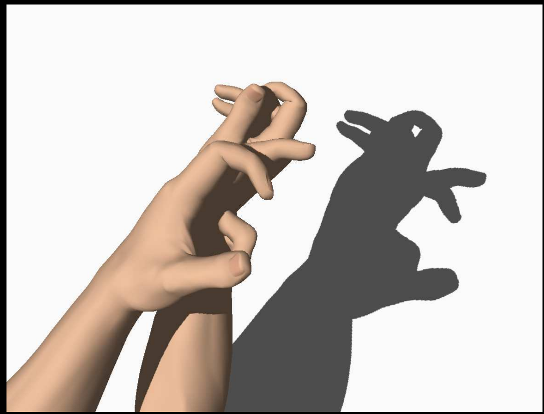


The Programmable Graphics Hardware Pipeline

Doug James
Asst. Professor
CS & Robotics

Credits

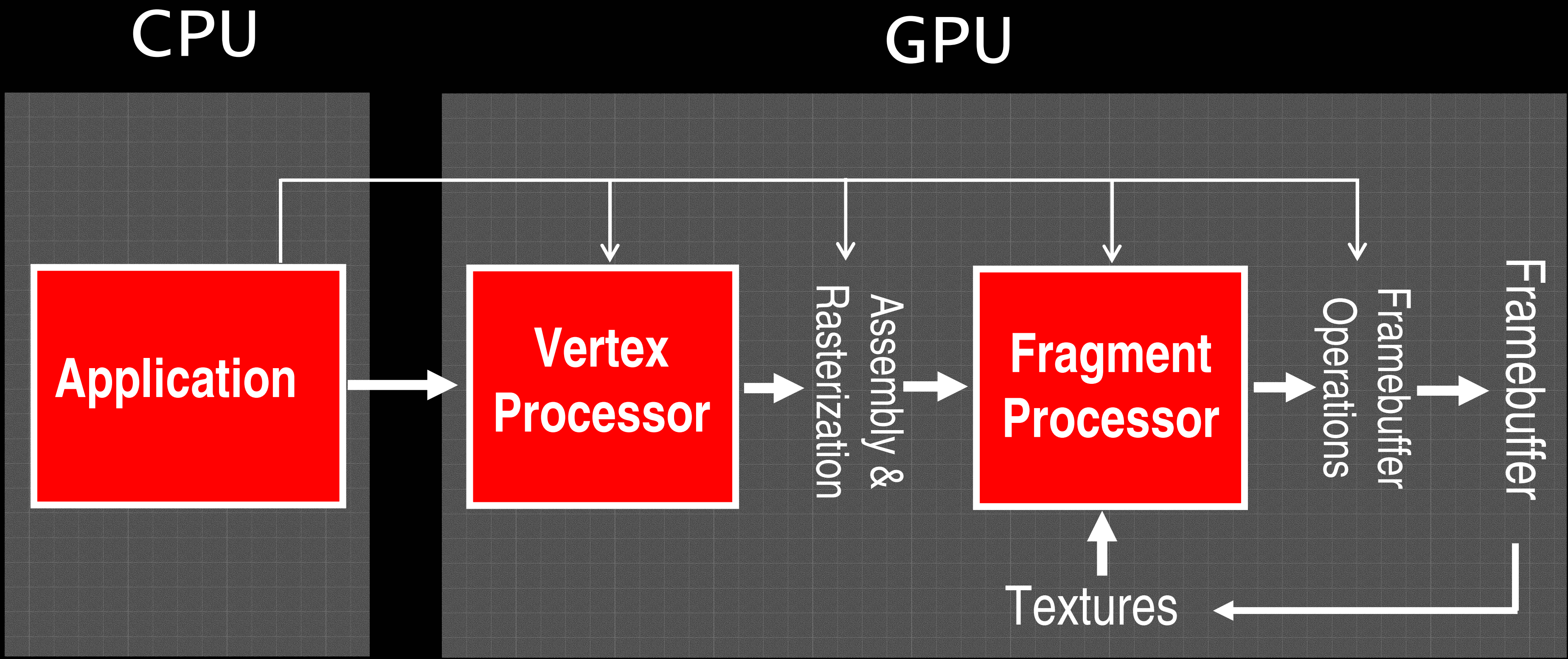
- Includes many slides from…
  - Bill Mark, "NVIDIA Programmable Graphics Technology," SIGGRAPH 2002 Course.
  - See www.nvidia.com for more.

