RSP Graphics Microcode Architecture

This document describes the RSP graphics microcode in detail. Together with the commented source code, this document outlines the microcode design and illuminates major implementation details.

A thorough understanding of the RCP architecture is assumed; prerequisite skills include:

- Mastery of the Nintendo64 as an application programmer (creating graphics display lists, programming the RDP, etc.).
- Complete understanding of the material in the *NU64 Programming Manual.* Understanding the graphics microcode engine is impossible without understanding the top-level of the system.
- Complete understanding of the material in the *Nintendo Ultra64 RSP Programmer's Guide.* Reading, understanding, modifying, or creating graphics microcode requires an understanding of the RSP architecture, the RSP instruction set, and the RSP programming environment.
- Complete understanding of the material in the *RCP Graphics Binary Interface.* This document, along with the *RDP Command Summary,* describe the details of the display list format and rasterization hardware interface.

Additional useful background information:

- A thorough background in 3D graphics.
- The *OpenGL Programming Guide.*

- Published papers on previous SGI graphics architectures[1]. These papers provide an excellent overview of the major issues associated with designing high-performance graphics hardware (and systems).

Note: This document describes the microcode as of release 2.0D. Since the microcode is always being tweaked and improved, this document may grow out of date. Final word on "what the microcode does and how it works" is contained in the current source code.

[1] Clark, J. H., "The Geometry Engine: A VLSI Geometry System for Graphics", Proceedings of SIGGRAPH '82, in *Computer Graphics*, Vol. 16, No. 3, pp. 127-133, ACM SIGGRAPH, New York.

Akeley, K., Jermoluk, T., "High-Performance Polygon Rendering", Proceedings of SIGGRAPH '88, in *Computer Graphics*, Vol. 22, No. 4, pp. 239-246, ACM SIGGRAPH, New York.

Akeley, K., "Reality Engine Graphics", Proceedings of SIGGRAPH '93, ACM SIGGRAPH, New York.

Major Features of the RCP

For graphics processing, the RSP is used as a programmable geometry engine. The RSP microcode implements display list decoding, vertex transformations, shading, clipping, etc. Output of the RSP graphics microcode is rasterization commands to the RDP (back-end).

The RSP is an intermediate stage in the graphics pipeline that is resource constrained.

Before focusing on the graphics microcode, it is useful to review the global RCP graphics features:

- several thousand polygons per frame. (> 100K polys/second)
- mip-mapped (anti-aliased) textures.
- antialiasing.
- z-buffering.
- adequate lighting/shading support.
- adequate support for “effects”.
- adequate support for traditional games (2D).

For the game programmer, the graphics pipeline is exposed through the *Graphics Binary Interface*, or GBI, which is defined in the file `gbi.h`.

The major function of the RSP graphics microcode is to implement portions[1] of the Graphics Binary Interface (GBI). The microcode parses the display list and implements a state machine which processes the geometry for final rendering.

[1] We say “portions” here because many of the GBI commands are actually direct hardware instructions destined to be interpreted by the RDP. These simply “pass through” the graphics microcode.

Major Design Goals of the Microcode

The major design goals of the microcode can be stated succinctly:

- Core processing (geometry pipeline) should run at maximum speed.
- Compact code (must fit in IMEM, 1K instructions).
- Implement a robust and useful display list processing machine.
- Support the hardware featurs of the RCP.

The first goal is to provide the highest performance possible. Although in a narrow sense this means just writing good, efficient code, we must also consider the “big picture”; the place the RSP has in the total pipeline. Maximum throughput will come from a balanced system. If the RSP and RDP are not balanced, then we can afford to do more processing somewhere, or we need to do less processing elsewhere. For the microcode, this might mean feature reductions, or permit feature enhancements.

Compact code is a strict requirement. The size of the RSP instruction memory (IMEM) is fixed. While code overlays are possible, the performance penalty of swapping microcode may limit their application.

The RSP graphics microcode implements the display list machine presented to the application programmer. This display list language must be useful, with a sufficiently rich set of commands to do high-quality, real-time animation. Besides the traditional geometry processing, special real-time constructs are desired (level of detail culling, bounding volumes, etc.) It must be efficient, both for the application and the microcode (although we always favor the microcode if we have a choice).

The RCP excels in generating very high-quality images. The graphics microcode needs to support these features. In practical terms, this can mean performing calculations using increased precision, or using algorithms not normally associated with “low-end” graphics systems.

Architecture Overview

Graphics microcode exists in several variations, *gspFast3D, gspLine2D, gspF3DNoN, gspTurbo3D,* etc. All variations except *gspTurbo3D* are very similar and will be discussed in the same context. The turbo microcode implements a completely different display list engine and is not discussed in this document.

Each microcode version also has several variations for RDP output, *.dram, .fifo,* etc. The output code is shared among all microcode and will be discussed only once.

There are only 1K lines of microcode, and the display list feature set is relatively small, so what are the interesting parts of the architecture? The display list engine implemented by the microcode must be compact, efficient, and complete; the design must also be extensible and elegant. Software elegance promotes solid, bug-free code; it also encourages a clean, understandable system.

Most microcode objects are compiled in a single pass; program modularity is enforced by using `#include` to separate source code files. Code which can be shared is abstracted as best as possible, so that register conflicts and calling sequences can be as clean and efficient as possible.

From a top-level view, the microcode could be divided into two layers: the display list processing engine (described in Figure 0-1), and the implementation of each display list command (described in Figure 0-2).

Display list commands are further divided into three types: *RDP Commands, Immediate Commands, and DMA Commands.* In all cases, commands are grouped because of their similar processing requirements, in fact, in most cases command parameters are parsed opaquely, sharing this code among many commands.

RDP Commands are direct RDP hardware commands, they are usually passed through the RSP without modification[1].

Immediate Commands are perhaps poorly named; they are not *immediate mode* commands, in the sense of a traditional display list graphics architecture. This

[1] The main modification of RDP commands is address translation, turning segment-offset addresses into the physical addresses required by the RDP. There is also some state processing.

grouping of commands are display list commands that can be entirely contained in one 64-bit display list word.

DMA Commands are those which require data to be DMA’d into DMEM before processing.

Figure 0-1 Top-Level Display List Processing

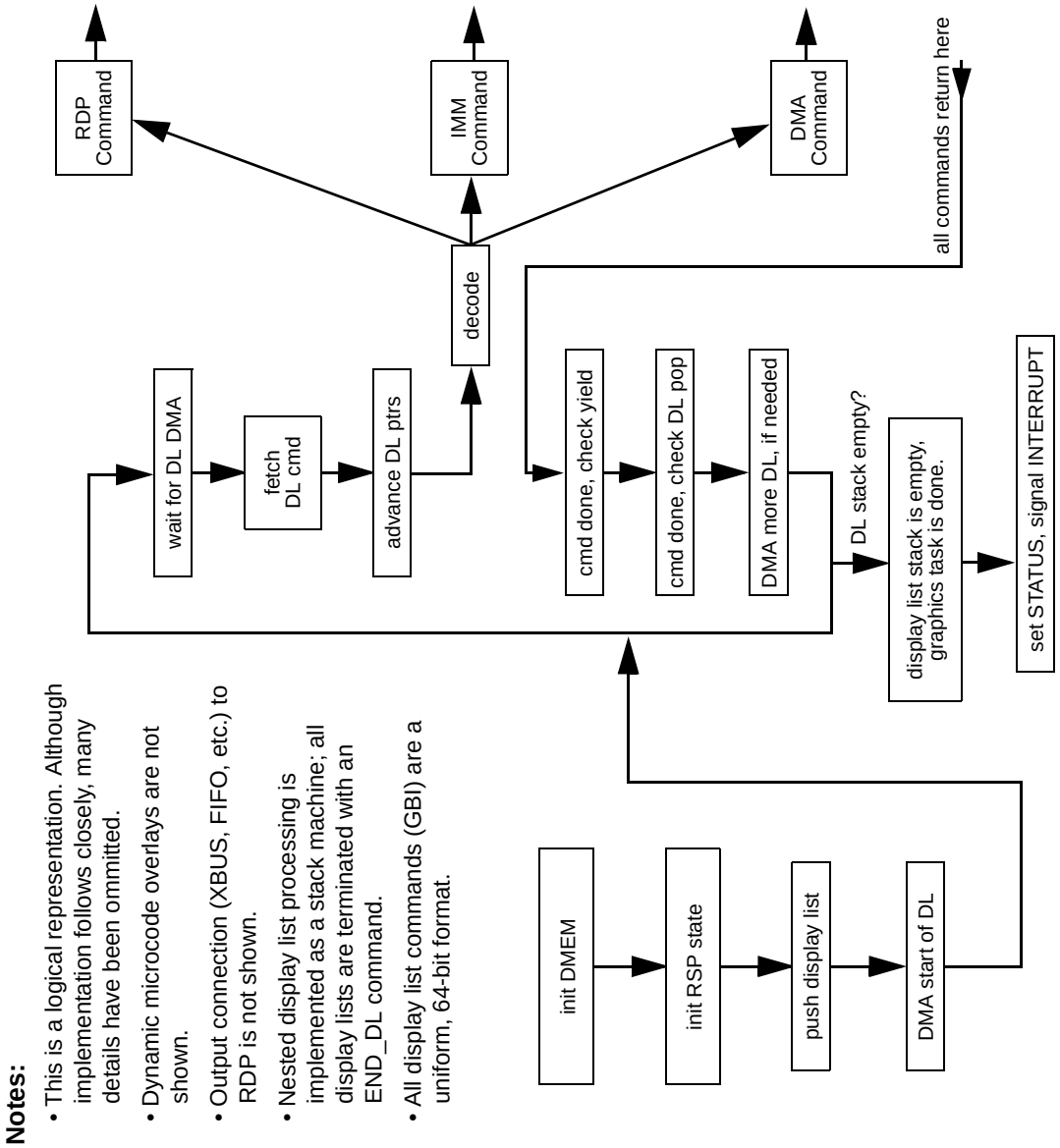
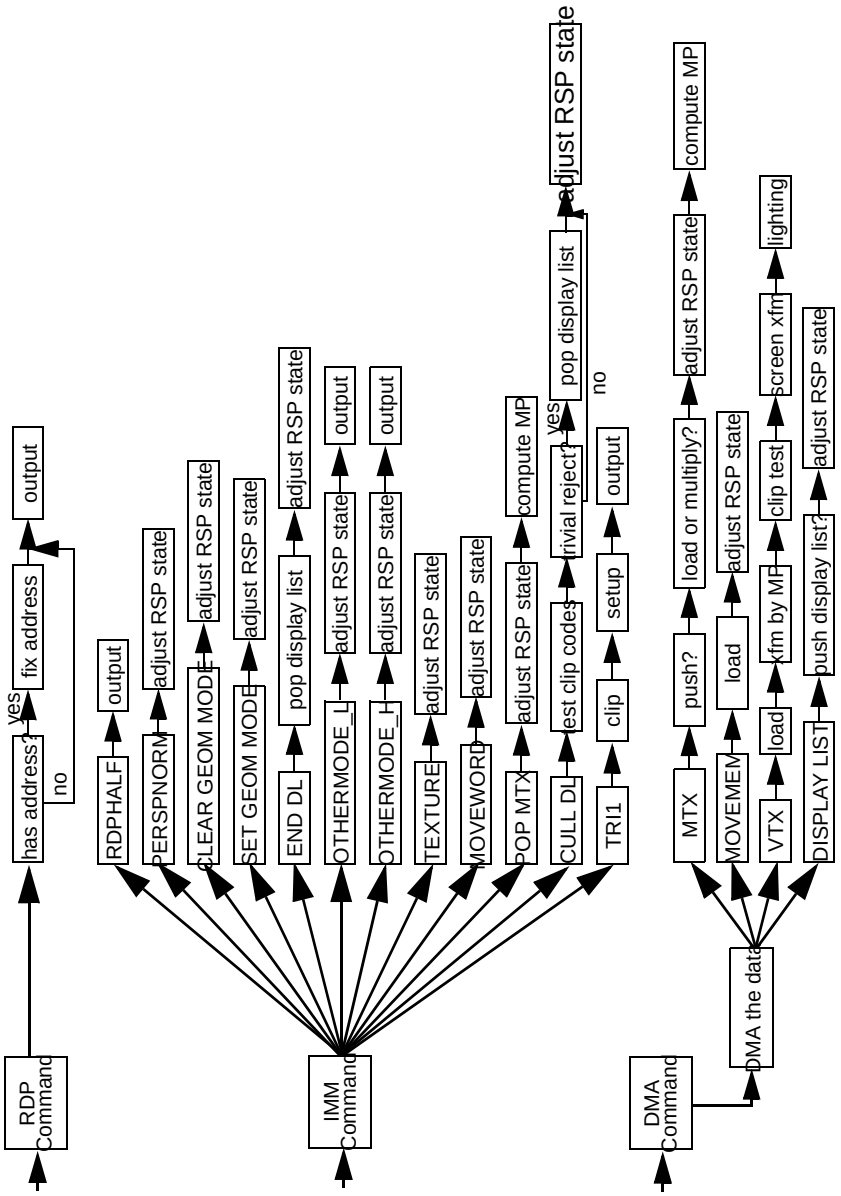


Figure 0-2 Display List Command Processing



Register Usage

Global Register Assignment

Global registers are assigned beginning at register `$31` (and `$v31`), growing downward. The most stable registers should remain near the top.

Transient registers are assigned beginning at register `$1` (and `$v0`), growing upward. Sections of code should use and release registers as necessary.

Parameters to functions are passed in the lower registers.

If a section of code finds it necessary to break these rules, it is responsible for saving and restoring any registers' contents that it disturbs. For efficiency, some often-used subroutines define other registers to use; these registers are coordinated with all calling routines.

The following global registers are defined in `gfx_regs.h`, and are available to any module:

Scalar Registers

`return, $31` This register is the MIPS convention for `JAL` (jump and link) opcodes. We use it in other places to hold a return address for a function.

`return_save, $30` An additional save register for return addresses.

`rsp_state, $29` A pointer to the RSP graphics state. This state structure (defined in `gdmem.h`) holds matrix stack information, shading state, RDP state, etc.

`dlcount, $28` The current counter of the display list buffer (in bytes). When this counter decrements to 0, we need to load in more display list.

`dinp, $27` The current pointer into the display list buffer. Points to the next display list command to be executed.

`inp, $26` This is the DRAM pointer of the current location in the display list. It shadows `dinp`, so that we can push the current display list location onto a stack for nested display lists. When dlcount reaches 0, `inp` also tells us where to find more display list commands.

`gfx0, $25` The first word of the current display list command.

`gfx1, $24` The second word of the current display list command.

`outp, $23` The current output pointer.

`in_bufp, $22` This is the pointer to a (DMA) display list command's input data.

`zero, $0` The MIPS convention that register `$0` always holds a 0 in it. You cannot change this register, so if you attempt a write it will behave like `/dev/null`. This register is also used as a base address to reference DMEM. Since DMEM is only 4K, we can reference any address using the 16-bit offset field of the load and store instructions.

Vector Registers

`vconst, $v31` This is a vector of constants used in the graphics code. This vector is initialized from the data defined in `gdmem.h`. This register holds the following constants: `0, 1, 2, -1, 0.25, 4.0, 1587,` and `1/512` (all fractions are in 16-bit fixed point fraction format).

`vconst1, $v30` This is a vector of constants used in the graphics code. This vector is initialized from the data defined in `gdmem.h`. This register holds the following constants: `0x7fff, 0xfff8, 8, 0x0040, 0x0020, 0x8000, 0x1ccc,` and `0xcccc`.

Register Naming

All registers used in the graphics microcode should be named. This makes it easier to track register-allocation bugs and modify code safely.

Two schemes are used: upon entrance to a procedure or block of code, a group of `.name` directives for that code occur. Following the code, all of these registers are un-named using the `.unname` directive.

More complicated pieces of code require a tighter “scope maintenance” of registers. In this case, named registers are “born” and “die” when needed, allowing us to re-use registers. Actual register allocation tries to use lower-numbered registers for names with a longer “life”, making it a little easier to track bugs and modify code.

Functions which are called from several places within the code use ranges of registers which have been arbitrated with the calling functions.

DMEM Usage

The microcode uses DMEM as an explicitly managed cache. Regions containing data structures are allocated at compile time and referenced with assembler-calculated offsets. Dynamic areas are also reserved at compile time; at run time they are filled with the appropriate data.

All DMEM reservations are located in the source file `gdmem.h`. Laying out the DMEM map in one place makes things more clear.

DMEM Map

Table 0-1 presents the DMEM organization.

Table 0-1 DMEM Map

Usage	Size
Program Data From Compiler (constants, jump tables, etc.)	272 bytes
RSP Graphics State	80 bytes
Addressing Segment Table (16 entries x 4 bytes)	64 bytes
Lights (8 x 48 bytes)	384 bytes
Viewport	16 bytes
Fog Factors	6 bytes
Display List Stack (10 deep stack, 4 bytes per pointer)	40 bytes
Modelview Matrix, top of stack (4x4 x 4 bytes)	64 bytes
Projection Matrix, top of stack (4x4 x 4 bytes)	64 bytes
MP Matrix (Modelview x Projection) (4x4 x 4 bytes)	64 bytes

Table 0-1 DMEM Map

Usage	Size
Points Buffer (16 points x 40 bytes)	640 bytes
Display List Input (40 commands x 8 bytes)	320 bytes
Input (Data) (big enough to hold 16 incoming points, etc.)	256 bytes
Rasterization Setup temporary area	96 bytes
Clipping temporary area	160 bytes
Output to RDP (XBUS) (at least 6 full-featured triangles x 160 bytes)	1024 bytes
Scratch space. Holds clipped vertices during clip and setup.	480 bytes

Program data This area holds any static data initialized at compile time, such as constants, jump tables (or other contants who's value is determined by an assembler symbol), overlay information, etc.

Graphics state This area holds a structure which remembers many state variables; matrix stack information, display list stack information, number of lights, RDP mode words, etc.

Segment Table These 16 segment base addresses are used to translate display list addresses into physical addresses. Entry 0 should hold `0x0` by convention, allowing direct physical addresses to be translated.

Lights This buffer holds up to 8 light structures.

Viewport This buffer holds the viewport information.

Fog Factors Some fog information.

Display list stack Stack of nested display lists, pointers to DRAM.

Modelview matrix The top of the modelview matrix is kept in DMEM here.

Projection matrix The top of the projection matrix is kept in DMEM here.

