The Programmable Graphics Hardware Pipeline

Doug James
Asst. Professor
CS & Robotics

Overview: The Programmable Graphics Hardware Pipeline

- What is it?
- Why do we want it?
- Applications
- Recent advances
- How do we use it?
- Quick tutorial
- Current research!

GPU Programming Model

CPU
- Application
- Vertex Processor

GPU
- Application
- Vertex Processor
- Assembly & Rasterization
- Fragment Processor
- Framebuffer
- Textures

How are current GPU’s different from CPU?

1. GPU is a stream processor
   - Multiple programmable processing units
   - Connected by data flows
How are current GPU's different from CPU?

2. Greater variation in basic capabilities
   - Most processors don’t yet support branching
   - Vertex processors don’t support texture mapping
   - Some processors support additional data types

How are current GPU's different from CPU?

3. Optimized for 4-vector arithmetic
   - Useful for graphics - colors, vectors, texcoords
   - Easy way to get high performance/cost

- Shading languages have vector data types and operations
  - e.g. Cg has float2, float3, float4
- Obvious way to get high performance
- Other matrix data types
  - e.g. Cg has float3x3, float3x4, float4x4

Why do we want it?
- Frees us from the fixed function pipeline
- Expands the range of possibilities
- Real-time cinematic shading
- Enormous research opportunity

Procedural Shading

Character Skinning

The Programmable Graphics Hardware Pipeline

NPR Rendering
- Cartoon-style shading

How do we use it?
Quick Cg Tutorial

Let's use it!

Recent advances

32-bit IEEE floating-point throughout pipeline
- Framebuffer
- Textures
- Fragment processor
- Vertex processor
- Interpolants

Hardware supports several other data types
- Fragment processor also supports:
  - 16-bit 'half' floating point
  - 12-bit fixed point
  - These may be faster than 32-bit on some HW
- Framebuffer/textures also support:
  - Large variety of fixed-point formats
  - E.g., classical 8-bit per component
  - These formats use less memory bandwidth than FP32

Vertex processor capabilities
- 4-vector FP32 operations, as in GeForce3/4
- True data-dependent control flow
- Conditional branch instruction
- Subroutine calls, up to 4 deep
- Jump table (for switch statements)
- Condition codes
- New arithmetic instructions (e.g. COS)
- User clip-plane support
Vertex processor has high resource limits
- 256 instructions per program (effectively much higher w/branching)
- 16 temporary 4-vector registers
- 256 “uniform” parameter registers
- 2 address registers (4-vector)
- 6 clip–distance outputs
- 16 per–vertex attributes (only)

Fragment processor has clean instruction set
- General and orthogonal instructions
- Much better than previous generation
- Same syntax as vertex processor:
  MUL R0, R1.xyz, R2.yxw;
- Full set of arithmetic instructions:
  RCP, RSQ, COS, EXP, ...

Fragment processor has flexible texture mapping
- Texture reads are just another instruction (TEX, TXP, or TXD)
- Allows computed texture coordinates, nested to arbitrary depth
- Allows multiple uses of a single texture unit
- Optional LOD control – specify filter extent
- Think of it as a memory–read instruction, with optional user–controlled filtering

Additional fragment processor capabilities
- Read access to window–space position
- Read/write access to fragment Z
- Built–in derivative instructions:
  - Partial derivatives w.r.t. screen–space x or y
  - Useful for anti–aliasing
- Conditional fragment–kill instruction
- FP32, FP16, and fixed–point data

Fragment processor limitations
- No branching
- But, can do a lot with condition codes
- No indexed reads from registers
- Use texture reads instead
- No memory writes

Fragment processor has high resource limits
- 1024 instructions
- 512 constants or uniform parameters
- Each constant counts as one instruction
- 16 texture units
- Reuse as many times as desired
- 8 FP32 x 4 perspective–correct inputs
- 128–bit framebuffer “color” output
  (use as 4 x FP32, 8 x FP16, etc...)
Current Research

- GPU is becoming a general purpose stream processor
- ...

The Programmable Graphics Hardware Pipeline