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!

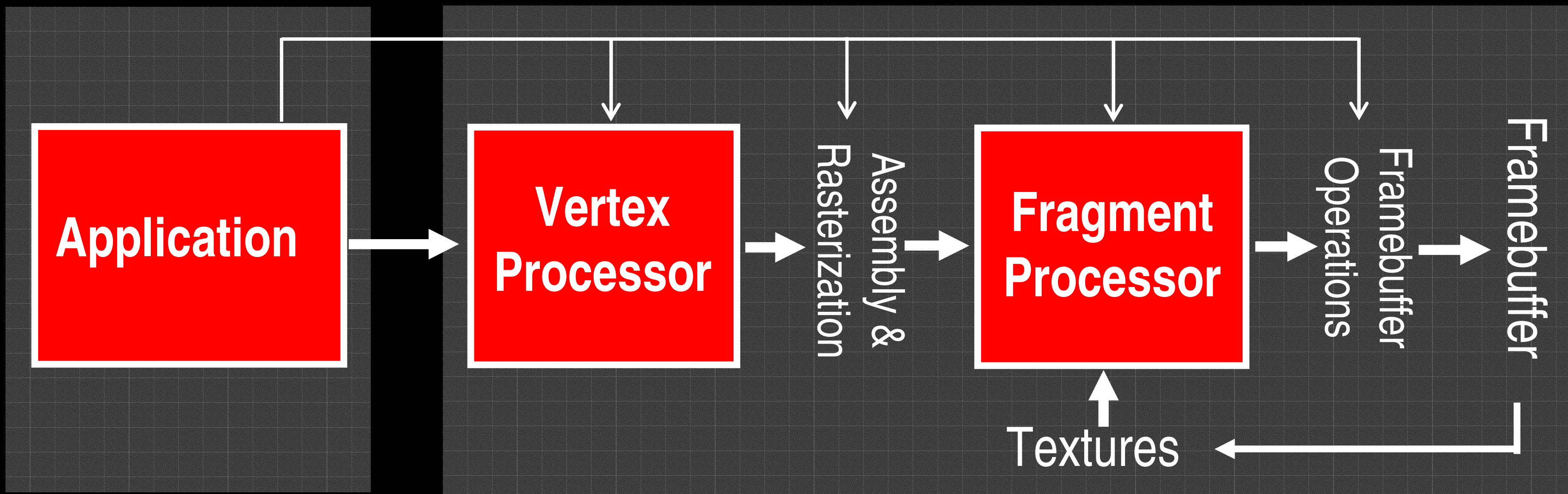
What is it?