MP matrix	The concatenation of the modelview matrix and the projection matrix is kept in DMEM here.
Points buffer	This buffer holds up to 16 vertices. The structure of this buffer is not the same as incoming vertices, this structure holds 3D clipping points, screen points, and other information. During clipping, additional points may be generated. These are stored in the scratch area.
Display list input	A buffer holding up to 40 display list commands.
Data input	A buffer holding data from DMA commands. This buffer needs to be large enough to hold the largest possible display list structure, 16 vertices.
Setup temp	This holds data needed for rasterization setup. This must be different from the scratch space below, since the clipping uses the scratch space (and clipping calls setup).
Clipping temp	This area holds temporary data used during clipping. Newly-generated points are stored below in the scratch area.
RDP output	This buffer is used to transfer output to the RDP. For XBUS transfer, a ring-buffer organization is used, and the RDP consumes commands directly from this buffer.
Scratch space	This area may be used during processing of any GBI command. Contents of this area are not guaranteed to remain between commands.

Static Initialized Program Data Segment

For the graphics task, we use the static initialized data segment to hold program constants and program jump tables, as well as initialize some of our logical structures. Initializing constants in this way saves code space, and allows us to load up a vector of constants with one instruction. The use of jump tables allow us to quickly compute offsets from the GBI opcodes, then load the address of the procedure for the necessary opcode. This is fast, and also compact.

We place the program data segment at the beginning of DMEM in order to save some steps when computing program data offsets.

Notice also, that the data segment must be loaded by the graphics task, to the location the source code expects it. The task header (`OSTask`) contains a pointer to this data, and it's length; the application will provide it to us. This loading of the DMEM is also used to initalize several important data structures.

DRAM Stack

The graphics code also needs some DRAM space that it can control. It uses this to maintain the matrix stacks (or other longer-term storage purposes).

Graphics Code Which Vectorizes

Certainly all of the graphics microcode deserves attention to make sure that it fully exploits the vector unit, vector pipelining, and parallel execution of the SU and VU. However, in accordance with Amdahl's law ("make the common case fast"), we want to analyze in detail the geometry pipeline; which, when measured by clock cycle, instruction count, or sheer "pixel excitement", will be the most important part of the graphics code.

We also want to favor certain methods of processing the geometry, i.e., an application which sends down 3 points, a single triangle, 3 points, a single triangle, etc., should expect not to run at full speed.

The geometry pipeline includes the following steps:

- matrix operations
- vertex transformations (modeling space to screen space)
- vertex lighting (including normal and texture transformation)
- triangle clipping
- triangle edge and attribute setup for rasterization.

Matrix Operations

By definition, most of the matrix operations vectorize well. A 4x4 matrix multiply (matrix concatenation) requires just 16 vector multiplies (and some adds to accumulate the terms).

Multiplying a point by a (4x4) matrix is just 4 vector multiplies (which pipeline quite nicely), taking about 12 clock cycles. We keep the matrix doubled in the registers and operate on two points at a time.

Manipulation of 4x4 matrices does not take full advantage of the 8-element vector unit. While this is unfortunate, improving this would require certainly more vectorization setup work and probably even extensions to the VU instruction set to allow us to keep multiple rows/columns in the vector registers (and select vector halves as destination registers, yuk!).

For lighting calculations, we need the inverse transpose of the transformation matrix, in order to transform the vertex normal into the proper space for lighting. We do not do that, we light in “world space”, and can calculate the inverse transpose directly.

The actual 4x4 matrix concatenation routine is abstracted into separate code which operates on matrices stored in DMEM. It has the restriction that the destination matrix cannot be one of the source matrices, which simplifies the code. Multiplying matrices directly from DMEM also uses fewer instructions and fewer registers, both precious resources. Since most of the loads/stores can be pipelined, the only downside to this approach is some scalar code to do the looping, which is acceptable.

Vertex Transforms

The vertex transformation step includes:

- multiplying the incoming point by the current `MP` matrix.
- performing the clip testing.
- doing the *w*-divide (projection).
- image space scale and translate (viewport mapping).
- transforming texture coordinates

Each of these steps happens “per-vertex”, so we can loop over the incoming vertices. Each of these steps also vectorizes well but they do not pipeline together very well. The key to fully utilizing the vector unit is to process two points per vector register. In order to improve pipelining, we could un-roll the loop once, operating on four points at a time (but we don’t, since we don’t have the IMEM space. Instead, we interleave other computations and accept some pipeline penalty).

Current vertex processing (with no lighting) averages about 35 clocks per vertex.

Vertex Lighting

Lighting vectorization is similar to vertex transforms; we light two points at a time in parallel, vectorizing across the *r,g,b,a* components.

Lighting calculations also include per-vertex fog (modifying the alpha value) and texture coordinate generation (for specular, chrome, and environment effects).

Another Idea for Lighting

> **Note:** This section describes a technique which was explored, but not used in the final microcode. It was decided that the visual artifacts of vertex-interpolated specular highlights made this feature not worth implementing. Nevertheless, this section remains useful as an exercise of algorithm analysis and vectorization.

The most important concern for vectorization of the lighting step is selection of the algorithm. We want to implement a lighting equation which looks something like:

$$color = (surfAmb)(ambient) + (surfDiff)(diffuse) + (surfSpec)(specular)$$

surfAmb is the product of the material's ambient coefficient and the light's ambient coefficient. *surfDiff* is the product of the material diffuse and light diffuse. *surfSpec* is the product of the material specular and light specular. These terms are precomputed when the material or light changes, and stored in DMEM.

ambient is constant for a light. *diffuse* is a Lambertian diffuse coefficient, the angle between the surface normal and the light direction (computed as a dot product). *specular* is the specular highlight distribution function (discussed more below) which uses the angle between the surface normal and the "vector of maximum specular highlight" (also computed as a dot product). For our implementation, we choose the *H* vector as the vector of maximum specular highlight. This is the "halfway" vector which bi-sects the angle between the light source and the viewer. For our model, the *H* vector is constant per light, since the viewer and lights are "infinitely" far away.

Computation of the *specular* reflectivity term of the shading equation presents some problems. A traditional Phong[1] specular reflection distribution function requires a *pow()* function which does not vectorize well at all (due to data recurrence).

Piece-wise or polynomial approximations of the *pow()* function could be substituted with better performance, but each of these solutions has their own problems.

A better solution is to choose a different shading model that gives us the effects (and parameter control) we desire, but has better computational properties for our machine. A subset of the Blinn shading model meets these goals.

The Blinn shading model[1] uses a physically-inspired specular reflection distribution function, based on Trowbridge and Reitz, which models microfacets of reflection as tiny spheroids of revolution:

$$specular = \left(\frac{k2}{k1(NdotH^2)}\right)^2$$

where

$$k1 = \frac{1}{shiny^2 - 1.0}$$

and

$$k2 = k1 + 1.0$$

[1] Phong, Bui-Tuong, "Illumination for Computer Generated Pictures", *Communications of the ACM*, 18(6), June 1975, pp. 311-317.

[1] Blinn, J.F., "Models of Light Reflection for Computer Synthesized Pictures", SIGGRAPH 77, pp. 192-198.

k1 and *k2* are computed once per material and stored. *Shiny* is the highlight control parameter, ranging from 0.0 for shiny surfaces to 1.0 for matte surfaces (in the GBI and microcode, this range is 0-127).

So at vertex shade time, the final *specular* term requires four multiplies and one reciprocal operation. At least one of these multiplies has no data dependencies and can be pipelined in among other work.

Since the shading equation must be computed for each color component (*red, green, blue, alpha*), we can vectorize this operation nicely. Since the VU has 8 vector elements, we can fit two vertices in the registers and shade two vertices in parallel with minimum additional vectorization setup costs, and maximum utilization of our vector unit. As in the vertex code, we could un-roll the loop once and process a total of four vertices at a time.

Triangle Clipping

Clipping is a case where we have made different trade-offs of code space vs. performance. Our strategy is to avoid clipping as much as possible, and when necessary we use a simple clip routine optimized for code space rather than performance.

We avoid clipping in several ways: (1) we assume the application (game) is probably doing some coarse culling for it's own purposes, and therefore the data being sent to the RSP has an increased probability of being seen. (2) we provide the "clip ratio" to open up the viewing frustrum a bit wider, and depend on hardware scissoring. This increases the number of polygons we can trivially accept or trivially reject.

Although we do not do so in the final microcode, we could also use the hardware scissoring information in the RSP to trivially reject some triangles:

```
if ( (all the sy's of a poly are < scissor.miny) ||
    (all the sy's of a poly are > scissor.maxy) ||
    (all the sx's of a poly are < scissor.minx) ||
    (all the sx's of a poly are > scissor.maxx)) {
       /* don't bother to clip */
}
```

This would require knowing about the scissor info, but it might be worth it.

As for the actual clipping code, there are only two places that vectorize well, the clip test (done during the transform stage) and the computation of a new point (which vectorizes across *x,y,z,* (and attributes) during the intersection calculation).

The code budget for the clipper is 120 instructions. This code space is shared with lighting, using dynamic overlays.

Triangle Edge and Attribute Setup

Triangle setup is the crucial step in the pipeline. During this step, the edge slopes and attribute interpolation deltas are computed for the RDP. The interpolants and setup information is optimized for rendering in the RDP (as it should be), so there are a few things which make the setup difficult.

We can achieve full vectorization when computing the vertex attributes (*r,g,b,a,s,t,w,z*), but most of the other arithmetic (plane equation, edge setup) can vectorize over *x,y* at best.

Precision is important in the setup code. There are probably 30 additional lines of code that increase the precision of the edge and attribute slopes. This has been judged to be worth it, improving the picture quality.

Some attributes, like *z,* require additional processing. When computing attributes, we compute all the attributes (at no extra cost, due to vectorization) and only perform additional computation (or output) those attributes we really need.

Precision Issues

Overview

Our coordinate precision problems (especially *z* and *w*) really include several different problems:

- Original size and precision of the data. (less than full range)
- Choice of a perspective projection. (good one for fixed-point)
- Location of near and far clipping planes. (far as “far” away as possible, and both “tight” around data)
- Loss of precision during computations. (use extra precision for intermediate calculations)
- Choice of computation to scale *z* for z-buffer range.

Our Coordinate Spaces

- data space - S15. (+/- 32K)
- world space - S15.16 (same as data space, with fractional precision)

These two imply that if your data is really defined to use the whole space, your transform (including camera) should be careful not to move it out of the coordinate range. The fractional space helps prevent the rotation and scaling from “deforming” the data.

- screen space - S11.2, however the internal accumulators are S15.16. This limitation is primarily introduced by the intermediate terms used to calculate plane equation for the attributes.
- screen z - screen z has the same basic integer restrictions as x and y, however we use more fractional bits: S11.18. Since we pass an S15.16 number to the RDP, we shift our z up 5 bits, so the floating point format will retain the most accuracy. The hardware z uses a unique floating-point encoding that can be stored in 18 bits, but gives us almost the performance of a 32-bit z-buffer.

Our Computations

- 3D Transform - S15. * S15.16 matrix = S15.16 3D points. (these points used for clipping, w texture, z, etc.)
- Clipping - (uses S15.16 3D points, results same as projection, etc.)
- Projection - S15.16 / S15.16 = S15.16
- Viewport - S15.16 * S15. + S15 = S15.16
- Screen space - int(S15.16 << 2) = S11.2
- Setup - S11.2 y-positions
- plane equation S15.16 * S11.2 = S15.16 deltas
- adjust x’s S15.16 * S11.2 = S15.16 x-positions

