LSU EE 7700-1 **Homework 7** Solution **Due: 15 April 2009**

Read "NVIDIA GeForce 8800 GPU Architecture Overview," linked to the course references page, and answer the questions below.

Problem 1: The 8800 implements a *unified shader model*, meaning it uses the same processor for both vertex and fragment (pixel) processing. What is the advantage of doing so. *Hint: This is really easy and can be answered directly from the white paper.*

The GPU area needs to be divided between vertex and fragment shaders (and other items). If there are too many vertex processors then some vertex processors will go unused, the GPU would have been faster if the chip area used for vertex processors was instead used for fragment processors. The situation is similar if there were too many fragment processors. The problem is that it's the application that determines how many is too many, so for a given GPU there may be too many vertex processors running application A and too many fragment processors running application B. There is no easy way to choose.

With a unified shader model the same processor can be used for both vertex and fragment shading, so there's no problem.

Problem 2: Describe at least two possible disadvantages of the unified shader model. These disadvantages might have outweighed the advantages in previous generation GPUs, which had separate (not unified) programmable shaders. *Hint: This question is harder and requires some understanding of computer architecture.*

Vertex and fragment shaders typically perform different tasks and so might have different computational requirements. This is especially true for older rendering pipeline models in which a vertex shader would not have access to textures and a fragment shader would not need to perform lighting calculations. A vertex processor under that model could avoid a costly texture unit (typically including specialized address calculation hardware, the cache itself, and texel filtering hardware). A fragment processor might omit instructions or hardware for computing lighting.

Under the unified model a processor would need hardware for both. That's less of an issue in modern systems because many graphics effects require texture access in the vertex processor anyway.

A second disadvantage of using a single processor type is that it precludes a simple path for data from GPU input to vertex processor inputs, vertex processor inputs to rasterizer inputs, rasterizer outputs to fragment processors. Instead, with a single processor type there must be some kind of a path from each processor output to each processor input. The GPU Architecture white paper on page 36 implies some kind of a fast connection from stream processor outputs to inputs. (CUDA applications are forced to go through uncached global memory.)

Problem 3: The white paper describes texture filter and texture address units. What they do is straightforward, but how are they used? Are they: (1) programmable units (like the stream processors) with a (possibly secret) instruction set of their own, (2) functional units (like floating-point ALUs) which are used using stream-processor instructions, (3) state machines that are operated using special control registers (like DMA controllers), (4) just subroutines that run on stream processors using ordinary stream processor instructions (like system calls or library functions), or (5) something else.

Your answer should indicate what you are basing your conclusion on (page number, etc) and how sure you are it is correct. The answer may range from an educated guess to a precise answer based on a source. Feel free to find sources other than the NVIDIA white paper.

First lets rule out some possibilities. Possibility (4), just subroutines, is unlikely because then there would be no way to overlap texture fetch and filtering with other shader code. The white paper specifically mentions this overlap.

One place in which more information can be found on the low-level hardware is the PTX instruction set, an intermediate assembly-like language described by NVIDIA as close enough to machine language for optimization purposes (see the course references page). An enterprising individual has written a disassembler for NVIDIA GE 80-family GPUs, called

Decuda. Both PTX and Decuda show texture access instructions with texture unit and coordinate source operands that write the filtered texel to the destination.

These instructions rule out (3) since texture operations are initiated by a specific instruction, not by reads and writes to some set of control registers. (Control registers might be part of a special register set or might be mapped into an address space and accessed using load and store instructions. Either way, they aren't used for textures.)

This leaves (1) and (2). In (1) the texture instruction would trigger code on the programmable processor, in (2) it would initiate the operation in non-programmable hardware. In both cases the texture unit would have to signal completion to the stream processor controller.

There are many variations on texture access (such as linear or nearest-texel filtering) but no way to specify them in the texture instruction. One possibility is that the driver inserts code into a programmable texture processor (1) based on the texture filtering specified or for (2) the driver might simply specify the type of filtering and other options via control registers writable only from the command processor. (The control registers would only be used for setup, not for initiating a texture operation.)

To achieve a high throughput the texture processor would need specialized hardware to compute mipmap levels and filter texels, with such hardware in place control would probably be simple so a programmable unit would be overkill. Therefore (2) is more likely. An advantage of (1) though is greater flexibility in filtering using filtering techniques not considered when the hardware was designed.