GPU Programming Model

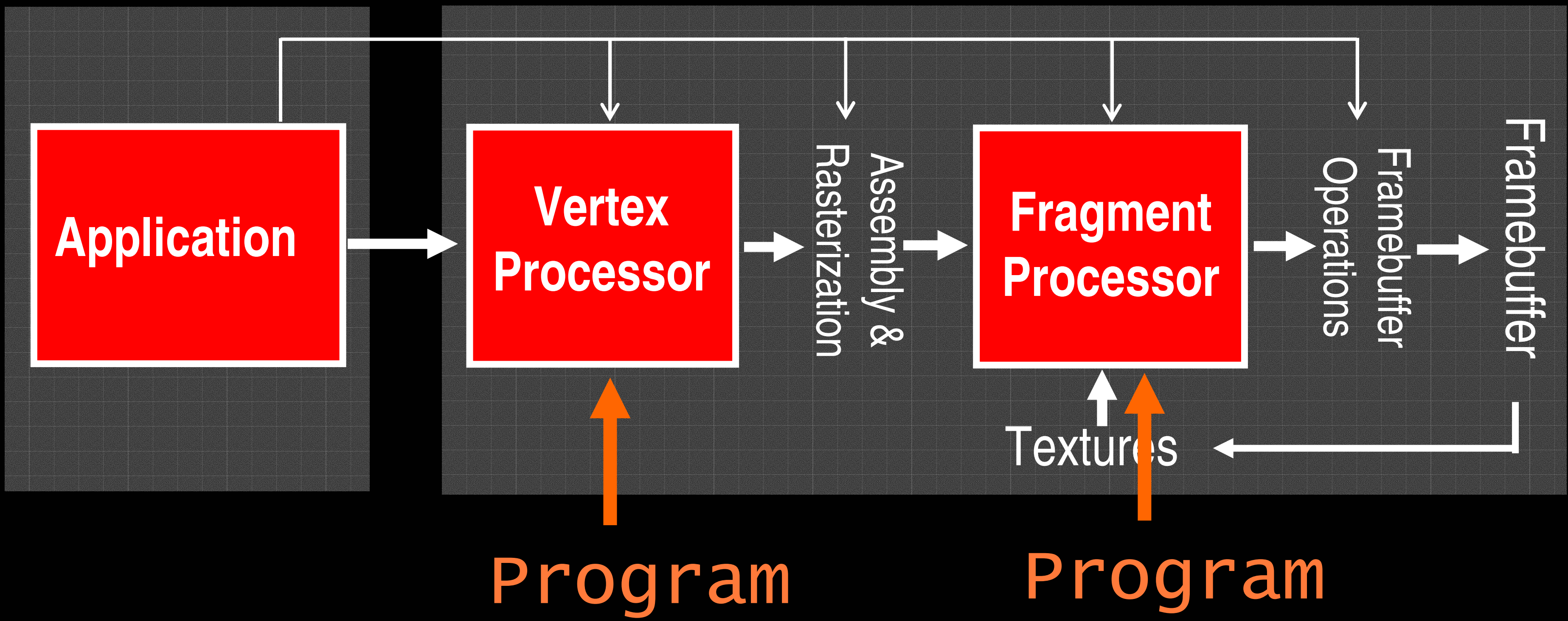


How are current GPU's different from CPU?