Makefile

```
1   ##############################################################################
2   #
3   # Makefile for RSP Fast3D graphics microcode
4   #
5   ##############################################################################
6
7   PRDEPTH=../../../
8   include $(PRDEPTH)/PRdefs
9
10  # local defs
11  LCDEFS = $(HW_FLAGS)
12
13  # local includes
14  LCINCS = -I. -I$(ROOT)/usr/include -I$(ROOT)/usr/include/PR
15
16  LDIRT  = *.dat *.lst *.sym *.dbg *.elf *.tvd                    \
17                  gfillnops.s gfillnops.dram.s gfillnops.fifo.s
18
19  GFX_INCL =      ../gdmem.h ../gfx_regs.h
20
21  #
22  # this microcode uses all "common" microcode:
23  #
24  GFX_CODE =      ../gmain.s ../gyield.s ../gdma.s ../gimm.s      \
25                   ../grdp.s ../gdone.s ../gmtx.s ../gvtx.s       \
26                  ../gsetup.s ../gclip.s ../gflight.s     \
27                  ../newt.s ../goverlays.s ../goutxbus.s          \
28                  ../goutdram.s ../goutfifo.s ../ginit.s          \
29                  ../gsetup1.s ../gboot.s
30
31  INST_LIB_TARGETS =      gspFast3D.o gspFast3D.dram.o gspFast3D.fifo.o
32
33  TARGETS =       gspFast3D gspFast3D.dram gspFast3D.fifo \
34                  gspFast3D.u.tvd gspFast3D.dram.u.tvd    \
35                  gspFast3D.fifo.u.tvd                    \
36                  gclip.only.u.tvd gclip.only.dram.u.tvd  \
37                  gclip.only.fifo.u.tvd                   \
38                  gflight.u.tvd gflight.dram.u.tvd        \
39                  gflight.fifo.u.tvd                      \
40                  gdone.u.tvd gdone.dram.u.tvd            \
41                  gdone.fifo.u.tvd                        \
42                  ${INST_LIB_TARGETS}
43
```

```
44  default: ${TARGETS} checksize
45
46  install exports: ${TARGETS} checksize
47          $(INSTALL) -m 555 -F /usr/lib/PR $(INST_LIB_TARGETS)
48
49  #
50  # what the heck is this?
51  #
52  ig:     gspFast3D gspFast3D.u.tvd gclip.only.u.tvd gflight.u.tvd \
53          gdone.u.tvd gspFast3D.o
54          $(INSTALL) -m 555 -F /usr/lib/PR gspFast3D.o
55
56  include $(PRDEPTH)/PRrules
57
58  ###############################################################################
59  #
60  # use the RSP linker 'buildtask' to construct the tasks from the objects.
61  # use the rsp2elf program to construct the debug executables and library
62  # executables.
63  #
64  # 'dd' is used to chop the data section to the minimum required, in
65  # order to save ROM/RAM space.
66  #
67
68  #
69  # the regular 3D polygon ucode:
70  #
71  gspFast3D:      gspFast3D.u newt.u gclip.only.u gflight.u gdone.u gboot.u
72          ${BUILDTASK} -f 1 -o $@ -s 34 -p gboot.u gspFast3D.u -l 34 gspFast3D.u gclip.only.u gflight.u
     gdone.u
73          dd if=$@.dat of=tmp1.dat bs=1 count=2048
74          mv tmp1.dat $@.dat
75
76  gspFast3D.o:    gspFast3D
77          ${RSP2ELF} -p -r $?
78
79  gspFast3D.u.tvd:        gspFast3D
80          ${RSP2ELF} -p gspFast3D.u
81
82  gclip.only.u.tvd:       gspFast3D
83          ${RSP2ELF} -p gclip.only.u
84
85  gflight.u.tvd:  gspFast3D
86          ${RSP2ELF} -p gflight.u
87
```

```
 88  gdone.u.tvd:    gspFast3D
 89          ${RSP2ELF} -p gdone.u
 90
 91
 92  #
 93  # the 3D polygon ucode with output back to DRAM:
 94  #
 95  gspFast3D.dram: gspFast3D.dram.u newt.u gclip.only.dram.u gflight.dram.u gdone.dram.u gboot.dram.u
 96          ${BUILDTASK} -f 1 -o $@ -s 34 -p gboot.dram.u gspFast3D.dram.u -l 34 gspFast3D.dram.u
      gclip.only.dram.u gflight.dram.u gdone.dram.u
 97          dd if=$@.dat of=tmp2.dat bs=1 count=2048
 98          mv tmp2.dat $@.dat
 99
100  gspFast3D.dram.o:       gspFast3D.dram
101          ${RSP2ELF} -p -r $?
102
103  gspFast3D.dram.u.tvd:   gspFast3D.dram
104          ${RSP2ELF} -p gspFast3D.dram.u
105
106  gclip.only.dram.u.tvd:  gspFast3D.dram
107          ${RSP2ELF} -p gclip.only.dram.u
108
109  gflight.dram.u.tvd:     gspFast3D.dram
110          ${RSP2ELF} -p gflight.dram.u
111
112  gdone.dram.u.tvd:       gspFast3D.dram
113          ${RSP2ELF} -p gdone.dram.u
114
115
116  #
117  # the 3D polygon ucode with output to circular DRAM fifo:
118  #
119  gspFast3D.fifo: gspFast3D.fifo.u newt.u gclip.only.fifo.u gflight.fifo.u gdone.fifo.u gboot.fifo.u
120          ${BUILDTASK} -f 1 -o $@ -s 34 -p gboot.fifo.u gspFast3D.fifo.u -l 34 gspFast3D.fifo.u
      gclip.only.fifo.u gflight.fifo.u gdone.fifo.u
121          dd if=$@.dat of=tmp3.dat bs=1 count=2048
122          mv tmp3.dat $@.dat
123
124  gspFast3D.fifo.o:       gspFast3D.fifo
125          ${RSP2ELF} -p -r $?
126
127  gspFast3D.fifo.u.tvd:   gspFast3D.fifo
128          ${RSP2ELF} -p gspFast3D.fifo.u
129
130  gclip.only.fifo.u.tvd:  gspFast3D.fifo
```

```
131         ${RSP2ELF} -p gclip.only.fifo.u
132
133 gflight.fifo.u.tvd:     gspFast3D.fifo
134         ${RSP2ELF} -p gflight.fifo.u
135
136 gdone.fifo.u.tvd:       gspFast3D.fifo
137         ${RSP2ELF} -p gdone.fifo.u
138
139 #
140 # Ensure IMEM size does not overflow
141 #
142 checksize:      ${INST_LIB_TARGETS}
143         ../sizecheck ${INST_LIB_TARGETS}
144         ../labeltest
145
146
147 ###############################################################################
148 #
149 # build the individual objects.
150 #
151
152 newt.u: gspFast3D.u ../newt.s ${GFX_INCL}
153         @echo "****"
154         @echo "**** Building $@:"
155         @echo "****"
156         ${RSPASM} ${LCINCS} ${LCDEFS} -DNEWT_ALONE -S gspFast3D.u -o $@ ../newt.s
157
158 gspFast3D.u: gclip.only_tmp.u gflight.u ../fillnops gspFast3D_tmp.u ${GFX_INCL} ${GFX_CODE}
159         @echo "****"
160         @echo "**** Building $@:"
161         @echo "****"
162         ../fillnops gclip.only_tmp.u.dbg clipKill gspFast3D_tmp.u.dbg clipAndSetup 1 > gfillnops.s
163         ${RSPASM} ${LCINCS} ${LCDEFS} -DFASTLIGHT3D -o $@ -S gclip.only_tmp.u ../gmain.s
164         @echo " "
165
166 gspFast3D.dram.u: gclip.only_tmp.dram.u gflight.dram.u ../fillnops gspFast3D_tmp.dram.u ${GFX_INCL}
     ${GFX_CODE}
167         @echo "****"
168         @echo "**** Building $@:"
169         @echo "****"
170         ../fillnops gclip.only_tmp.dram.u.dbg clipKill gspFast3D_tmp.dram.u.dbg clipAndSetup 1 >
     gfillnops.dram.s
171         ${RSPASM} ${LCINCS} ${LCDEFS} -DFASTLIGHT3D -DOUTPUT_DRAM -o $@ -S gclip.only_tmp.dram.u ../gmain.s
172         @echo " "
173
```

```
174  gspFast3D.fifo.u: gclip.only_tmp.fifo.u gflight.fifo.u ../fillnops gspFast3D_tmp.fifo.u ${GFX_INCL}
      ${GFX_CODE}
175          @echo "****"
176          @echo "**** Building $@:"
177          @echo "****"
178          ../fillnops gclip.only_tmp.fifo.u.dbg clipKill gspFast3D_tmp.fifo.u.dbg clipAndSetup 1 >
      gfillnops.fifo.s
179          ${RSPASM} ${LCINCS} ${LCDEFS} -DFASTLIGHT3D -DOUTPUT_FIFO -o $@ -S gclip.only_tmp.fifo.u ../gmain.s
180          @echo " "
181
182  gboot.u: gspFast3D.u ../gboot.s
183          ${RSPASM} ${LCINCS} ${LCDEFS} -o $@ -S gspFast3D.u ../gboot.s
184
185  gboot.dram.u: gspFast3D.dram.u ../gboot.s
186          ${RSPASM} ${LCINCS} ${LCDEFS} -o $@ -S gspFast3D.dram.u ../gboot.s
187
188  gboot.fifo.u: gspFast3D.fifo.u ../gboot.s
189          ${RSPASM} ${LCINCS} ${LCDEFS} -o $@ -S gspFast3D.fifo.u ../gboot.s
190
191  gspFast3D_tmp.u:        ${GFX_INCL} ${GFX_CODE}
192          @echo "****"
193          @echo "**** Building $@:"
194          @echo "****"
195          ${RSPASM} ${LCINCS} ${LCDEFS} -DFASTLIGHT3D -DNODATA -o $@ ../gmain.s
196          @echo " "
197
198  gspFast3D_tmp.dram.u:   ${GFX_INCL} ${GFX_CODE}
199          @echo "****"
200          @echo "**** Building $@:"
201          @echo "****"
202          ${RSPASM} ${LCINCS} ${LCDEFS} -DFASTLIGHT3D -DNODATA -DOUTPUT_DRAM -o $@ ../gmain.s
203          @echo " "
204
205  gspFast3D_tmp.fifo.u:   ${GFX_INCL} ${GFX_CODE}
206          @echo "****"
207          @echo "**** Building $@:"
208          @echo "****"
209          ${RSPASM} ${LCINCS} ${LCDEFS} -DFASTLIGHT3D -DNODATA -DOUTPUT_FIFO -o $@ ../gmain.s
210          @echo " "
211
212  gclip.only_tmp.u:       gspFast3D_tmp.u ../goverlays.s ../gclip.s ${GFX_INCL}
213          @echo "****"
214          @echo "**** Building $@:"
215          @echo "****"
```

```
216         ${RSPASM} ${LCINCS} ${LCDEFS} -DCLIP_ALONE -DPRECISE_CLIP -DNODATA -o $@ -S gspFast3D_tmp.u
    ../goverlays.s
217
218 gclip.only_tmp.dram.u:  gspFast3D_tmp.dram.u ../goverlays.s ../gclip.s ${GFX_INCL}
219         @echo "****"
220         @echo "**** Building $@:"
221         @echo "****"
222         ${RSPASM} ${LCINCS} ${LCDEFS} -DCLIP_ALONE -DPRECISE_CLIP -DNODATA -DOUTPUT_DRAM -o $@ -S
    gspFast3D_tmp.dram.u ../goverlays.s
223
224 gclip.only_tmp.fifo.u:  gspFast3D_tmp.fifo.u ../goverlays.s ../gclip.s ${GFX_INCL}
225         @echo "****"
226         @echo "**** Building $@:"
227         @echo "****"
228         ${RSPASM} ${LCINCS} ${LCDEFS} -DCLIP_ALONE -DPRECISE_CLIP -DNODATA -DOUTPUT_FIFO -o $@ -S
    gspFast3D_tmp.fifo.u ../goverlays.s
229
230 gclip.only.u:           gspFast3D.u ../goverlays.s ../gclip.s ${GFX_INCL}
231         @echo "****"
232         @echo "**** Building $@:"
233         @echo "****"
234         ${RSPASM} ${LCINCS} ${LCDEFS} -DCLIP_ALONE -DPRECISE_CLIP -DNODATA -o $@ -S gspFast3D.u
    ../goverlays.s
235
236 gclip.only.dram.u:      gspFast3D.dram.u ../goverlays.s ../gclip.s ${GFX_INCL}
237         @echo "****"
238         @echo "**** Building $@:"
239         @echo "****"
240         ${RSPASM} ${LCINCS} ${LCDEFS} -DCLIP_ALONE -DPRECISE_CLIP -DNODATA -DOUTPUT_DRAM -o $@ -S
    gspFast3D.dram.u ../goverlays.s
241
242 gclip.only.fifo.u:      gspFast3D.fifo.u ../goverlays.s ../gclip.s ${GFX_INCL}
243         @echo "****"
244         @echo "**** Building $@:"
245         @echo "****"
246         ${RSPASM} ${LCINCS} ${LCDEFS} -DCLIP_ALONE -DPRECISE_CLIP -DNODATA -DOUTPUT_FIFO -o $@ -S
    gspFast3D.fifo.u ../goverlays.s
247
248 gflight.u:              gspFast3D_tmp.u ../gflight.s ../goverlays.s ${GFX_INCL}
249         @echo "****"
250         @echo "**** Building $@:"
251         @echo "****"
252         ${RSPASM} ${LCINCS} ${LCDEFS} -DFLIGHT -DNODATA -o $@ -S gspFast3D_tmp.u ../goverlays.s
253
254 gflight.dram.u:         gspFast3D_tmp.dram.u ../gflight.s ../goverlays.s ${GFX_INCL}
```

```
255         @echo "****"
256         @echo "**** Building $@:"
257         @echo "****"
258         ${RSPASM} ${LCINCS} ${LCDEFS} -DFLIGHT -DNODATA  -DOUTPUT_DRAM -o $@ -S gspFast3D_tmp.dram.u
    ../goverlays.s
259
260 gflight.fifo.u:        gspFast3D_tmp.fifo.u ../gflight.s ../goverlays.s ${GFX_INCL}
261         @echo "****"
262         @echo "**** Building $@:"
263         @echo "****"
264         ${RSPASM} ${LCINCS} ${LCDEFS} -DFLIGHT -DNODATA  -DOUTPUT_FIFO -o $@ -S gspFast3D_tmp.fifo.u
    ../goverlays.s
265
266 gdone.u:               gspFast3D_tmp.u ../gdone.s ${GFX_INCL}
267         @echo "****"
268         @echo "**** Building $@:"
269         @echo "****"
270         ${RSPASM} ${LCINCS} ${LCDEFS} -DFASTLIGHT3D -DNODATA -o $@ -S gspFast3D_tmp.u ../gdone.s
271
272 gdone.dram.u:          gspFast3D_tmp.dram.u ../gdone.s ${GFX_INCL}
273         @echo "****"
274         @echo "**** Building $@:"
275         @echo "****"
276         ${RSPASM} ${LCINCS} ${LCDEFS} -DFASTLIGHT3D -DNODATA -DOUTPUT_DRAM -o $@ -S gspFast3D_tmp.dram.u
    ../gdone.s
277
278 gdone.fifo.u:          gspFast3D_tmp.fifo.u ../gdone.s ${GFX_INCL}
279         @echo "****"
280         @echo "**** Building $@:"
281         @echo "****"
282         ${RSPASM} ${LCINCS} ${LCDEFS} -DFASTLIGHT3D -DNODATA -DOUTPUT_FIFO -o $@ -S gspFast3D_tmp.fifo.u
    ../gdone.s
283
284 # DO NOT DELETE THIS LINE -- make depend depends on it.
```

