT dif, tmp, lVec;
ADD_SAT dif.rgb, dif, const;
```

53

Procedural Bricks

```
#Apply the stagger
MUL r2.w, r0.y, half;
FRC r2.w, r2.w;
SGE r2.w, half, r2.w;
MAD r0.x, r2.w, half, r0.x;

#determine whether it is brick or mortar
FRC r0.xy, r0;
SGE r2.xy, freq, r0;
SUB r3.xy, 1.0, freq;
SGE r0.xy, r3, r0;
SUB r0.xy, r2, r0;
MUL r0.w, r0.x, r0.y;
```

54

Wrap up

- Fragment Program is an OpenGL standard for Fragment Programmability
- Assembly Language Shaders
- Enhances Pixel Level Effects

55