1. GPU is a stream processor
   - Multiple programmable processing units
   - Connected by data flows



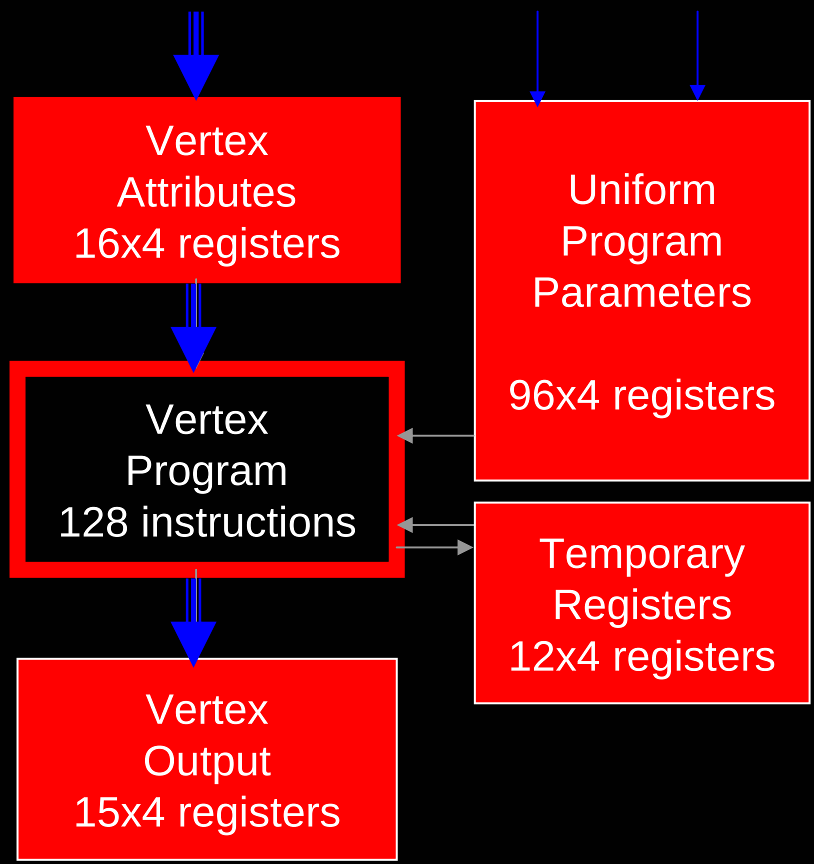
Separate vertex and fragment programs



Vertex Processor Program

- Input vertex attributes
  - Position
  - Normal
  - Bone weights
  - Colour
  - Texture
  - Other ;)
  - …

GeForce 3 Vertex Prog.



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

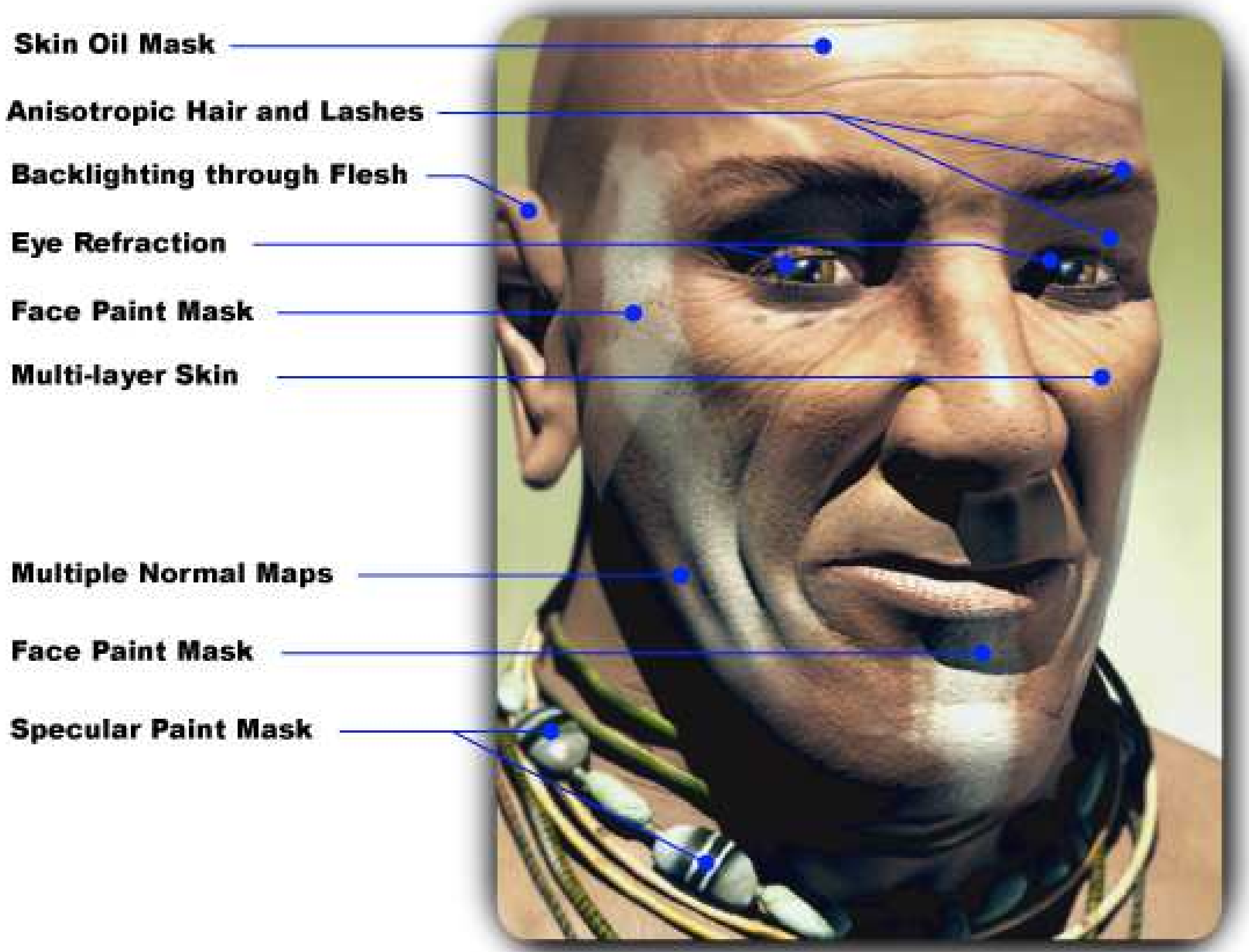
How are current GPU's different from CPU?