gdmem.h

```
 1
 2  /**************************************************************************
 3   *                                                                        *
 4   *               Copyright (C) 1994, Silicon Graphics, Inc.               *
 5   *                                                                        *
 6   *  These coded instructions, statements, and computer programs  contain  *
 7   *  unpublished  proprietary  information of Silicon Graphics, Inc., and  *
 8   *  are protected by Federal copyright  law.  They  may not be disclosed  *
 9   *  to  third  parties  or copied or duplicated in any form, in whole or  *
10   *  in part, without the prior written consent of Silicon Graphics, Inc.  *
11   *                                                                        *
12   **************************************************************************/
13
14   #
15   # gdmem.h
16   #
17   # This file lays out the DMEM usage for the RSP gspFast3D tasks
18   # and the gspLine3D tasks.
19   #
20
21  #include <rcp.h>
22  #include <os.h>
23  #include <sptask.h>
24
25  /*
26   * Memory layout of DMEM:
27   *
28   * The strategy is to divide up DMEM into a bunch of known regions
29   * that we can get to quickly to store/retrieve stuff.
30   *
31   * The order of things here is kind of important. Things that need
32   * to be initialized via DMA should be near the top. Some buffers
33   * require 64-bit alignment.
34   *
35   *      ------------------------------------------------
36   *      | Program data from compiler...                |     (must be first)
37   *      |     overlays, constants, etc.  272 bytes     |
38   *      |----------------------------------------------|
39   *      | RSP GFX State                                |
40   *      |                                 80 bytes     |
41   *      |----------------------------------------------|
42   *      | RSP memory segment table:                    |
43   *      |      16 x 4b =                64 bytes       |
```

```
44   *      |                                              |
45   *      |----------------------------------------------|
46   *      | Lights                                       |
47   *      |                                384 bytes     |
48   *      |----------------------------------------------|
49   *      | Viewport:                                    |
50   *      |                                 16 bytes     |
51   *      |----------------------------------------------|
52   *      | Fog factors:                                 |
53   *      |                                  6 bytes     |
54   *      |----------------------------------------------|
55   *      | padding                                      |
56   *      |                                  2 bytes     |
57   *      |----------------------------------------------|
58   *      | Display list stack:  10 deep   40 bytes      |
59   *      |                                              |
60   *      |----------------------------------------------|
61   *      | Modelview Matrix top of stack:               | 64-bit aligned
62   *      |     4x4 x 4b                                 |
63   *      |                                 64 bytes     |
64   *      |----------------------------------------------|
65   *      | Projection Matrix top of stack:              | 64-bit aligned
66   *      |     4x4 x 4b                                 |
67   *      |                                 64 bytes     |
68   *      |----------------------------------------------|
69   *      | MP matrix: (Modelview * Projection)          |
70   *      |     4x4 x 4b (doubled for vector register)   |
71   *      |                                 64 bytes     |
72   *      |----------------------------------------------|
73   *      | Points buffer:                640 bytes      |
74   *      |                                              |
75   *      | 16 points, @ 40 bytes                        |
76   *      | (includes clip coords, screen, etc.)         |
77   *      |                                              |
78   *      |----------------------------------------------|
79   *      | Input (display list)          320 bytes      | 64-bit aligned
80   *      |    40 cmds, @ 8bytes                         |
81   *      |----------------------------------------------|
82   *      | Input (data)                                 | 64-bit aligned
83   *      |                                256 bytes     |
84   *      | (needs to be big enough to hold 16 pts, etc.)|
85   *      |----------------------------------------------|
86   *      | setup tmp                       96 bytes     |
87   *      |----------------------------------------------|
88   *      | clip tmp                      160 bytes      |
```

```
 89   *      |-----------------------------------------------|
 90   *      | Output to RDP:                                | 64-bit aligned
 91   *      |    6 nasty triangles x 160b+                  |
 92   *      |    (cycle memory)           1024 bytes        |
 93   *      |-----------------------------------------------|
 94   *      | Scratch space for intermediate results        |
 95   *      | Holds clipped vertices during clip and setup. |
 96   *      |                                               |
 97   *      |                               480 bytes       |
 98   *      -------------------------------------------------
 99   *
100   */
101
102  /*
103   * Memory addressing Strategy:
104   *
105   * All dmem will be addressed relative to register zero which is
106   * equivalent to the top of dmem.
107   * Memory is allocated in this file by adding bytes, halfs, words, etc.
108   * If alignment is required attempt to align by hand and use a
109   *      .bound <byte_alignment>
110   * where <byte_alignment> is the number of bytes to align to.  This will
111   * supply an error message when alignment is incorrect.  To force alignment use
112   *      .align <byte_alignment>
113   * which will force alignment by adding pad bytes.
114   * An error will occur if dmem is overflowed.
115   *
116   */
117
118      .data
119
120   ###################################################################
121   ############## BEGIN INITIALIZING AND DEFINING DMEM USE #########
122   ###################################################################
123
124          .print  "-------------------------------------------------------\n"
125  RSP_PDATA_OFFSET:
126
127   ###################################################################
128   ############## OVERLAY TABLE ######################################
129   ###################################################################
130   #
131   # Program module overlay table. Offsets and sizes are filled in by
132   # the 'buildtask' utility, destinations are the responsibility of
133   # the ucode.
```

```
134   #
135   # OVERLAY_OFFSET:  offset from beginning of microcode in RDRAM and
136   #                    in .o file (filled in by buildtask).
137   # OVERLAY_SIZE:    length of overlay in bytes (filled in by buildtask).
138   # OVERLAY_DEST:    where in IMEM to put the overlay (filled in by
139   #                    microcode).
140   #
141   # The overlay table must be the first thing in DMEM.
142   # The 1st overlay must be the initial code.
143   #
144           .bound  0x80000000
145
146   OVERLAY_TAB_OFFSET:
147
148   #define OVERLAY_OFFSET  0
149   #define OVERLAY_SIZE    4
150   #define OVERLAY_DEST    6
151
152    #=======================================
153    #============= MAIN CODE OVERLAY ==========
154    #=======================================
155   OVERLAY_0_OFFSET:
156       # main module.
157           .word   0x0             # offset from beginning of code
158           .half   0x0             # size in bytes (-1)
159           .half   0x04001080      # destination
160
161    #=======================================
162    #============= NEWTONS OVERLAY ============
163    #=======================================
164   OVERLAY_1_OFFSET:
165   OVERLAY_NEWTON:
166       # Newton's module laid over boot code.
167           .word   0x0             # offset from beginning of code
168           .half   0x0             # size in bytes (-1)
169           .half   0x04001000      # destination
170
171    #=======================================
172    #============= CLIPPING OVERLAY ===========
173    #=======================================
174   #if defined(FASTLIGHT3D)||defined(CLIP_ALONE)||defined(FLIGHT)
175   #else
176   #   define startClip 0          # bogus startClip if not using overlays
177   #endif
178
```

```
179  OVERLAY_2_OFFSET:
180  OVERLAY_CLIP:
181      # clip code
182          .word   0x0             # offset from beginning of code
183          .half   0x0             # size in bytes (-1)
184          .half   startClip       # destination
185
186   #=======================================
187   #============ LIGHTING OVERLAY ==========
188   #=======================================
189  OVERLAY_3_OFFSET:
190  OVERLAY_LIGHT:
191      # light code
192          .word   0x0             # offset from beginning of code
193          .half   0x0             # size in bytes (-1)
194          .half   startClip       # destination
195
196   #=======================================
197   #============ DONE OVERLAY ==============
198   #=======================================
199  OVERLAY_4_OFFSET:
200  OVERLAY_DONE:
201      # TaskDone and Yield code
202          .word   0x0             # offset from beginning of code
203          .half   0x0             # size in bytes (-1)
204  TASKYIELD:
205          .half   startClip       # destination
206
207   ### ADD NEW OVERLAY TABLE ENTRIES HERE
208
209   #
210   #
211   ############## END OF OVERLAY TABLE ##############################
212
213   ###################################################################
214   ############## QUAD CONSTANTS ####################################
215   ###################################################################
216   #
217   # Constants. Setup a bunch of constants at compile-time. Be
218   # sure to #define the macros and offsets needed to access them.
219   #
220   ###################################################################
221   #
222   #
223   # Initialize vconst; each element a different scalar constant...
```

```
224   #
225
226           .bound  8
227   VCONST_SCREENCLAMP:
228           .half   4090            # +xy clamp to +1000 (*4)
229           .half   -4090           # -xy clamp to -1000 (*4)
230
231           .half   0x7fff          # +z clamp max
232           .half   0x0000          # -z clamp to 0
233
234           .align  16
235           .bound  16
236
237   VCONST_OFFSET:
238           .half   0x0000          # 0
239           .half   0x0001          # 1
240           .half   0x0002          # 2
241           .half   0xffff          # -1
242           .half   0x4000          # 1/4.0 in fixed-point fraction
243           .half   0x0004          # 4.0, for screen point scaling.
244           .half   0x0633          # (2047-4e0) for screen-space clamping.
245           .half   0x0200          # 1/512.0 for pixel-packed loads
246    #
247    # Initialize vconst; each element a different scalar constant...
248    #
249   #ifdef LINE3D
250   VCONST1_OFFSET:
251           .half   0x7fff          # for setup w scale
252           .half   0xfff8          # to clear lower bits for EW fracs
253           .half   0x0008          # mult by 8 for LOD computation.
254           .half   0x0010          #
255           .half   0x0020          # for z scale
256           .half   0x8000          #
257           .half   0x0755          # 1877=(2048-640)/0.75 for slope clamp
258           .half   0x0000          # not used
259
260   #else /* LINE3D */
261   VCONST1_OFFSET:
262           .half   0x7fff          # for setup w scale
263           .half   0xfff8          # to clear lower bits for EW fracs
264           .half   0x0008          # mult by 8 for LOD computation.
265           .half   0x0040          # 1/128 * .5 * .25 (sqrt correct) (.5 for sf
266           .half   0x0020          # for z scale
267           .half   0x8000          # test bow-tie fix
268           .half   0x01cc          # 460 = 2048/3 for slope clamp
```

```
269                                 # WARNING: also used in lighting.
270             .half   0xcccc      # W multiplier(0.8). 1/w ranges from 0 to 0.8
271     #endif /* LINE3D */
272
273      #
274      # Initialize OpenGL correction scale
275      #
276     VOPENGL_OFFSET:
277             .half   0x0001      # x
278             .half   0xffff      # y
279             .half   0x0001      # z
280             .half   0x0001      # w
281             .half   0x0001      # x
282             .half   0xffff      # y
283             .half   0x0001      # z
284             .half   0x0001      # w
285
286      #
287      # Initialize constant vector for Newton's iteration
288      #
289     VNEWT_OFFSET:
290             .half   0x0002      # this element not used
291             .half   0x0002      # this element not used
292             .half   0x0002      # this element not used
293             .half   0x0002      # MUST BE 2 !
294             .half   0x0002      # this element not used
295             .half   0x0002      # this element not used
296             .half   0x0002      # this element not used
297             .half   0x0002      # MUST BE 2 !
298      #
299      #
300      ############## END OF CONSTANTS ###################################
301
302      ##################################################################
303      ############## CLIPPING & LIGHTING ################################
304      ##################################################################
305      #
306      # static tables and masks for clipping & lighting
307      #
308      ##################################################################
309      #
310             .bound  16
311     CLIP_SELECT:                        # NOTE: MOVEWORD command alters these
312             .word   0x00010000          # -x plane (1 0 0  1)
313             .word   0x00000001          # -x
```

```
314         .word   0x00000001             # -y plane (0 1 0  1)
315         .word   0x00000001             # -y
316         .word   0x00010000             # +x plane (1 0 0 -1)
317         .word   0x0000ffff             # +x
318         .word   0x00000001             # +y plane (0 1 0 -1)
319         .word   0x0000ffff             # +y
320         .word   0x00000000             # +z plane (0 0 1 -1)
321         .word   0x0001ffff             # +z
322         .word   0x00000000             # -z plane (0 0 1  1)
323         .word   0x00010001             # -z   if NEARCLIP_OFF: (0 0 0 1)
324
325         .bound  8
326 #if defined(FASTLIGHT3D)||defined(CLIP_ALONE)||defined(FLIGHT)
327 DOLIGHT:
328         .half   doLight
329 #else /* FASTLIGHT3D or CLIP_ALONE */
330 DOLIGHT:
331         .half   0              # pad
332 #endif /* FASTLIGHT3D or CLIP_ALONE */
333         .half   0x7fff         # 1-.0x0001
334         .half   0x571d         # 0x571d for arccos
335         .half   0x3a0c         # 0x3a0b+.0x0001 for arccos
336
337         .bound 2
338 CLIP_MASKS_TABLE:
339 CLIPMASKS:
340         .half   0x0001                 # -x plane
341         .half   0x0002                 # -y plane
342         .half   0x0100                 # +x plane
343         .half   0x0200                 # +y plane
344         .half   0x4000                 # +z plane
345         .half   0x0040                 # -z plane (not always used)
346
347 ANCHOR:
348         .half   0x0000
349 TASKDONE:
350 #if defined(FASTLIGHT3D)||defined(CLIP_ALONE)||defined(FLIGHT)
351         .half   taskDone
352 #else /* FASTLIGHT3D or CLIP_ALONE */
353         .half   0              # pad
354 #endif /* FASTLIGHT3D or CLIP_ALONE */
355
356  #
357  #
358  ############### END OF CLIPPING & LIGHTING ########################
```

```
359
360   ##################################################################
361   ############## SCALAR CONSTANTS ##################################
362   ##################################################################
363   #
364   # Constants. Setup a bunch of constants at compile-time. Be
365   # sure to #define the macros and offsets needed to access them.
366   #
367   ##################################################################
368   #
369
370   #
371   # a common necessary mask value
372   #
373
374           .bound  4
375   SEGADDR_MASK_OFFSET:
376             .word   0x00ffffff      # segment address mask.
377
378   #
379   #
380   ############## END OF SCALAR CONSTANTS ###########################
381
382   ##################################################################
383   ############## JUMP TABLES #######################################
384   ##################################################################
385   #
386   # Jump Tables: These DMEM jump tables are only 16 bits, that's
387   # plenty, given only 4K of IMEM...
388   #
389   # Jump tables should be 'complete', any new command in the
390   # mbi should be included here, even if it's a jump to GfxDone
391   # in some cases (a no-op).
392   #
393   # Notice also that a bunch of #ifdef's are used to disable
394   # certain commands for certain versions of the microcode.
395   # Patching the 'GfxDone' label into the table makes a command
396   # be a no-op.
397   #
398   ##################################################################
399   #
400
401   #==================================================================
402   # setup a jump table for the optypes:
403   # This table is the high-level handler for the display list commands.
```

```
404   # Based on the upper 2 bits, each different type of command is handled
405   # differently.
406   #=====================================================================
407         .bound  2
408
409   OPTYPE_JMP_OFFSET:
410         .half   doDMA
411   GFXDONE:
412         .half   GfxDone         # dummy label, invalid op type
413         .half   doIMM
414         .half   doRDP
415
416   #=====================================================================
417   # setup a jump table for the DMA ops:
418   #=====================================================================
419   DMA_JMP_OFFSET:
420         .half   GfxDone         # all 0's is G_SPNOOP
421         .half   case_G_MTX
422         .half   GfxDone         # case_G_MTX_STATE not implemented
423         .half   case_G_MOVEMEM  # viewport and light
424         .half   case_G_VTX
425         .half   GfxDone         # case_G_TRIN not implemented
426         .half   case_G_DL
427         .half   GfxDone         # not implemented
428         .half   GfxDone         # not implemented
429         .half   GfxDone
430
431   #=====================================================================
432   # setup a jump table for the IMM ops:
433   # IMPORTANT: ADD NEW OPCODES AT TOP OF THIS TABLE
434   #=====================================================================
435   IMM_JMP_OFFSET:
436         .half   case_G_RDPHALF_CONT
437         .half   case_G_RDPHALF_2
438         .half   case_G_RDPHALF_1
439         .half   case_G_PERSPNORM
440   #ifdef LINE3D
441         .half   case_G_LINE3D
442   #else
443         .half   GfxDone             # Lines not grokked in triangle code
444   #endif
445         .half   case_G_CLEARGEOMETRYMODE
446         .half   case_G_SETGEOMETRYMODE
447         .half   case_G_ENDDL
448         .half   case_G_SETOTHERMODE_L
```

```
449         .half   case_G_SETOTHERMODE_H
450         .half   case_G_TEXTURE
451         .half   case_G_MOVEWORD
452         .half   case_G_POPMTX
453         .half   case_G_CULLDL
454 #ifdef LINE3D
455         .half   GfxDone             # Tri's not grokked in line or sprite code
456 #else
457         .half   case_G_TRI1
458 #endif
459  # ADD NEW IMM OPCODES ONLY AT TOP OF LIST, NOT HERE
460
461         .symbol NUMBER_OF_IMM,  15
462         .symbol IMM_JMP_ADD,    (IMM_JMP_OFFSET)+(((G_IMMFIRST)+(NUMBER_OF_IMM)+1)*2)
463
464  #=====================================================================
465  # no jump table for the RDP ops is needed.
466  #=====================================================================
467
468  #
469  #
470  ############## END OF JUMP TABLES ################################
471
472  ##################################################################
473  ############## LABEL CONSTANTS ###################################
474  ##################################################################
475  #
476  # Labels for loading jump addresses
477  #
478  ##################################################################
479  #
480
481         .bound 2
482
483  #=======================================
484  #============= DUMMY LABELS ============
485  #=======================================
486 #ifdef NODATA
487 CLIP_STATE_TABLE:
488         .half   0
489 FOUND_OUT:
490         .half   0
491         .half   0
492         .half   0
493
```

```
494   CLIPDRAWLOOP:
495           .half   0
496   DOCLIP:
497           .half   0
498   NEXTCLIP:
499           .half   0
500   #else /* NODATA */
501
502   #ifdef LINE3D
503   CLIP_STATE_TABLE:
504           .half   0
505   FOUND_OUT:
506           .half   0
507           .half   0
508           .half   0
509
510   CLIPDRAWLOOP:
511           .half   0
512   #else  /* LINE3D */
513
514    #=======================================
515    #============= REAL LABELS ==============
516    #=======================================
517   CLIP_STATE_TABLE:
518           .half   foundIn
519   FOUND_OUT:
520           .half   foundOut
521           .half   foundFirstIn
522           .half   foundFirstOut
523
524   CLIPDRAWLOOP:
525           .half   clipDrawLoop
526   #endif  /* LINE3D */
527
528   DOCLIP:
529           .half   doClip
530   NEXTCLIP:
531           .half   nextClip
532   #endif /* NODATA */
533
534   DMAWAITDL:
535           .half   DMAWaitDL
536
537    #
538    #
```

```
539   ############## END OF LABELS ####################################
540
541   PROGRAM_PAD_OFFSET:
542
543   ########################################################################
544   ############## DMEM STATE ##############################################
545   ########################################################################
546   #
547   # Up to this point, we have been just filling in compile-time
548   # structures. The rest of this initialization sets up the DMEM
549   # state for the beginning of the graphics task.
550   #
551   # Alignment is important, changes above might screw up initialization
552   # in a bad way...
553   #
554   ########################################################################
555   #
556   #
557   # initialize RSP GFX state.
558   #
559           .bound  2
560   RETURNJUMP:
561           .half   0x0000
562
563           .bound  4
564   RSP_STATEP_YIELD_STORE:
565           .word   0x0                     # where to store yield dmem
566
567           .bound  4
568   RSP_STATEP_RDPHALF:
569           .word   0x0                     # RSP_STATE_RDPHALF
570
571           .align  8
572           .bound  8
573   RSP_STATE_OFFSET:
574
575           .bound  1
576   RSP_STATEP_DL_N:
577           .byte   0x0                     # RSP_STATE_DL_N
578
579           .align  2
580           .bound  2
581   RSP_STATEP_PERSPNORM:
582           .half   0xffff                  # RSP_STATE_PERSPNORM
583
```

```
584         .align  4
585         .bound  4
586 RSP_STATEP_RENDER:
587         .half   0x0                 # RSP_STATE_RENDER
588 RSP_STATEP_RENDER_L:
589         .byte   0x0                 # RSP_STATE_RENDER
590 RSP_STATEP_TRI:
591         .byte   0x0                 # RSP_STATE_RENDER
592
593         .bound  8
594 RSP_STATEP_OTHER_H:
595         .byte   G_RDPSETOTHERMODE   # RSP_STATE_OTHER_H
596         .byte   0x08                # RSP_STATE_OTHER_H
597         .byte   0x0c                # RSP_STATE_OTHER_H
598         .byte   0xff                # blend mask is fixed
599 RSP_STATEP_OTHER_L:
600         .word   0x0                 # RSP_STATE_OTHER_L
601
602         .bound  8
603 RSP_STATEP_TEX_CMD:
604         .byte   0x0                 # RSP_STATE_TEX_CMD
605 RSP_STATEP_TEX_LOD:
606         .byte   0x0                 # RSP_STATE_TEX_LOD
607 RSP_STATEP_TEX_TILE:
608         .byte   0x0                 # RSP_STATE_TEX_TILE
609 RSP_STATEP_TEX_PAD:
610         .byte   0x0
611 RSP_STATEP_TEX_SCALE_S:
612         .half   0x0                 # RSP_STATE_TEX_SCALE_S
613 RSP_STATEP_TEX_SCALE_T:
614         .half   0x0                 # RSP_STATE_TEX_SCALE_T
615
616         .bound  4
617 RSP_STATEP_FIFO_OUTP:
618 RSP_STATEP_DRAM_OUTP:
619         .word   0x0                 # RSP_STATE_DRAM_OUTP
620
621 RSP_STATEP_L_LEN:                   # altered by MOVEWORD command
622         .word   0x80000040          # lights * RSP_L_LEN (1 light default)
623
624         .bound  4
625 RSP_STATEP_DRAM_STACK:
626         .word   0x0                 # RSP_STATE_DRAM_STACK
627
628         .bound  4
```

```
629 RSP_STATEP_MMIX_STACK_P:
630         .word   0x0                 # RSP_STATE_MMIX_STACK_P
631
632 RSP_STATEP_TXTR_ATTR:                       # Texture Attributes
633 RSP_STATEP_XOFF:
634         .half   0x4000              # x offset
635 RSP_STATEP_YOFF:
636         .half   0x4000              # y offset
637 RSP_STATEP_HCURVE:
638         .half   0                   # horizontal curve
639 RSP_STATEP_VCURVE:
640         .half   0                   # vertical curve
641         .bound  16                  # MUST BE LAST 4 half'S OF QUAD
642
643
644         .bound 8
645 RSP_STATEP_DRAM_OUT_LENP:           # RSP_STATE_DRAM_OUT_LENP
646         .word   0x0
647
648 #ifdef LINE3D
649         .bound  4
650 RSP_STATEP_SCISSOR_XH:
651         .half   0x0000              # RSP_STATE_SCISSOR_XH
652 RSP_STATEP_SCISSOR_YH:
653         .half   0x0000              # RSP_STATE_SCISSOR_YH
654 RSP_STATEP_SCISSOR_XL:
655         .half   0x0000              # RSP_STATE_SCISSOR_XL
656 RSP_STATEP_SCISSOR_YL:
657         .half   0x0000              # RSP_STATE_SCISSOR_YL
658 #else  /* LINE3D */
659         .half   0x0000              # pad
660         .half   0x0000              # pad
661         .half   0x0000              # pad
662         .half   0x0000              # pad
663 #endif /* LINE3D */
664
665         .align 8
666         .bound 8
667 RSP_STATEP_FIFO_BUF_TOP:
668 RSP_STATEP_DRAM_OUT_LEN:
669         .word   0x0                 # RSP_STATE_DRAM_OUT_LEN
670 RSP_STATEP_FIFO_BUF_END:
671         .word   0x0                 # is a 64-bit counter
672
673         .bound  2
```

```
674 RETURNJUMP2:
675         .half   0x0000
676
677         .align  4
678         .bound  4
679 RSP_STATEP_MMIX_STACK_MAX:
680         .word   0x0                     # RSP_STATE_MMIX_STACK_MAX
681
682
683
684     .symbol RSP_STATE_YIELD_STORE,     (RSP_STATEP_YIELD_STORE) - (RSP_STATE_OFFSET)
685     .symbol RSP_STATE_DL_N,            (RSP_STATEP_DL_N) - (RSP_STATE_OFFSET)
686     .symbol RSP_STATE_OTHER_H,         (RSP_STATEP_OTHER_H) - (RSP_STATE_OFFSET)
687     .symbol RSP_STATE_OTHER_L,         (RSP_STATEP_OTHER_L) - (RSP_STATE_OFFSET)
688     .symbol RSP_STATE_RENDER,          (RSP_STATEP_RENDER) - (RSP_STATE_OFFSET)
689     .symbol RSP_STATE_RENDER_L,        (RSP_STATEP_RENDER_L) - (RSP_STATE_OFFSET)
690     .symbol RSP_STATE_TRI,             (RSP_STATEP_TRI) - (RSP_STATE_OFFSET)
691     .symbol RSP_STATE_DRAM_STACK,      (RSP_STATEP_DRAM_STACK) - (RSP_STATE_OFFSET)
692     .symbol RSP_STATE_MMIX_STACK_P,    (RSP_STATEP_MMIX_STACK_P) - (RSP_STATE_OFFSET)
693     .symbol RSP_STATE_MMIX_STACK_MAX,  (RSP_STATEP_MMIX_STACK_MAX) - (RSP_STATE_OFFSET)
694     .symbol RSP_STATE_TEX_CMD,         (RSP_STATEP_TEX_CMD) - (RSP_STATE_OFFSET)
695     .symbol RSP_STATE_TEX_LOD,         (RSP_STATEP_TEX_LOD) - (RSP_STATE_OFFSET)
696     .symbol RSP_STATE_TEX_TILE,        (RSP_STATEP_TEX_TILE) - (RSP_STATE_OFFSET)
697     .symbol RSP_STATE_TEX_SCALE_S,     (RSP_STATEP_TEX_SCALE_S) - (RSP_STATE_OFFSET)
698     .symbol RSP_STATE_TEX_SCALE_T,     (RSP_STATEP_TEX_SCALE_T) - (RSP_STATE_OFFSET)
699     .symbol RSP_STATE_FIFO_OUTP,       (RSP_STATEP_FIFO_OUTP) - (RSP_STATE_OFFSET)
700     .symbol RSP_STATE_FIFO_BUF_TOP,    (RSP_STATEP_FIFO_BUF_TOP) - (RSP_STATE_OFFSET)
701     .symbol RSP_STATE_FIFO_BUF_END,    (RSP_STATEP_FIFO_BUF_END) - (RSP_STATE_OFFSET)
702     .symbol RSP_STATE_DRAM_OUTP,       (RSP_STATEP_DRAM_OUTP) - (RSP_STATE_OFFSET)
703     .symbol RSP_STATE_DRAM_OUT_LEN,    (RSP_STATEP_DRAM_OUT_LEN) - (RSP_STATE_OFFSET)
704     .symbol RSP_STATE_DRAM_OUT_LENP,   (RSP_STATEP_DRAM_OUT_LENP) - (RSP_STATE_OFFSET)
705     .symbol RSP_STATE_L_LEN,           (RSP_STATEP_L_LEN) - (RSP_STATE_OFFSET)
706     .symbol RSP_STATE_PERSPNORM,       (RSP_STATEP_PERSPNORM) - (RSP_STATE_OFFSET)
707     .symbol RSP_STATE_RDPHALF,         (RSP_STATEP_RDPHALF) - (RSP_STATE_OFFSET)
708 #ifdef LINE3D
709     .symbol RSP_STATE_SCISSOR_XH,      (RSP_STATEP_SCISSOR_XH) - (RSP_STATE_OFFSET)
710     .symbol RSP_STATE_SCISSOR_XL,      (RSP_STATEP_SCISSOR_XL) - (RSP_STATE_OFFSET)
711     .symbol RSP_STATE_SCISSOR_YH,      (RSP_STATEP_SCISSOR_YH) - (RSP_STATE_OFFSET)
712     .symbol RSP_STATE_SCISSOR_YL,      (RSP_STATEP_SCISSOR_YL) - (RSP_STATE_OFFSET)
713 #endif /* LINE3D */
714     .symbol RSP_STATE_HCURVE,          (RSP_STATEP_HCURVE) - (RSP_STATE_OFFSET)
715     .symbol RSP_STATE_VCURVE,          (RSP_STATEP_VCURVE) - (RSP_STATE_OFFSET)
716     .symbol RSP_STATE_XOFF,            (RSP_STATEP_XOFF) - (RSP_STATE_OFFSET)
717     .symbol RSP_STATE_YOFF,            (RSP_STATEP_YOFF) - (RSP_STATE_OFFSET)
718
```

```
719   #
720   #
721   ############## END OF RSP STATE ###################################
722
723
724   ##################################################################
725   ############## MEMORY SEGMENT TABLE ###############################
726   ##################################################################
727   #
728   # initialize memory segment table.
729   # altered by MOVEWORD command
730   #
731           .align  4
732           .bound  4
733   RSP_SEG_OFFSET:
734           .word   0x0
735           .word   0x0
736           .word   0x0
737           .word   0x0
738           .word   0x0
739           .word   0x0
740           .word   0x0
741           .word   0x0
742           .word   0x0
743           .word   0x0
744           .word   0x0
745           .word   0x0
746           .word   0x0
747           .word   0x0
748           .word   0x0
749           .word   0x0
750   #
751   #
752   ############## END OF MEMORY SEGMENT TABLE ########################
753
754   ##################################################################
755   ############## REFLECTANCE TABLE ##################################
756   ##################################################################
757   #
758   #
759   # define offsets for the light structure:
760   #
761   #  bytes:
762   #  0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15
763   #  .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
```

```
764   #  ------------------------------------------------------------------
765   #  |S.x|S.y|S.z|000|S.x|S.y|S.z|000|000|000|000|000|000|000|000|000|
766   #  ------------------------------------------------------------------
767   #  |T.x|T.y|T.z|000|T.x|T.y|T.z|000|TVx|TVy|TVz|000|000|000|000|000|
768   #  ------------------------------------------------------------------
769   #  |TMx|TMy|TMz|000|TMx|TMy|TMz|000|000|000|000|000|000|000|000|000|
770   #  ------------------------------------------------------------------
771   #  |D.x|D.y|D.z|000|D.x|D.y|D.z|000|SVx|SVy|SVz|000|000|000|000|000|
772   #  ------------------------------------------------------------------
773   #  |SMx|SMy|SMz|000|SMx|SMy|SMz|000|000|000|000|000|000|000|000|000|
774   #  ------------------------------------------------------------------
775   #
776   # S.xyz = S select vector (always 0x80000000 = [-1  0 0])
777   # T.xyz = T select vector (always 0x00800000 = [ 0 -1 0])
778   # TVxyz = T direction in Viewspace (Up vector)
779   # TMxyz = T direction in Modelspace (calculated by microcode)
780   # SVxyz = S direction in Viewspace (Right vector)
781   # SMxyz = S direction in Modelspace (calculated by microcode)
782   # D.xyz = Dummy color (must always be 0x00000000 = [0 0 0])
783   #
784   #
785  #define RSP_R_SCALE     0
786  #define RSP_R_DIR_VIEW  8
787  #define RSP_R_PAD       12
788  #define RSP_R_DIR       16
789
790
791          .align  16
792          .bound  16      /* must be quad aligned */
793  RSP_LIGHTS_OFFSET:
794  RSP_SELECT_S:
795          .word   0x80000000      # S select vector
796          .word   0x80000000      # S select vector
797          .word   0x0             # pad
798          .word   0x0             # pad
799
800  RSP_R_LOOKATX:
801          .word   0x00800000      # T select vector
802          .word   0x00800000      # T select vector copy
803          .word   0x7f000000      # reflectance direction (world) T (x)
804          .word   0x0             # pad
805          .word   0x0             # reflectance direction (modeling) T
806          .word   0x0             # reflectance direction copy
807          .word   0x0             # pad
808          .word   0x0             # pad
```

```
809
810  RSP_R_LOOKATY:
811          .word   0x00000000      # dummy color (must be 0)
812          .word   0x00000000      # dummy color (must be 0)
813          .word   0x007f0000      # reflectance direction (world) S (y)
814          .word   0x0             # pad
815          .word   0x0             # reflectance direction (modeling) S
816          .word   0x0             # reflectance direction copy
817          .word   0x0             # pad
818          .word   0x0             # pad
819   #
820   #
821   ############### END OF REFLECTANCE TABLE ##########################
822
823   ### NOTE: DO NOT PUT ANYTHING BETWEEN THE REFLECTANCE AND LIGHT TABLES
824
825   ###################################################################
826   ############### LIGHTING TABLE ####################################
827   ###################################################################
828   #
829   # IMPORTANT!!! THIS MUST IMMEDIATELY FOLLOW THE REFLECTANCE TABLE!!!
830   #
831   # define offsets for the light structure:
832   #
833   #  bytes:
834   #  0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15
835   #  .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
836   #  ---------------------------------------------------------------
837   #  |c.r|c.g|c.b|   |c.r|c.g|c.b|   |w.x|w.y|w.z|   |   |   |   |
838   #  ---------------------------------------------------------------|
839   #  |m.x|m.y|m.z|   |m.x|m.y|m.z|   |   |   |   |   |   |   |   |
840   #  ---------------------------------------------------------------|
841   #
842   # c.r   light color, red componant    (supplied vi MOVEMEM command)
843   # c.g   light color, green componant  (supplied vi MOVEMEM command)
844   # c.b   light color, blue componant   (supplied vi MOVEMEM command)
845   # w.x   light direction, world coords (supplied vi MOVEMEM command)
846   # w.y   light direction, world coords (supplied vi MOVEMEM command)
847   # w.z   light direction, world coords (supplied vi MOVEMEM command)
848   # m.x   light direction, model coords (calculated by the microcode)
849   # m.y   light direction, model coords (calculated by the microcode)
850   # m.z   light direction, model coords (calculated by the microcode)
851   # blank pad, don't care (can be any value)
852   #
853   # There are 2 copies of the light color and 2 copies of the modeling
```

```
854    # coordinate light direction.
855    #
856    # light direction is a vector pointing from the object towards the light.
857    #
858    # NOTE: the last light in the table is the ambient light of which only c.r,
859    #       c.g, and c.b are used.
860    #
861    #
862    #define RSP_L_COL       0
863    #define RSP_L_DIR_VIEW  8
864    #define RSP_L_PAD       12
865    #define RSP_L_DIR       16
866    #define RSP_L_LEN       32
867
868            .bound  16      /* must be quad aligned. */
869    RSP_L_0:
870    RSP_LIGHT_COL:
871            .word   0x0             # light #1 color (black)
872            .word   0x0             # light #1 color copy
873            .word   0x0             # light #1 direction (world)
874            .word   0x0             # pad
875            .word   0x0             # light #1 direction (modeling)
876            .word   0x0             # light #1 direction copy
877
878        # this constant is stuck in the pad bytes for this light
879    VCONST3_OFFSET:
880            .half   0xe001          # min clamp for lighting direction
881            .half   0x1fff          # max clamp for lighting direction
882            .half   4               # multiplier after clamp
883            .half   0               # pad
884
885    RSP_L_1:
886            .word   0xFF000000      # ambient color (red)
887            .word   0xFF000000      # ambient color copy
888            .word   0x0
889            .word   0x0
890            .word   0x0
891            .word   0x0
892            .word   0x0             # pad
893            .word   0x0             # pad
894    RSP_L_2:
895            .word   0x0             # light #2 (uninitialized)
896            .word   0x0
897            .word   0x0
898            .word   0x0
```

```
899             .word   0x0
900             .word   0x0
901             .word   0x0             # pad
902             .word   0x0             # pad
903     RSP_L_3:
904             .word   0x0             # light #3 (uninitialized)
905             .word   0x0
906             .word   0x0
907             .word   0x0
908             .word   0x0
909             .word   0x0
910             .word   0x0             # pad
911             .word   0x0             # pad
912     RSP_L_4:                        # light #4 (initialized to magic values)
913             .word   0x52535020
914             .word   0x53572056
915             .word   0x65727369
916             .word   0x6f6e3a20
917             .word   0x322e3044
918             .word   0x2c203034
919             .word   0x2d30312d
920             .word   0x39360053
921     RSP_L_5:                        # light #5 (initialized to magic values)
922             .word   0x47492055
923             .word   0x36342047
924             .word   0x46582053
925             .word   0x57205445
926             .word   0x414d3a20
927             .word   0x5320416e
928             .word   0x64657273
929             .word   0x6f6e2c20
930     RSP_L_6:                        # light #6 (initialized to magic values)
931             .word   0x53204361
932             .word   0x72722c20
933             .word   0x48204368
934             .word   0x656e672c
935             .word   0x204b204c
936             .word   0x75737465
937             .word   0x722c2052
938             .word   0x204d6f6f
939     RSP_L_7:                        # light #7 (initialized to magic values)
940             .word   0x72652c20
941             .word   0x4e20506f
942             .word   0x6f6c6579
943             .word   0x2c204120
```

```
944           .word   0x5372696e
945           .word   0x69766173
946           .word   0x616e0a00
947           .word   0x00000000
948
949           .symbol RSP_L_BASE,     (RSP_L_0)-((RSP_L_LEN)*2)
950           .symbol RSP_L_NUM,      RSP_STATEP_L_LEN
951
952   #
953   #
954   ############### END OF LIGHTING TABLE ##############################
955
956   ### NOTE: DO NOT PUT ANYTHING BETWEEN THE LIGHTING AND MOVEMEM TABLES
957
958   ######################################################################
959   ############### MOVEMEM TABLE ########################################
960   ######################################################################
961   # This is a table of locations to store quad words which have been
962   # DMA'd with the MOVEMEM DMA command.  The MOVEMEM command
963   # specifies an index into this table to indicate where the 4 words
964   # of DMA'd data should be stored.
965
966           .bound  2
967   MOVEMEM_TBL:
968           .half   RSP_VIEWPORT_OFFSET     # viewport address
969           .half   RSP_R_LOOKATX           # lookat X vector in world space
970           .half   RSP_R_LOOKATY           # lookat Y vector in world space
971           .half   RSP_L_0                 # light 0
972           .half   RSP_L_1                 # light 1
973           .half   RSP_L_2                 # light 2
974           .half   RSP_L_3                 # light 3
975           .half   RSP_L_4                 # light 4
976           .half   RSP_L_5                 # light 5
977           .half   RSP_L_6                 # light 6
978           .half   RSP_L_7                 # light 7
979           .half   RSP_STATEP_TXTR_ATTR    # Texture Attributes
980           .half   RSP_CURR_MPMTX_OFFSET_2 # Matrix part 2
981           .half   RSP_CURR_MPMTX_OFFSET_3 # Matrix part 3
982           .half   RSP_CURR_MPMTX_OFFSET_4 # Matrix part 4
983
984   MOVEWORD_TBL:
985           .half   RSP_CURR_MPMTX_OFFSET   # matrix IMPORTANT! Must be first! (for movemem)
986           .half   RSP_L_NUM               # number of active lights * RSP_L_LEN
987           .half   CLIP_SELECT             # clip lookup table
988           .half   RSP_SEG_OFFSET          # segment table
```

```
989         .half   RSP_FOG_FACTOR         # fog multipliers
990         .half   RSP_LIGHT_COL          # light colors
991         .half   RSP_POINTS_OFFSET      # points buffer
992   #
993   #
994   ############## END OF MOVEMEM TABLE ##############################
995
996   ####################################################################
997   ############## MORE CONSTS ########################################
998   ####################################################################
999   #
1000  #
1001  #
1002  ############## END OF MORE CONSTS ################################
1003
1004  ####################################################################
1005  ############## VIEWPORT ###########################################
1006  ####################################################################
1007  #
1008  # define offsets for the viewport structure:
1009  #
1010  #  bytes:
1011  #  0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15
1012  #  .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
1013  #  ----------------------------------------------------------------
1014  #  |scal.x |scal.y |scal.z |       |trns.x |trns.y |trns.z |       |
1015  #  ----------------------------------------------------------------
1016  #
1017  # These guys have 1-bit of fraction to them. This is factored out
1018  # during the transform.
1019  #
1020  #
1021  #define RSP_VIEWPORT_SX         0
1022  #define RSP_VIEWPORT_SY         2
1023  #define RSP_VIEWPORT_SZ         4
1024  #define RSP_VIEWPORT_TX         8
1025  #define RSP_VIEWPORT_TY         10
1026  #define RSP_VIEWPORT_TZ         12
1027
1028        .align  16
1029        .bound  16
1030  RSP_VIEWPORT_OFFSET:
1031        .word   0x0
1032        .word   0x0
1033        .word   0x0
```

```
1034            .word   0x0
1035     #
1036     #
1037     ############### END OF VIEWPORT ###################################
1038
1039     ####################################################################
1040     ############### FOG FACTOR #########################################
1041     ####################################################################
1042     #
1043            .bound  16
1044    RSP_FOG_FACTOR:                         # NOTE: altered by MOVEWORD
1045            .half   0x0100  # multiplier
1046            .half   0x0000  # add
1047            .half   0x00ff  # clamp
1048     #
1049     #
1050     ############### END OF FOG FACTOR ##################################
1051
1052     ####################################################################
1053     ############### DISPLAY LIST STACK #################################
1054     ####################################################################
1055     #
1056
1057    RSP_DLSTACK_OFFSET:
1058            .word   0x0
1059            .word   0x0
1060            .word   0x0
1061            .word   0x0
1062            .word   0x0
1063            .word   0x0
1064            .word   0x0
1065            .word   0x0
1066            .word   0x0
1067            .word   0x0
1068     #
1069     #
1070     ############### END OF DISPLAY LIST STACK ##########################
1071
1072     ####################################################################
1073     ############### END OF INITIALIZED DATA#############################
1074     ####################################################################
1075            .align  16
1076            .bound  16
1077    RSP_END_INITIALIZE:
1078
```

```
1079  ################################################################
1080  ############## MATRICES ########################################
1081  ################################################################
1082  #
1083          .bound  16
1084  RSP_CURR_MMIX_OFFSET:   /* 64 bytes per matrix */
1085          .space  64
1086
1087  RSP_CURR_PMIX_OFFSET:   /* 64 bytes per matrix */
1088          .space  64
1089
1090
1091  RSP_CURR_MPMIX_OFFSET:  /* 64 bytes per matrix */
1092          .space  16      /* 1st quad word */
1093  RSP_CURR_MPMIX_OFFSET_2:
1094          .space  16      /* 2nd quad word */
1095  RSP_CURR_MPMIX_OFFSET_3:
1096          .space  16      /* 3rd quad word */
1097  RSP_CURR_MPMIX_OFFSET_4:
1098          .space  16      /* 4th quad word */
1099  #
1100  #
1101  ############## END OF MATRICES ###################################
1102
1103  ################################################################
1104  ############## POINTS BUFFER #####################################
1105  ################################################################
1106  #
1107  # define offsets for the points buffer structure:
1108  #
1109  #  bytes:
1110  #  0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15
1111  #  .   .   .   .   .   .   .   .   .   .   .   .   .   .   .   .
1112  #  ---------------------------------------------------------------
1113  #  | x.int | y.int | z.int | w.int | x.frac| y.frac| z.frac| w.frac|
1114  #  ---------------------------------------------------------------|
1115  #  | r | g | b | a |   s   |   t   | xscr  | yscr  | zscr.i|zscr.f |
1116  #  ---------------------------------------------------------------
1117  #  | 1/w.i | 1/w.f |  0c0c |flg| 00|
1118  #  --------------------------------
1119  #
1120  #  depending on flag, color could be normal (nx, ny, nz).
1121  #
1122  # 32-bit 3D xyzw might be overkill; but we do need the resolution for z,
1123  # and certainly for intermediate computations.
```

```
1124    #
1125    #
1126    #define RSP_PTS_X_INT           0
1127    #define RSP_PTS_Y_INT           2
1128    #define RSP_PTS_Z_INT           4
1129    #define RSP_PTS_W_INT           6
1130    #define RSP_PTS_X_FRAC          8
1131    #define RSP_PTS_Y_FRAC          10
1132    #define RSP_PTS_Z_FRAC          12
1133    #define RSP_PTS_W_FRAC          14
1134    #define RSP_PTS_R_NX            16
1135    #define RSP_PTS_G_NY            17
1136    #define RSP_PTS_B_NZ            18
1137    #define RSP_PTS_A               19
1138    #define RSP_PTS_S               20
1139    #define RSP_PTS_T               22
1140    #define RSP_PTS_XS              24
1141    #define RSP_PTS_YS              26
1142    #define RSP_PTS_ZS              28
1143    #define RSP_PTS_ZSF             30
1144    #define RSP_PTS_INVW_INT        32
1145    #define RSP_PTS_INVW_FRAC       34
1146    #define RSP_PTS_CC              36
1147    #define RSP_PTS_FLAG            38
1148    #define RSP_PTS_LEN             40
1149
1150    RSP_POINTS_OFFSET:
1151            .space  640     /* 40 bytes * 16 points */
1152    #
1153    #
1154    ############### END OF POINTS BUFFER ##############################
1155
1156    ###################################################################
1157    ############### I/O and SCRATCH BUFFERS ###########################
1158    ###################################################################
1159    #
1160    RSP_DLINPUT_OFFSET:
1161            .space  320     /* load display lists in 320 byte chunks */
1162
1163    RSP_INPUT_OFFSET:
1164            .space  256     /* 256 bytes of DMA input buffer */
1165
1166    RSP_YIELD_SAVE_LEN1:    /* what to save when yield occurs */
1167            .print  ""
1168            .symbol RSP_YIELD_SAVE_LEN,     (RSP_YIELD_SAVE_LEN1)+(0x20)
```

```
1169
1170            .dmax   (OS_YIELD_DATA_SIZE) - (31)
1171
1172     #
1173     # WARNING!
1174     # We do not use the correct DMA length in the RSP_INPUT transfer in
1175     # gmain.s... This means we might overwrite the first word of SETUP_TMP.
1176     # That's okay, SETUP_TMP is guarenteed to be trash-able at the time we
1177     # do this transfer.
1178     #
1179     # Don't move this around, or put anything between RSP_INPUT and SETUP_TMP
1180     # If you do, it might get overwritten when you least expect it.
1181     #
1182
1183   RSP_SETUP_TMP_OFFSET:
1184            .space  96      /* 96 bytes of setup scratch mem */
1185
1186   RSP_CLIP_TMP_OFFSET:
1187            .space  160     /* 160 bytes of clip scratch space */
1188
1189   RSP_OUTPUT_OFFSET:
1190            .space  1024    /* 1024 bytes of output buffer */
1191
1192   RSP_SCRATCH_OFFSET:
1193            .space  480     /* 12 points*40 bytes/pt = 480 bytes of clip points */
1194
1195            .symbol RSP_DLINPUT_SIZE8,      (RSP_INPUT_OFFSET)-(RSP_DLINPUT_OFFSET)
1196            .symbol RSP_OUTPUT_SIZE8,       (RSP_SCRATCH_OFFSET)-(RSP_OUTPUT_OFFSET)
1197     #
1198     #
1199     ############## END OF I/O and SCRATCH BUFFERS ####################
1200
1201     ##################################################################
1202     ############## END OF ALLOCATED DMEM #############################
1203     ##################################################################
1204   RSP_END_DMEM:
1205
1206     ##################################################################
1207     ############## TASK HEADER ADDRESS ###############################
1208     ##################################################################
1209
1210            .symbol RSP_TASK_OFFSET,        (0x1000)-(OS_TASK_SIZE)
1211
1212     ##################################################################
1213     ############## CHECK END OF ALLOCATED DMEM #######################
```

```
1214    ##################################################################
1215
1216            .dmax   4096
1217
1218    ##################################################################
1219    ############### PRINT COMPILATION REPORT ###########################
1220    ##################################################################
1221    #
1222    /* During compilation, report an informative message about DMEM allocation: */
1223    #ifdef _LANGUAGE_ASSEMBLY
1224
1225
1226            .print  "-----------------------------------------------------\n"
1227            .print  "FILE: "
1228            .print  __FILE__
1229            .print  "                                          |\n"
1230            .print  "                                                    |\n"
1231
1232            .print  "Total DMEM:    hex bytes                           |\n"
1233            .print  "-------------- ---------                           |\n"
1234            .print  "Initialized  : %3x",RSP_END_INITIALIZE
1235            .print  "                                        |\n"
1236            .print  "In Yield Buf : %3x",  RSP_YIELD_SAVE_LEN
1237            .print  "                                        |\n"
1238            .print  "Allocated    : %3x",RSP_END_DMEM
1239            .print  "                                        |\n"
1240            .print  "                                                    |\n"
1241
1242            .print  "DMEM Map:              loc  l"
1243            .print  "en (hex bytes)              |\n"
1244            .print  "---------------------- ---  ---------------         |\n"
1245
1246
1247            .print  "RSP_PDATA_OFFSET      : %3x  %3x                    |\n",
1248                     RSP_PDATA_OFFSET, (-
1249                     RSP_PDATA_OFFSET  )+
1250                     RSP_STATE_OFFSET
1251            .print  "RSP_STATE_OFFSET      : %3x  %3x                    |\n",
1252                     RSP_STATE_OFFSET, (-
1253                     RSP_STATE_OFFSET  )+
1254                     RSP_SEG_OFFSET
1255            .print  "RSP_SEG_OFFSET        : %3x  %3x                    |\n",
1256                     RSP_SEG_OFFSET, (-
1257                     RSP_SEG_OFFSET  )+
1258                     RSP_LIGHTS_OFFSET
```

```
1259          .print  "RSP_LIGHTS_OFFSET     : %3x  %3x                    |\n",
1260                   RSP_LIGHTS_OFFSET, (-
1261                   RSP_LIGHTS_OFFSET  )+
1262                   RSP_VIEWPORT_OFFSET
1263          .print  "RSP_VIEWPORT_OFFSET   : %3x  %3x                    |\n",
1264                   RSP_VIEWPORT_OFFSET, (-
1265                   RSP_VIEWPORT_OFFSET  )+
1266                   RSP_DLSTACK_OFFSET
1267          .print  "RSP_DLSTACK_OFFSET    : %3x  %3x                    |\n",
1268                   RSP_DLSTACK_OFFSET, (-
1269                   RSP_DLSTACK_OFFSET  )+
1270                   RSP_CURR_MMIX_OFFSET
1271          .print  "RSP_CURR_MMIX_OFFSET  : %3x  %3x                    |\n",
1272                   RSP_CURR_MMIX_OFFSET, (-
1273                   RSP_CURR_MMIX_OFFSET  )+
1274                   RSP_CURR_PMIX_OFFSET
1275          .print  "RSP_CURR_PMIX_OFFSET  : %3x  %3x                    |\n",
1276                   RSP_CURR_PMIX_OFFSET, (-
1277                   RSP_CURR_PMIX_OFFSET  )+
1278                   RSP_CURR_MPMIX_OFFSET
1279          .print  "RSP_CURR_MPMIX_OFFSET : %3x  %3x                    |\n",
1280                   RSP_CURR_MPMIX_OFFSET, (-
1281                   RSP_CURR_MPMIX_OFFSET  )+
1282                   RSP_POINTS_OFFSET
1283          .print  "RSP_POINTS_OFFSET     : %3x  %3x                    |\n",
1284                   RSP_POINTS_OFFSET, (-
1285                   RSP_POINTS_OFFSET  )+
1286                   RSP_DLINPUT_OFFSET
1287          .print  "RSP_DLINPUT_OFFSET    : %3x  %3x                    |\n",
1288                   RSP_DLINPUT_OFFSET, (-
1289                   RSP_DLINPUT_OFFSET  )+
1290                   RSP_INPUT_OFFSET
1291          .print  "RSP_INPUT_OFFSET      : %3x  %3x                    |\n",
1292                   RSP_INPUT_OFFSET, (-
1293                   RSP_INPUT_OFFSET  )+
1294                   RSP_SETUP_TMP_OFFSET
1295          .print  "RSP_SETUP_TMP_OFFSET  : %3x  %3x                    |\n",
1296                   RSP_SETUP_TMP_OFFSET, (-
1297                   RSP_SETUP_TMP_OFFSET  )+
1298                   RSP_CLIP_TMP_OFFSET
1299          .print  "RSP_CLIP_TMP_OFFSET   : %3x  %3x                    |\n",
1300                   RSP_CLIP_TMP_OFFSET, (-
1301                   RSP_CLIP_TMP_OFFSET  )+
1302                   RSP_OUTPUT_OFFSET
1303          .print  "RSP_OUTPUT_OFFSET     : %3x  %3x                    |\n",
```

```
1304                    RSP_OUTPUT_OFFSET, (-
1305                    RSP_OUTPUT_OFFSET  )+
1306                    RSP_SCRATCH_OFFSET
1307            .print  "RSP_SCRATCH_OFFSET    : %3x  %3x                          |\n",
1308                    RSP_SCRATCH_OFFSET, (-
1309                    RSP_SCRATCH_OFFSET  )+
1310                    RSP_END_DMEM
1311
1312            .print  "-------------------------------------------------------\n"
1313
1314        .text
1315
1316    #   undef _DumpDMEMOffset
1317    #else  /* _LANGUAGE_ASSEMBLY */
1318    #endif /* _LANGUAGE_ASSEMBLY */
```

