LSU EE 7700-1 Homework 3 Solution Due: 4 March 2009

Answer the following questions about Lindohom 2001, which describes the NVIDIA GForce 3.

Problem 1: Section 3.7.1 justifies the decision to exclude branch instructions by pointing out that the OpenGL API was intentionally designed to avoid the need for branching up until clipping. (That is, steps such as lighting and transform would be branch free.)

The OpenGL code below contains some calls which are not allowed between a begin/end pair. (They would result in an *invalid operation* error.)

```
glBegin(GL_TRIANGLES);

while ( Group* const group = group_list )
  {
    glEnable(GL_RESCALE_NORMAL);                         // DISALLOWED (a)
    glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 0.1);   // DISALLOWED (b)
    glColor3f(1,.1,.1);
    glNormal3fv(group->p->surface_normal);
    glVertex4fv(group->p->pos);

    glDisable(GL_RESCALE_NORMAL);                       // DISALLOWED (a)
    glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 0.2);  // DISALLOWED (b)
    glNormal3fv(group->q->surface_normal);
    // Color left out intentionally.
    glVertex4fv(group->q->pos);

    glNormal3fv(group->r->surface_normal);
    glColor3f(0.1,0.1,1);
    glVertex4fv(group->r->pos);
  }

glEnd();
```

(*a*) Identify the calls which are not allowed because they might require branching. Explain why branching would probably be needed for these calls.

In the code above calls disallowed because of branching are followed by the comment `DISALLOWED (a)`.

These disallowed calls turn normal re-scaling (normalization) on and off. If the vertex processor instruction set included branches then the `GL_RESCALE_NORMAL` option could be implemented by having a branch instruction simply jump over some rescaling code when `GL_RESCALE_NORMAL` was false (the `GL_RESCALE_NORMAL` state would be sent to the vertex processor in the same way as attributes).

(*b*) Identify the call which is not allowed for another reason. That reason is given on the same page, 151. Explain why allowing the call would complicate the vertex processor design.

Calls disallowed for another reason are followed by the comment `DISALLOWED (b)`; these calls change a lighting parameter. If they were not disallowed then that lighting parameter would have to be sent as an attribute. Section 3.5 describes the 16 existing input attributes, and from Section 4.2 one can see that adding additional input attributes would impact the size of the VAB plus the n input buffers. Though it might be possible to squeeze in just one more attribute, the `GL_CONSTANT_ATTENUATION` attribute is one of many lighting attributes so it would not make sense to add that one without adding others. So to keep costs down lighting parameters are not allowed as attributes.

Problem 2: Let r denote the number of FP operations a processor initiates per cycle when running some program. Let $r_{\max}$ denote the maximum possible r for a processor, one attained with just the right program. Define FP efficiency of the processor on a program to be $r/r_{\max}$.

Estimate the efficiency of the vertex processor of the GF 3, as described in Lindholm 2001, on the program given in Section 6.2 of that paper. Note that a vector instruction such as `ADD R1, R2, R3` initiates four FP operations, while `ADD R1.x, R2.x, R3.x` initiates just one.

To determine $r_{\max}$ we need to find the maximum instruction execution rate (number of instructions per cycle) and the maximum number of floating-point operations per instruction. The end of Section 4.3 implies a maximum execution rate of 1 instruction per cycle (IPC). The number of FP operations varies by instruction. An `ADD` performs four operations (one for each vector component), `DST` performs five (see page 152), and `DP4` performs seven, and `MAD` performs eight. (The description of the `LIT` instruction in the paper is misleading: the dot products are not performed by `LIT`. See the NV VERTEX PROGRAM extension specification.) Assuming the system can initiate one `MAD` every cycle, $r_{\max} = 8$.

To compute r we divide the total number of FP operations performed by the program by the number of instructions. The program is shown below, along with the number of FP operations used by each instruction. An instruction like `MUL` is capable of using 4 FP operations, but in some cases fewer are used, for example, the last `MUL` only operates on one of the vector elements to it performs just one FP operation.

For some instructions the number of FP operations is an estimate and one might quibble over whether it's fair to say that `RSQ` performs one FP operation ($x^{-\frac{1}{2}}$) because exponentiation is harder than addition. Any adjustment for these issues should not detract from the point of this assignment, FP efficiency.

The program below consists of 33 instructions and performs a total of 126 FP operations and so $r = \frac{126}{33} = 3.82$. The overall efficiency is then $\frac{r}{r_{\max}} = \frac{126}{33}\frac{1}{8} = 0.48$, meaning on average less than half the FP units are being used.

The NVIDIA 8000-series GPUs (GeForce 80) avoids this particular inefficiency by not using a vector data type.

```
DP4 o[HPOS].x, c[0], v[OPOS];          7
DP4 o[HPOS].y, c[1], v[OPOS];          7
DP4 o[HPOS].z, c[2], v[OPOS];          7
DP4 o[HPOS].w, c[3], v[OPOS];          7
DP4 R0.x, v[OPOS], c[4] ;              7
DP4 R0.y, v[OPOS], c[5] ;              7
DP4 R0.z, v[OPOS], c[6] ;              7
DP3 R0.w, R0, R0 ;                     5
RSQ R0.w, R0.w ;                       1
MUL R0, R0, R0.w ;                     4
DP3 R1.x, v[NRML], c[8] ;              5
DP3 R1.y, v[NRML], c[9] ;              5
DP3 R1.z, v[NRML], c[10] ;             5
DP3 R2.x, v[TEX0], c[4] ;              5
DP3 R2.y, v[TEX0], c[5] ;              5
DP3 R2.z, v[TEX0], c[6] ;              5
DP3 R3.x, c[30], R2;                   5
MAD R3.y, R3.x, R3.x, -c[24].w ;       2  (Only 1 element used.)
RSQ R3.z, -R3.y ;                      1
MUL R3.y, -R3.y, R3.z ;                1  (Only 1 element used.)
DP3 R3.w, c[30], R1 ;                  5
SGE R3.z, R3.w, c[24].y ;              1
MUL R4.x, R3.z, R3.y ;                 1
DP3 R5.x, R0, R2 ;                     5
MAD R5.y, R5.x, R5.x, -c[24].w ;       2
RSQ R5.z, -R5.y ;                      1
MUL R5.y, -R5.y, R5.z ;                1
MUL R5.w, R3.x, R5.x ;                 1
```

```
MAD R4.y, R3.y, R5.y, R5.w ;           2
MOV R4.w, c[24].x ;                    0
LIT R6, R4 ;                           3
MAD R7, R6.y, c[40], R6.z ;            2
ADD o[COL0], R7, c[41] ;               4
========================================
33 insn                              126 fp ops
```