4. No support for pointers
   - Arrays are first-class data types in Cg
5. No integer data type
   - Cg adds "bool" data type for boolean operations
   - This change isn't obvious except when declaring vars

why do we want it?

- Frees us from the fixed function pipeline
- Expands the range of possibilities
- Real-time cinematic shading

- Enormous research opportunity


Skin Oil Mask
Anisotropic Hair and Lashes
Backlighting through Flesh
Eye Refraction
Face Paint Mask
Multi-layer Skin
Multiple Normal Maps
Face Paint Mask
Specular Paint Mask

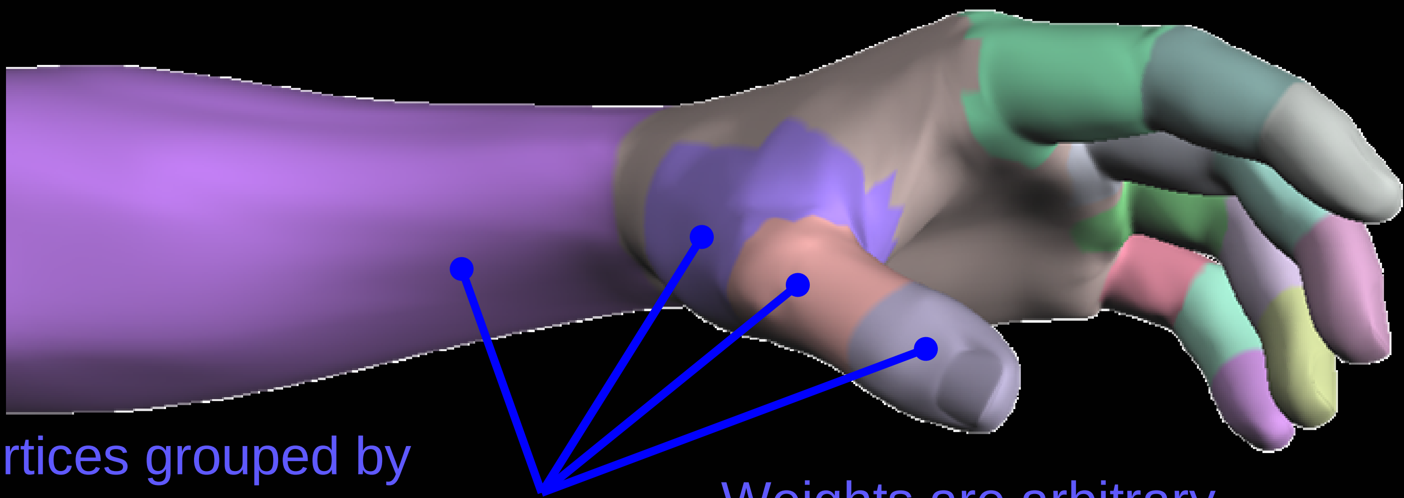
Cg

Procedural Shading

Figure 1.3. Evolution of a brick shader. a) simple version. b) with indented mortar. c) with added graininess. d) with variations in color from brick to brick. e) with color variations within each brick.

Character Skinning

$$\tilde{v}_i = \left(\sum_{b \in B_i} w_{ib} T_b \right) v_i$$



Vertices grouped by
common bone influences
Groups are arbitrary

Weights are arbitrary
- Defined by an artist
- Function of vertex-bone distances

NPR Rendering

- Cartoon-style shading

How do we use it?

Quick Cg Tutorial

The Programmable Graphics Hardware Pipeline

Let’s use it!

Quick Cg Lab
(Diffuse + Specular Shader)

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

Current Research

- GPU is becoming a general purpose stream processor
- …