gfx_regs.h

```
 1
 2  /**************************************************************************
 3   *                                                                        *
 4   *               Copyright (C) 1994, Silicon Graphics, Inc.               *
 5   *                                                                        *
 6   *  These coded instructions, statements, and computer programs  contain  *
 7   *  unpublished  proprietary  information of Silicon Graphics, Inc., and  *
 8   *  are protected by Federal copyright  law.  They  may not be disclosed  *
 9   *  to  third  parties  or copied or duplicated in any form, in whole or  *
10   *  in part, without the prior written consent of Silicon Graphics, Inc.  *
11   *                                                                        *
12   **************************************************************************/
13
14  #ifndef _gfx_regs_h_
15  #define _gfx_regs_h_ 1
16
17
18   ############################################################################
19   #
20   # Scalar Register Usage Conventions:
21   #
22   # Starting from $31 growing down, we allocate 'permanent' registers.
23   # These should only be re-used carefully, possibly saved/restored.
24   # (during development we might re-shuffle these so the more crucial
25   # or stable ones remain higher)
26   #
27   # Register $0 is always 0, as per the MIPS R4000 spec.
28   #
29   # Starting from $1 growing up, we use 'temporary' registers which
30   # are free to be used by procedures or small blocks of code. These
31   # registers are not named (or they are .name'd and .unname'd in small
32   # code blocks).
33   #
34   # Function parameters are passed and returned in $1, $2, $3, etc.
35   #
36
37  .name   return,         $31     # MIPS R4000 convention for JAL op
38  .name   return_save,    $30     # used for return addresses not in return
39  .name   rsp_state,      $29     # pointer to RSP state in DMEM
40  .name   dlcount,        $28     # curr counter of DL buffer
41  .name   dinp,           $27     # curr pointer into DL buffer
42  .name   inp,            $26     # DRAM ptr of current DL location (for stack)
43  .name   gfx0,           $25     # holds first word of DL command
```

```
44  .name   gfx1,           $24     # holds second word of DL command
45  .name   outp,           $23
46  .name   in_bufp,        $22
47  .name   zero,           $0
48
49   ###############################################################################
50   #
51   # Vector Register Usage Conventions:
52   #
53   # Similar to scalar registers, named registers start at $v31 and grow down.
54   # These should only be re-used carefully, possibly saved/restored.
55   # (during development we might re-shuffle these so the more crucial
56   # or stable ones remain higher)
57   #
58   # Register $v0 is NOT always 0, like in the MIPS R4000 spec.
59   #
60   # Starting from $v1 growing up, we use 'temporary' registers which
61   # are free to be used by procedures or small blocks of code. These
62   # registers are not named (or they are .name'd and .unname'd in small
63   # code blocks).
64   #
65   # Function parameters are passed and returned in $v1, $v2, $v3, etc.
66   #
67  .name   vconst,         $v31
68  .name   vconst1,        $v30
69
70  #endif /* _gfx_regs_h_ */
```

gboot.s

```
 1
 2  /**************************************************************************
 3   *                                                                        *
 4   *               Copyright (C) 1994, Silicon Graphics, Inc.               *
 5   *                                                                        *
 6   *  These coded instructions, statements, and computer programs  contain  *
 7   *  unpublished  proprietary  information of Silicon Graphics, Inc., and  *
 8   *  are protected by Federal copyright  law.  They  may not be disclosed  *
 9   *  to  third  parties  or copied or duplicated in any form, in whole or  *
10   *  in part, without the prior written consent of Silicon Graphics, Inc.  *
11   *                                                                        *
12   **************************************************************************/
13
14  /*
15   * File:               gboot.s
16   * Creator:            acorn@sgi.com
17   * Create Date:        Mon Oct 23 18:02:33 PDT 1995
18   *
19   */
20
21
22  #include <rsp.h>
23  #include <rcp.h>
24  #include <os.h>
25  #include <sptask.h>
26  #include "mbi.h"
27
28                  .text   TASKBASE        # this is coordinated with rspboot.s
29                  .data   RSP_PDATA_OFFSET
30
31  #define MAIN
32  #include "gdmem.h"
33  #include "gfx_regs.h"
34
35                  .text
36
37          ########################################################################
38          #
39          #  Begin task initialization
40          #
41          #  Register $1 holds the task header address
42          #  jump to init routine
43          #
```

```
44          # (see task.h)
45          #
46
47                  j       doInit          # initialization routine
48          # set state pointer:
49                  addi    rsp_state, zero, RSP_STATE_OFFSET
50    ### JUMP OCCURS here to doInit:
51
52          #
53          #
54          ###########################################################################
```

gclip.s

```
 1
 2  /*
 3   * Copyright 1995, Silicon Graphics, Inc.
 4   * ALL RIGHTS RESERVED
 5   *
 6   * UNPUBLISHED -- Rights reserved under the copyright laws of the United
 7   * States.   Use of a copyright notice is precautionary only and does not
 8   * imply publication or disclosure.
 9   *
10   * U.S. GOVERNMENT RESTRICTED RIGHTS LEGEND:
11   * Use, duplication or disclosure by the Government is subject to restrictions
12   * as set forth in FAR 52.227.19(c)(2) or subparagraph (c)(1)(ii) of the Rights
13   * in Technical Data and Computer Software clause at DFARS 252.227-7013 and/or
14   * in similar or successor clauses in the FAR, or the DOD or NASA FAR
15   * Supplement.  Contractor/manufacturer is Silicon Graphics, Inc.,
16   * 2011 N. Shoreline Blvd. Mountain View, CA 94039-7311.
17   *
18   * THE CONTENT OF THIS WORK CONTAINS CONFIDENTIAL AND PROPRIETARY
19   * INFORMATION OF SILICON GRAPHICS, INC. ANY DUPLICATION, MODIFICATION,
20   * DISTRIBUTION, OR DISCLOSURE IN ANY FORM, IN WHOLE, OR IN PART, IS STRICTLY
21   * PROHIBITED WITHOUT THE PRIOR EXPRESS WRITTEN PERMISSION OF SILICON
22   * GRAPHICS, INC.
23   *
24   */
25
26   ###############################################################################
27   #
28   # Triangle Clip Routine.
29   #
30   # When entering this code we have a points buffer full of points,
31   # and registers r1, r2, r3 point to the three vertices of a triangle.
32   #
33   ###############################################################################
34
35  /* scalar registers: */
36  .name   minp,           $1
37  .name   midp,           $2
38  .name   maxp,           $3
39  .name   tmp,            $8
40
41   ############################## CLIPPING ####################################
42
43   ###################
```

```
44    # CONSTANTS NEEDED:
45    # POINTLIST1: address of 10 free halfwords in DMEM
46    # POINTLIST2: address of 10 free halfwords in DMEM
47    # CLIPMASKS: address of 6 CC masks (halfwords) in DMEM, initialized
48    # CLIP_STATE_TABLE: address of state jumptable (4 halfwords)
49    # CLIP_SELECT: address of clip select table, 1 halfword for each plane (6 hw)
50    # FREEPOINTS: address of 12 empty point structures (40 * 12 bytes)
51
52    #define POINTLIST1      (RSP_CLIP_TMP_OFFSET)
53    #define POINTLIST2      (POINTLIST1+20)
54    #define POINTSWAP       (POINTLIST1^POINTLIST2)
55    #define FREEPOINTS      ((RSP_SCRATCH_OFFSET+7)&0xfff8)
56
57    #define C_OUT           0
58    #define C_IN            2
59    #define C_FIRSTOUT      4
60    #define C_FIRSTIN       6
61
62    .name   oldlist,        $5      # old list of vertices
63    .name   plane,          $6      # plane (0-10 by 2's) we are currently clipping
64    .name   ccor,           $11     # OR of all points' clip codes
65    .name   ccand,          $12     # AND of all points' clip codes
66    .name   voutp,          $7      # where the newly generated points will go
67                                    #
68    .unname return_save             # $30
69    .name   newlist,        $30     # new list of vertices
70
71                  .ent    doClip
72
73    doClip:
74     # ############################ FIND 2 EDGES TO CLIP ########################
75     #      bne     ccand, zero, clipKill                   # trivial reject?
76            sh      maxp, POINTLIST1(zero)                  # put point ptrs in list
77            sh      midp, (POINTLIST1+2)(zero)              # ending with 0
78            sh      minp, (POINTLIST1+4)(zero)              #
79            sh      zero, (POINTLIST1+6)(zero)              #
80                                                            #-----------------------
81    .unname maxp                                            # reuse until setup time
82                                                            #-----------------------
83            ori     voutp,$0,(FREEPOINTS - RSP_PTS_LEN)     # free scratch points
84            ori     newlist,$0,POINTLIST1                   # pointer to new list
85            ori     plane,$0,12                             # start at plane 6
86                                                            #-----------------------
87    .unname ccor                                            # done with this
88    .name   clipmask,       $11                             # bit in cc which we are
```

```
 89                                                      # clipping against
 90                                                      #----------------------
 91  nextClip:                                           #
 92          or      oldlist,newlist,newlist             # swap oldlist & newlist
 93          xori    newlist,newlist,POINTSWAP           #
 94                                                      #----------------------
 95  nextPlane:                                          #
 96          beq     plane,$0,drawItAll                  # did all planes?
 97                                                      #----------------------
 98  findClipPlane:                                      #
 99          lh      clipmask,(CLIPMASKS - 2)(plane)     # cur clipplane mask bit
100                                                      #
101   ### JUMP OCCURS to drawItAll: IF all planes have been clipped
102                                                      #
103          addi    plane,plane,-2                      # next clip plane
104
105  .unname ccand                                       #
106  .unname midp                                        #
107
108  .name   prevpoint,      $20     # point preceding current point
109  .name   pointptr,       $10     # point to current point
110
111  .name   inoutcheck,     $18     # =clipmask if look for in, =0 if look for out
112  .name   pointhdl,       $2      # point to current point in point list
113  .name   newhdl,         $14     # build new list of points
114  .name   state,          $17     # looking for C_OUT, C_FIRSTIN, or C_FIRSTOUT
115
116  clipToPlane:                                #
117          ori     state,$0,C_OUT              # first look for a clipped vtx
118          or      inoutcheck,$0,$0            # looking for inside
119  foundIn:                                    #
120          ori     pointhdl,oldlist,0          # start @ 1st pt to find in pt
121  foundOut:                                   #
122          j       checkNextPoint              # find next appropriate point
123          addi    newhdl,newlist,2            # start @ 1st new point
124   ### JUMP OCCURS to checkNextPoint
125                                              #
126                                              #
127  findInOutLoop:                              #
128          and     tmp,tmp,clipmask            # 0 if plane doesn't clip point
129          beq     tmp,inoutcheck,nextState    # increment state if cross plane
130          addi    pointhdl,pointhdl,2         # prepare to check next point
131   ### JUMP OCCURS to nextState IF plane boundary was crossed
132  checkNextPoint:                             #
133          or      prevpoint,pointptr,$0       # current point becomes previous
```

```
134         sh      pointptr,0(newhdl)              # move point to new list
135         addi    newhdl,newhdl,2                 # next element of new list
136 checkFirstPoint:                                #
137         lh      pointptr,0(pointhdl)            # pointer to point
138         bne     pointptr,$0,findInOutLoop       # went throug whole list?
139         lh      tmp,RSP_PTS_CC(pointptr)        # point's clip code
140  ### JUMP OCCURS to findInOutLoop IF didn't go through whole list yet
141 endedList:                                       #
142         addi    tmp,state,(-C_IN)               # no in or no out points?
143         bgtz    tmp,checkFirstPoint             # continue if both in & out
144         ori     pointhdl,oldlist,0              # handle to 1st point
145  ### JUMP OCCURS to checkFirstPoint IF there are points inside & outside plane
146         beq     tmp,$0,nextPlane                # not clipped by this plane
147         nop                                     #
148  ### JUMP OCCURS to return IF triangle is completely rejected
149         j       clipKill                        # plane rejected triangle
150                                                 #
151
152 nextState:                                      #
153         xor     inoutcheck,inoutcheck,clipmask  # found out(in); now find in(out
154         lh      tmp,CLIP_STATE_TABLE(state)     # get state routine
155         addi    state,state,2                   # increment to next state
156         jr      tmp                             # jump to state routine
157         lh      tmp,NEXTCLIP($0)                # nxt plane loop address
158  ### JUMP OCCURS to state routine
159
160
161  # ############################ PERFORM THE CLIP ############################
162  # clip between lastout and firstin
163  # also clip between prevpoint and pointptr
164
165
166 .name   ini,            $v4     # non clipped points vector
167 .name   inf,            $v5
168 .name   outi,           $v9     # clipped points vector
169 .name   outf,           $v10
170 .name   topi,           $v11    # numerator of n calculation
171 .name   topf,           $v12
172 .name   boti,           $v27    # denominator of n calculation
173 .name   botf,           $v26
174 .name   selp,           $v1     # select which plane we are clipping against
175 .name   seln,           $v0     # negative of selp
176 .name   nf,             $v15    # parameter of clip (0.16)
177 .name   negnf,          $v8     # 1 - parameter of clip (0.16)
178 .name   vtmpi,         $v29    # temporary vector
```

```
179  .name   vtmpf,          $v28    # temporary vector
180  .name   ptptrhold,      $v13    # remembers pointptr
181
182  foundFirstIn:                                       # now clip can occur
183                                                      #
184          mtc2    pointptr,ptptrhold[0]               # swap pointptr & prevpt
185          or      pointptr,prevpoint,$0               #
186          mfc2    prevpoint,ptptrhold[0]              #
187          ori     newhdl,newlist,0                    # start @ 1st new point
188          lh      tmp,FOUND_OUT($0)                   # Return to foundOut:
189                                                      #
190                                                      #
191  foundFirstOut:                                      # now clip can occur
192                                                      #
193          sh      tmp,RETURNJUMP($0)                  # save return address
194          addi    voutp,voutp,RSP_PTS_LEN             # new point addr
195          sh      voutp,0(newhdl)                     #     ...into list
196          sh      $0,2(newhdl)                        # zero @ end of new list
197                                                      #
198          ldv     outi[0],RSP_PTS_X_INT(pointptr)     # get XYZW of in and out
199          ldv     outf[0],RSP_PTS_X_FRAC(pointptr)    # vertices
200          ldv     ini[0],RSP_PTS_X_INT(prevpoint)     #
201          ldv     inf[0],RSP_PTS_X_FRAC(prevpoint)    #
202                                                  #
203          sll     tmp,plane,2                     # plane to select
204          ldv     selp[0],(CLIP_SELECT)(tmp)      # select vector
205          vmudh   seln,selp,vconst[3]             # negative select vector
206                                                  # ----------------------------
207                                                  # CALCULATE n (comments assume
208                                                  # positive x plane clipping,
209                                                  # so selp=[1 0 0 -1])
210                                                  # xyz= in point; XYZ=out point
211                                                  #----------------------------
212          vmudn   topf,inf,selp                   # top=[x 0 0 -w]
213          vmadh   topi,ini,selp                   #
214          vmadn   topf,vconst,vconst[0]           #
215                                                  #----------------------------
216          vmadn   vtmpf,outf,seln                 # bot= [-X 0 0 W] + top =
217          vmadh   vtmpi,outi,seln                 #  [(x-X) 0 0 (W-w)]
218          vmadn   vtmpf,vconst,vconst[0]          #
219                                                  #----------------------------
220          vaddc   botf,vtmpf,vtmpf[0q]            # add all componants together
221          vadd    boti,vtmpi,vtmpi[0q]            # (2 of X,Y,Z are zero)
222          vaddc   vtmpf,botf,botf[1h]             # ... into element 3
223                                                  #
```

```
224         vadd    vtmpi,boti,boti[1h]         # result = [? ? ? (x-X)-(w-W)]
225                                             #
226                                             #-----------------------------
227                                             #
228 .name   wsclf,  $v3                         # calculate scalefactor to
229 .name   wscli,  $v7                         # improve reciprical precision
230                                             #
231         mfc2    tmp,vtmpi[6]                # sign of bot
232                                             #
233         vrcph   wscli[3], vtmpi[3]          # scalefactor wscl
234         vrcpl   wsclf[3], vtmpf[3]          # for greater precision in
235         vrcph   wscli[3], vconst[0]         # reciprical
236                                             #
237                                             #
238         vmudn   wsclf, wsclf, vconst[2]     # *2 to complete reciprical
239         bgez    tmp, clampWscl              # branch if non negative
240         vmadh   wscli, wscli, vconst[2]     # *2 to complete reciprical
241                                             #
242  ### BRANCH OCCURS TO clampWscl: IF wscl>=0  #
243                                             #-----------------------------
244         vmudn   wsclf, wsclf, vconst[3]     # *-1 for absolute value
245         vmadh   wscli, wscli, vconst[3]     # *-1 for absolute value
246 clampWscl:                                  #-----------------------------
247         veq     wscli,wscli,vconst[0]       # if   wscl >= 1.0
248         vmrg    wsclf,wsclf,vconst[3]       # then wsclf =  0.99999
249                                             # (only wsclf used; not wscl)
250                                             #-----------------------------
251         vmudl   vtmpf, vtmpf, wsclf[3]      # use scalefactor wsclf for 1/x
252         vmadm   vtmpi, vtmpi, wsclf[3]      # (improves 1/x precicion)
253         jal     NewtonDiv                   #
254         vmadn   vtmpf, vconst, vconst[0]    #
255                                             #-----------------------------
256  ### JUMP OCCURS to subroutine NewtonDiv:    # bot= 1/vtmp
257                                             #-----------------------------
258         vaddc   vtmpf,topf,topf[0q]         # add all componants together
259         vadd    vtmpi,topi,topi[0q]         # (2 of X,Y,Z are zero)
260         vaddc   topf,vtmpf,vtmpf[1h]        # ... into element 3
261         vadd    topi,vtmpi,vtmpi[1h]        # result = [? ? ? (x-w)]
262                                             #-----------------------------
263         vmudl   nf,topf,botf                # n= top * 1/bot
264         vmadm   nf,topi,botf                #
265         vmadn   nf,topf,boti                #
266         vmadh   negnf,topi,boti             #
267                                             #-----------------------------
268         vmudl   vtmpf, vconst, vconst[5]    # add 0.0003fffc (~0.0004)
```

```
269                                             # to ensure outside of frustum
270                                             #
271         vmadl   nf, nf, wsclf[3]            # remove scalefcator wsclf
272         vmadm   negnf, negnf, wsclf[3]      #
273         vmadn   nf, vconst, vconst[0]       #
274                                             #-----------------------------
275   .unname wscli                             # done with scalefactor
276   .unname wsclf                             #
277                                             #-----------------------------
278         veq     negnf,negnf,vconst[0]       # if number >=1
279         vmrg    nf,nf,vconst[3]             # then number = 0.99999
280
281         vne     nf,nf,vconst[0]             # if number =0
282         vmrg    nf,nf,vconst[1]             # then number = 0.00001
283                                             #---------------------------
284         vnxor   negnf,nf,vconst[0]          # negnf[0h] = 1-n (0.16)
285         vaddc   negnf,negnf,vconst[1]       # ones complement + 1 of frac
286         vadd    vtmpi,vtmpi,vtmpi           # VCO=0
287
288
289    ############ SET UP REGISTERS TO MATCH screenCalc: in gvtx.s ###############
290    #-------------------------------------------------------------------------
291    # THESE NEED TO BE LOADED WITH APPROPRIATE DATA TO SAVE TO VERTICES
292    #.................................#
293   .unname boti                        # bot   now called   persp12
294   .unname botf                        #
295   .name   invW12f,        $v26        #
296   .name   invW12i,        $v27        #
297   .name   st12,           $v18        # S and T
298    #.................................#
299   .name   clr1,           $15         # RGBA   new point 1
300    #.................................#
301   .name   clr2,           $16         # NOT TOUCHED !!!!
302                                       #
303    #-------------------------------------------------------------------------
304    # THESE NEED TO BE SET UP FOR THE VIX ROUTINE TO WORK
305    #.................................#
306   .unname selp                        #
307   .name   vptrans,        $v1         # loaded by getScaleTrans subroutine
308    #.................................#
309   .unname seln                        #
310   .name   vpscale,        $v0         # loaded by getScaleTrans subroutine
311    #.................................#
312   .name   i,              $9          # indicate 1 vertex to process
313                                       # (the clipped vertex)
```

```
314    #-------------------------------------------------------------------
315    # THESE NEED TO BE AVAILABLE          #
316    #.....................................#
317   .unname newhdl                         #
318   .name   flg2,           $14            #
319    #.....................................#
320   .unname vtmpi                          #
321   .name   vout12i,        $v29           #
322    #.....................................#
323   .unname vtmpf                          #
324   .name   vout12f,        $v28           #
325    #.....................................#
326   .name   flg1,           $13            #
327   .name   vtmp,           $v3            #
328   .name   scrn12f,        $v7            #
329   .name   scrn12i,        $v6            #
330   .name   vin12,          $v2            #
331    #.....................................#
332   .unname tmp                            # No change here
333   .name   tmp,            $8             #
334    #.....................................#
335   .name   vnewton1,       $v22           # used for newton raphson subroutine
336   .name   vnewton2,       $v23           #
337   .name   vnewton3,       $v24           #
338   .name   vnewton4,       $v25           #
339    #.....................................
340                                           # CALCULATE new values for
341                                           # coordinates & attributes
342                                           #--------------------------
343          vmudl   vout12f,inf,negnf[3h]    # in * (1-n)
344          vmadm   vout12i,ini,negnf[3h]    #
345          vmadl   vout12f,outf,nf[3h]      # ... + (out * n)
346          vmadm   vout12i,outi,nf[3h]      #
347          vmadn   vout12f,vconst,vconst[0] # = new xyzw
348                                           #--------------------------
349    #.....................................#
350   .unname inf                            #
351   .name   persp12f,       $v5            #
352    #.....................................#
353   .unname ini                            #
354   .name   persp12i,       $v4            #
355    #.....................................#
356                                           #--------------------------
357          luv     topf[0],RSP_PTS_R_NX(pointptr)  # topf[0-6] = RGBAout
358          luv     topi[0],RSP_PTS_R_NX(prevpoint) # topi[0-6] = RGBAin
```

```
359                                             #--------------------------
360         llv     topf[8],RSP_PTS_S(pointptr) # topf[8,10] = STout
361         llv     topi[8],RSP_PTS_S(prevpoint) # topi[8,10] = STin
362                                             #--------------------------
363         vmudm   st12,topf,nf[3]             # st12 = new ST & RGBA
364         vmadm   st12,topi,negnf[3]          #
365                                             #--------------------------
366         suv     st12[0],0(voutp)            # temp store new RGBA
367         sdv     st12[8],8(voutp)            #
368         ldv     st12[0],8(voutp)            #
369         jal     getScaleTrans               # get screen scale & trans
370         lw      clr1,0(voutp)               # new RGBA 1
371                                             #--------------------------
372   ### BRANCH OCCURS to subroutine getScaleTrans: # load regs for screenCalc
373                                             #--------------------------
374         mfc2    pointptr,ptptrhold[0]       # restore pointptr
375         j       screenCalc                  # calc screen coord & store
376         ori     i,$0,1                      # setup for vtx routine
377                                             #
378   ### JUMP OCCURS to screenCalc: subroutine; return to nextPlane:
379                                             #--------------------------
380   .unname vout12i                           #
381   .unname vout12f                           #
382   .unname flg1                              #
383   .unname flg2                              #
384   .unname clr1                              #
385   .unname clr2                              #
386   .unname st12                              #
387   .unname vptrans                           #
388   .unname vpscale                           #
389   .unname i                                 #
390   .unname vtmp                              #
391   .unname invW12f                           #
392   .unname invW12i                           #
393   .unname scrn12f                           #
394   .unname scrn12i                           #
395   .unname vin12                             #
396   .unname vnewton1                          #
397   .unname vnewton2                          #
398   .unname vnewton3                          #
399   .unname vnewton4                          #
400                                             #
401   .unname voutp                             #
402   .unname persp12i                          #
403   .unname persp12f                          #
```

```
404   .unname outi                                  #
405   .unname outf                                  #
406   .unname topi                                  #
407   .unname topf                                  #
408   .unname nf                                    #
409   .unname negnf                                 #
410                                                 #
411   .unname pointhdl                              #
412   .unname prevpoint                             #
413   .unname pointptr                              #
414   .unname plane                                 #
415   .unname clipmask                              #
416   .unname state                                 #
417   .unname inoutcheck                            #
418   .unname ptptrhold                             #
419                                                 #
420                                                 #
421   .name   maxp,           $3                    #
422   .name   midp,           $2                    #
423                                                 #
424   .unname newlist                               # $30
425   .name   return_save,    $30                   #
426                                                 #
427                                                 #--------------------------
428                                                 # DRAW THE CLIPPED POLYGON
429                                                 #--------------------------
430   drawItAll:                                    #
431           lh      tmp,0(oldlist)                # 1st point (anchor)
432           sh      tmp,ANCHOR($0)                # remember it
433           sh      oldlist,RETURNJUMP($0)        # pointer to list of points
434           lh      return_save,CLIPDRAWLOOP($0)  # loop address
435                                                 #--------------------------
436   clipDrawLoop:                                 #
437           lh      tmp,RETURNJUMP($0)            # pointer to list of points
438           lh      maxp,ANCHOR($0)               # start with anchor point
439           lh      midp,2(tmp)                   #  ...then next point
440           lh      minp,4(tmp)                   #  ...then point following
441           addi    tmp,tmp,2                     # next time use next 2 points
442           bne     minp,$0,beginSetup            # setup & draw if not last point
443           sh      tmp,RETURNJUMP($0)            # store next point handle
444    ### JUMP OCCURS to beginSetup: if not all triangles have been processed
445                                                 #
446   clipKill:                                     #
447           j       GfxDone                       # return
448                                       #
```

```
449                                     # NOTE DELAY SLOT!!!!
450                                     # the first instruction of gsetup.s
451                                     # (which follows this) is a load
452                                     # halfword into a register I do not
453                                     # care about.
454                                     #
455                         .end doClip     #
456     .unname oldlist                 #
457     .unname tmp                     #
458
459      # ############################ END CLIPPING ################################
460
461     .unname minp                    #
462     .unname midp                    #
463     .unname maxp                    #
464                                     #
465                         .align  8       # even out the end for DMA's
```

gdma.s

```
 1
 2  /**************************************************************************
 3   *                                                                        *
 4   *               Copyright (C) 1994, Silicon Graphics, Inc.               *
 5   *                                                                        *
 6   *  These coded instructions, statements, and computer programs  contain  *
 7   *  unpublished  proprietary  information of Silicon Graphics, Inc., and  *
 8   *  are protected by Federal copyright  law.  They  may not be disclosed  *
 9   *  to  third  parties  or copied or duplicated in any form, in whole or  *
10   *  in part, without the prior written consent of Silicon Graphics, Inc.  *
11   *                                                                        *
12   **************************************************************************/
13
14  /*
15   * File:               gdma.s
16   * Creator:            hsa@sgi.com
17   * Create Date:        Fri Jun 24 13:54:58 PDT 1994
18   *
19   * This file holds the top-level of the DMA command processing, and
20   * related routines.
21   *
22   */
23
24
25
26   ###############################################################################
27   #
28   # The following code processes the DMA type display list commands.
29   # Registers on input:
30   #      gfx0    - first word of display list command
31   #      gfx1    - second word of display list command
32   #      in_bufp - where the data has been read into
33   #      dlnp    - points to the *next* display list command
34   #
35
36   ### IMPORTANT!!!: The first instruction here is a branch delay slot
37   ### from grdp.s
38
39                 .ent    doDMA
40
41  doDMA:
42          # $2 is shifted in the delay slot of the branch that
43          # brought us here...
```

```
44
45          # 'switch' to correct DMA command:
46                  andi    $2, $2, 0x1fe           # shifted up 1 for offset
47                  lh      $2, DMA_JMP_OFFSET($2)
48
49          # 'len' is the same for all commands. But we don't need it here?
50
51          # Not double-buffered yet, so we have to wait here for the data...
52                  jal     DMAwait
53                  lbu     $1, (0-7)(dinp) # pick off first field
54
55                  jr      $2
56          # this is the same for some commands, so save code space
57          # and do it here in the delay slot.
58                  andi    $6, $1, 0x0f    # pick off 'v0' field
59
60                  .end    doDMA
61   #
62   #
63   ###########################################################################
64
65
66   ###########################################################################
67   #
68   # Process the G_MTX command.
69   #
70   #      $1 holds param
71   #      in_bufp holds pointer to data
72   #
73
74  #include "gmtx.s"       /* big routine, in a separate file */
75
76   #
77   #
78   #
79   ###########################################################################
80
81
82   ###########################################################################
83   #
84   #  MOVEMEM handles the VIEWPORT and LIGHT input.
85   #
86  .name outptr,   $5
87  .name target,   $1
88  .name buff,     $v0
```

```
89
90                  .ent    case_G_MOVEMEM
91
92    case_G_MOVEMEM:
93          # $1 (target) holds n (index to target address*2)
94          #             (hi bit set for causing light recomputation)
95          # in_bufp holds pointer to data
96          lqv     buff[0], 0(in_bufp)                     # get 4 words
97          lh      outptr,(MOVEMEM_TBL - 0x80)(target)     # get target address
98          j       GfxDone                                 # return
99          sqv     buff[0], 0(outptr)                      # store 4 words
100
101                 .end    case_G_MOVEMEM
102
103   .unname outptr
104   .unname target
105   .unname buff
106    #
107    #
108    #
109    ###############################################################################
110
111
112    ###############################################################################
113    #
114    # Transform, project, & shade as we load into the points buffer.
115    #
116    # $1 holds n and v0
117    # in_bufp holds pointer to data
118    #
119
120   .name   tmp,    $8
121
122     case_G_VTX:
123          lh      tmp,GFXDONE(zero)
124          sh      tmp,RETURNJUMP(zero)
125   .unname tmp
126
127   #include "gvtx.s"
128
129   .name   tmp,    $8
130   xfm_done:
131          # NOTE! the first statement here is a branch delay slot
132          # from the loop in gvtx.s
133
```

```
134         lh      tmp,RETURNJUMP($0)
135         jr      tmp
136         nop
137
138   .unname tmp
139
140    #
141    #
142    #
143    ###############################################################################
144
145
146    ###############################################################################
147    #
148   .name stack_sz, $2
149   .name stack_p,  $3
150   .name tmp,      $4
151
152                 .ent    case_G_DL
153     case_G_DL:
154         # $1 holds param
155         # in_bufp holds pointer to data
156
157         # check param to see if we need to push
158                 bgtz    $1, noPush
159                 # note delay slot...
160
161         # if push:
162                 lb      stack_sz, RSP_STATE_DL_N(rsp_state)
163
164         # check to see if stack is full (10 entries max)
165                 addi    tmp, stack_sz, -36
166                 bgtz    tmp, GfxDone
167                 # note delay slot
168
169                 addi    stack_p, stack_sz, RSP_DLSTACK_OFFSET
170                 addi    stack_sz, stack_sz, 4
171                 sw      inp,     0(stack_p)     # pointer of DL
172                 sb      stack_sz, RSP_STATE_DL_N(rsp_state)
173
174         # set some state, prepare to read next list
175         # set inp to the new pointer, jump to load...
176
177         noPush:
178                 jal     AddrFixup
```

```
179                 add     $19, gfx1, zero         # delay slot
180                 add     inp, $19, zero
181                 j       GfxDone
182                 addi    dlcount, zero, 0        # delay slot
183
184                 .end    case_G_DL
185   .unname stack_sz
186   .unname stack_p
187   .unname tmp
188    #
189    #
190    #
191    ###############################################################################
```

gdone.s

```
 1
 2  /*
 3   * Copyright 1995, Silicon Graphics, Inc.
 4   * ALL RIGHTS RESERVED
 5   *
 6   * UNPUBLISHED -- Rights reserved under the copyright laws of the United
 7   * States.   Use of a copyright notice is precautionary only and does not
 8   * imply publication or disclosure.
 9   *
10   * U.S. GOVERNMENT RESTRICTED RIGHTS LEGEND:
11   * Use, duplication or disclosure by the Government is subject to restrictions
12   * as set forth in FAR 52.227.19(c)(2) or subparagraph (c)(1)(ii) of the Rights
13   * in Technical Data and Computer Software clause at DFARS 252.227-7013 and/or
14   * in similar or successor clauses in the FAR, or the DOD or NASA FAR
15   * Supplement.  Contractor/manufacturer is Silicon Graphics, Inc.,
16   * 2011 N. Shoreline Blvd. Mountain View, CA 94039-7311.
17   *
18   * THE CONTENT OF THIS WORK CONTAINS CONFIDENTIAL AND PROPRIETARY
19   * INFORMATION OF SILICON GRAPHICS, INC. ANY DUPLICATION, MODIFICATION,
20   * DISTRIBUTION, OR DISCLOSURE IN ANY FORM, IN WHOLE, OR IN PART, IS STRICTLY
21   * PROHIBITED WITHOUT THE PRIOR EXPRESS WRITTEN PERMISSION OF SILICON
22   * GRAPHICS, INC.
23   *
24   */
25
26   ############################################################################
27   #
28   # This code includes yield and taskdone code
29   #
30   ############################################################################
31
32  #ifdef FASTLIGHT3D
33  #include <rsp.h>
34  #include "mbi.h"
35
36                  .text   startClip
37
38  #include "gdmem.h"
39  #include "gfx_regs.h"
40  #endif /* FASTLIGHT3D */
41
42
43   ############################################################################
```

```
44   ###############################################################################
45   #
46   # JUMP TO YIELD (MUST BE EXACILY 2 INSTRUCTIONS)
47   #
48  #ifdef FASTLIGHT3D
49                  j       RSPYieldOver
50                  nop
51  #endif /* FASTLIGHT3D */
52
53
54   ###############################################################################
55   ###############################################################################
56   #
57   # TASKDONE
58   #
59
60
61  .name   dramp, $19
62  .name   dmemp, $20
63  .name   dmasz, $18
64  .name   iswrite, $17
65
66  #ifndef FASTLIGHT3D
67  TaskDone:       # all done, return to executive:
68  #endif /* FASTLIGHT3D */
69          #
70          # If we sent output to the DRAM, update the counter
71          # in DRAM
72          #
73  #ifdef OUTPUT_DRAM
74                  addi    dmemp, rsp_state, RSP_STATE_DRAM_OUT_LEN
75                  lw      dramp, RSP_STATE_DRAM_OUT_LENP(rsp_state)
76                  addi    iswrite, zero, 1
77                  jal     DMAproc
78                  addi    dmasz, zero, 7  # delay slot
79  #else
80                  nop
81  #endif
82  .unname dramp
83  .unname dmemp
84  .unname dmasz
85  .unname iswrite
86
87  .name yield,    $2
88
```

```
 89  TaskHalt:
 90                  jal     DMAwait
 91                  ori     yield, zero, SP_SET_TASKDONE    # delay slot
 92
 93                  mtc0    yield, SP_STATUS                # DONE, not a BP
 94                  break                                   # tell cpu we are done
 95                  nop                                     # and HALT
 96  .unname yield
 97
 98   ###########################################################################
 99   ###########################################################################
100   #
101   # YIELD
102   #
103  #ifdef FASTLIGHT3D
104
105  #define YIELD_STOP
106  #include "gyield.s"
107  #undef YIELD_STOP
108
109                 nop
110                 nop
111  #endif /* FASTLIGHT3D */
112
113
```

gflight.s

```
 1
 2  /*
 3   * Copyright 1995, Silicon Graphics, Inc.
 4   * ALL RIGHTS RESERVED
 5   *
 6   * UNPUBLISHED -- Rights reserved under the copyright laws of the United
 7   * States.   Use of a copyright notice is precautionary only and does not
 8   * imply publication or disclosure.
 9   *
10   * U.S. GOVERNMENT RESTRICTED RIGHTS LEGEND:
11   * Use, duplication or disclosure by the Government is subject to restrictions
12   * as set forth in FAR 52.227.19(c)(2) or subparagraph (c)(1)(ii) of the Rights
13   * in Technical Data and Computer Software clause at DFARS 252.227-7013 and/or
14   * in similar or successor clauses in the FAR, or the DOD or NASA FAR
15   * Supplement.  Contractor/manufacturer is Silicon Graphics, Inc.,
16   * 2011 N. Shoreline Blvd. Mountain View, CA 94039-7311.
17   *
18   * THE CONTENT OF THIS WORK CONTAINS CONFIDENTIAL AND PROPRIETARY
19   * INFORMATION OF SILICON GRAPHICS, INC. ANY DUPLICATION, MODIFICATION,
20   * DISTRIBUTION, OR DISCLOSURE IN ANY FORM, IN WHOLE, OR IN PART, IS STRICTLY
21   * PROHIBITED WITHOUT THE PRIOR EXPRESS WRITTEN PERMISSION OF SILICON
22   * GRAPHICS, INC.
23   *
24   */
25
26   ##############################################################################
27   #
28   # Vertex Lighting Routine
29   #
30   # When entering this code we have
31   #
32   ##############################################################################
33
34   #### REGISTERS SHARED WITH VERTEX ROUTINE
35  .name   gmode,          $3
36  .name   clr1,           $15
37  .name   clr2,           $16
38  .name   tmp,            $8
39  .name   vtmp,           $v3
40  .name   scratch,        $7
41  .name   mtx0,           $v8
42  .name   mtx1,           $v9
43  .name   mtx2,           $v10
```

```
44  .name   mtx3,           $v11
45  .name   mtf0,           $v12
46  .name   mtf1,           $v13
47  .name   mtf2,           $v14
48  .name   mtf3,           $v15
49  .name   st12,           $v18
50
51
52   #### REGISTERS UNIQUE TO FAST LIGHT ROUTINE
53  .name   norm,           $v4
54  .name   lightdir,       $v5
55  .name   ndotl,          $v6
56  .name   color,          $v7
57  .name   sumcol,         $v27
58  .name   lptr,           $1
59   #.name stdot,          $v19
60  .name   stdot,          $v20
61
62  doLight:
63   #lightBegin:
64          lw      lptr,RSP_L_NUM(zero)            # pointer to first light
65
66          sw      clr1,0(scratch)                 # store normal 1
67          sw      clr2,4(scratch)                 # store normal 2
68          bltz    lptr,xformLights                # transform lights if needed
69          lpv     norm[0],0(scratch)              # get normal
70   ### JUMP OCCURS to xformLights if lights need transforming.
71
72          luv     color[0],(RSP_L_COL+RSP_L_0 - RSP_L_LEN)(lptr)  # ambient
73          vxor    sumcol,sumcol,sumcol                    # zero color sum
74
75  lightLoop:
76          vge     color,color,vconst[0]           # clamp color to 0
77          lpv     lightdir[0],(RSP_L_DIR+RSP_L_BASE)(lptr) # dir
78          vadd    sumcol,sumcol,color             # accumulate color
79          luv     color[0],(RSP_L_COL+RSP_L_BASE)(lptr)    # col
80
81          vor     stdot,ndotl,vconst[0]           # save previous dot product
82
83          vmulf   ndotl,norm,lightdir             # n dot l
84          vadd    vtmp,ndotl,ndotl[1q]            # sum ndotl
85          vadd    ndotl,vtmp,ndotl[2h]            # sum ndotl into 1st element
86          vmulf   color,color,ndotl[0h]           # color *= ndotl
87                                                  #
88          bgtz    lptr,lightLoop                  # loop if more lights
```

```
 89         addi    lptr,lptr,(-1*RSP_L_LEN)        # next light
 90   ### LOOP OCCURS to lightLoop: if more lights are on
 91                                                  #
 92         suv     sumcol[0],0(scratch)             # store color
 93
 94         andi    tmp,gmode,G_TEXTURE_GEN_H        # texture coord generation on?
 95         sb      clr1,3(scratch)                  # insert alpha 1
 96         sb      clr2,7(scratch)                  # insert alpha 2
 97         lw      clr1,0(scratch)                  # get color 1
 98         beq     tmp,zero,lightReturn             # if no tex gen, return
 99         lw      clr2,4(scratch)                  # get color 2
100   ### BRANCH OCCURS to retun to lightReturn: if no texture coord generation.
101
102   #
103   # TEXTURE COORDINATE GENERATION / REFLECTION MAPPING
104   #
105
106   #
107   # at this point ndotl = T calculation; stdot = S calculation (both s.15)
108   # T * [0, 0.5, 0] is in vector accumulator
109   #
110
111         andi    tmp,gmode,G_TEXTURE_GEN_LINEAR_H # linear texture coords on?
112         lpv     color[0],(RSP_SELECT_S-RSP_STATE_OFFSET)(rsp_state) # select s
113         ldv     ndotl[0],DOLIGHT(zero)           # constants
114         vmadn   stdot,color,stdot[0h]            # add S + T-> el[0h][1h]
115         beq     tmp,zero,texGenRet               # skip linear if turned off
116         vmadm   st12,vconst,vconst[0]            # add S + T-> el[0h][1h]
117   ### BRANCH OCCURS to texGenRet if linear texture generation is off.
118
119   #
120   # at this point st12 has
121   #   vtx 1's generated T in element 0
122   #   vtx 1's generated S in element 1
123   #   vtx 2's generated T in element 4
124   #   vtx 2's generated S in element 5
125   #
126   # all numbers range from -0x4000 to +0x4000
127   #
128
129
130   ####
131   #### S,T=ARCCOS(S,T)
132   ####
133
```

```
134         vmulf   color,st12,st12             # st^2
135         vmulf   color,color,st12            # st^3
136         vmulf   stdot,color,ndotl[1]        # st^3 * 1
137         vmacf   stdot,color,ndotl[3]        # st^3 + st^3 * (0x3A0B)
138         vmacf   st12,st12,ndotl[2]          # ...+ st * (0x517d)
139
140
141 texGenRet:
142
143         j       lightReturn                 # return when finished
144         vadd    st12,st12,vconst[4]         # ...+ (0x4000)
145  ### JUMP OCCURS TO return
146
147  ####
148  #### S,T=ARCCOS(S,T)
149  ####
150
151
152 .unname ndotl
153 .unname vtmp
154 .unname norm
155 .unname color
156 .unname stdot
157 .unname st12
158
159 .name   modnormi,       $v3
160 .name   modnormf,       $v6
161 .name   vtmpi,          $v4
162 .name   vtmpf,          $v7
163
164
165 xformLights:
166         andi    lptr,lptr,0xfff             # clear "untransformed" flag
167         sw      lptr,RSP_L_NUM(zero)        #
168
169         jal     getMatrix3                  # get the M matrix
170         addi    tmp,zero,RSP_CURR_MMTX_OFFSET   # address of Model Matrix
171  ### JUMP OCCURS to getMatrix3 to load the 3x3 model matrix
172
173         ori     tmp, zero, RSP_SCRATCH_OFFSET
174         stv     mtx0[2],16(tmp)         # transpose matrix
175         stv     mtx0[4],32(tmp)         #
176         stv     mtx0[12],48(tmp)        #
177         stv     mtx0[14],64(tmp)        #
178         ltv     mtx0[14],16(tmp)        #
```

```
179         ltv     mtx0[12],32(tmp)        #
180         ltv     mtx0[4],48(tmp)         #
181         ltv     mtx0[2],64(tmp)         #
182
183         sdv     mtf0[8],16(tmp)         # move float part from 8..15 to 0..7
184         sdv     mtf1[8],32(tmp)         #
185         sdv     mtf2[8],48(tmp)         #
186         ldv     mtf0[0],16(tmp)         #
187         ldv     mtf1[0],32(tmp)         #
188         ldv     mtf2[0],48(tmp)         #
189
190
191 litXformLoop:
192         lpv     lightdir[0],(RSP_L_DIR_VIEW+RSP_L_BASE)(lptr) # light dir
193
194         vmulf   lightdir, lightdir, vconst[4]   # half (to avoid overflow)
195
196         vmudn   modnormf, mtf0, lightdir[0h]    # transform normal with 3x3
197         vmadn   modnormf, mtf1, lightdir[1h]    # transpose MP matrix
198         vmadn   modnormf, mtf2, lightdir[2h]    #
199         vmadm   modnormi, vconst, vconst[0]     #
200
201         vmudm   modnormf, modnormi, vconst[2]   # shift frac part right in acc
202
203         vmacf   modnormi, mtx0, lightdir[0h]    # do integer part
204         vmacf   modnormi, mtx1, lightdir[1h]    #
205         vmacf   modnormi, mtx2, lightdir[2h]    #
206         vmadn   modnormf, vconst, vconst[0]     #
207
208  ### BEGIN NORMALIZE
209  ### BEGIN NORMALIZE
210
211 .unname lightdir
212 .name   invsqti,        $v26
213 .name   invsqtf,        $v5
214
215         vmudl   invsqtf,modnormf,modnormf       # invsqt = x^2, y^2, z^2
216         vmadm   invsqtf,modnormi,modnormf       #
217         vmadn   invsqtf,modnormf,modnormi       #
218         vmadh   invsqti,modnormi,modnormi       #
219
220         vaddc   vtmpf,invsqtf,invsqtf[1q]       # sum 3 terms
221         vadd    vtmpi,invsqti,invsqti[1q]       # (into 3rd element)
222         vaddc   vtmpf,invsqtf,vtmpf[0h]         #
223         vadd    vtmpi,invsqti,vtmpi[0h]         #
```

```
224
225   .unname mtx3
226   .unname mtf3
227   .name   sqguessi,               $v11
228   .name   sqguessf,               $v15
229   .name   const3,                 $v2
230
231           vrsqh   sqguessi[0],vtmpi[2]            # 1/sqrt(sum)
232           vrsql   sqguessf[0],vtmpf[2]            # (into 0th element)
233           vrsqh   sqguessi[0],vconst[0]           #
234
235           vmudl   sqguessf,sqguessf,vconst1[3]    # mult * 1/128 (1/sqrt moves
236           vmadm   sqguessi,sqguessi,vconst1[3]    # radix point 7 places)
237           vmadn   sqguessf,vconst,vconst[0]       # (actually 1/128 * .99 to
238                                                   # ensure result <1 so no clamp)
239   .unname vtmpi
240   .unname vtmpf
241   .name   lightdiri,      $v4
242   .name   lightdirf,      $v7
243
244           vmudl   lightdirf,modnormf,sqguessf[0]  # 1/sqrt * normal
245           vmadm   lightdirf,modnormi,sqguessf[0]  #
246           vmadn   lightdirf,modnormf,sqguessi[0]  #
247           vmadh   lightdiri,modnormi,sqguessi[0]  #
248           vmadn   lightdirf,vconst,vconst[0]      #
249
250    ### clamp to max or min
251           ldv     const3[0],VCONST3_OFFSET-RSP_STATE_OFFSET(rsp_state)
252           vge     lightdirf,lightdirf,const3[0]   # if < -1fff then =-1fff
253           vlt     lightdirf,lightdirf,const3[1]   # if >  1fff then = 1fff
254           vmudn   lightdirf,lightdirf,const3[2]   # multiply * 4
255
256
257   .unname sqguessi
258   .unname sqguessf
259   .unname const3
260
261    ### END NORMALIZE
262
263           spv     lightdirf[0],(RSP_R_DIR+RSP_L_BASE)(lptr)
264           lw      tmp,(RSP_R_DIR+RSP_L_BASE)(lptr)
265           sw      tmp,(RSP_R_DIR+RSP_L_BASE+4)(lptr)
266
267   .unname lightdiri
268   .unname lightdirf
```

```
269   .name   mtx3,           $v11
270   .name   mtf3,           $v15
271
272           bgtz    lptr,litXformLoop               # loop if more lights
273           addi    lptr,lptr,(-1*RSP_L_LEN)        # next light
274    ### LOOP OCCURS to litXformLoop: if more lights are on
275
276           j       getMatrix                       # reload the matrix
277           lh      return,DOLIGHT(zero)            # return address
278    ### JUMP OCCURS to getMatrix: and then to doLight:
279
280   .unname invsqti
281   .unname invsqtf
282   .unname modnormi
283   .unname modnormf
284   .unname clr1
285   .unname clr2
286   .unname tmp
287   .unname lptr
288   .unname sumcol
289   .unname scratch
290   .unname mtx0
291   .unname mtx1
292   .unname mtx2
293   .unname mtx3
294   .unname mtf0
295   .unname mtf1
296   .unname mtf2
297   .unname mtf3
298   .unname gmode
```

gimm.s

```
 1
 2  /**************************************************************************
 3   *                                                                        *
 4   *               Copyright (C) 1994, Silicon Graphics, Inc.               *
 5   *                                                                        *
 6   *  These coded instructions, statements, and computer programs  contain  *
 7   *  unpublished  proprietary  information of Silicon Graphics, Inc., and  *
 8   *  are protected by Federal copyright  law.  They  may not be disclosed  *
 9   *  to  third  parties  or copied or duplicated in any form, in whole or  *
10   *  in part, without the prior written consent of Silicon Graphics, Inc.  *
11   *                                                                        *
12   **************************************************************************/
13
14  /*
15   * File:               gimm.s
16   * Creator:            hsa@sgi.com
17   * Create Date:        Fri Jun 24 13:55:27 PDT 1994
18   *
19   * This file holds the top-level of the IMM command processing, and
20   * related routines.
21   *
22   */
23
24   ###############################################################################
25   #
26   # The following code processes the IMM type display list commands.
27   # Registers on input:
28   #      gfx0    - first word of display list command
29   #      gfx1    - second word of display list command
30   #      dinp    - points to *next* DL cmd, so back up for this one.
31   #
32                  .ent    doIMM
33  doIMM:
34          # $2 is shifted in the delay slot of the branch that
35          # brought us here...
36
37          # 'switch' to correct IMM command:
38                  andi    $2, $2, 0xfe            # shifted up 1 for offset
39                  lh      $2,(IMM_JMP_ADD)($2)
40
41                  jr      $2
42          # consolidate some of the similar decoding...
43                  lbu     $1, (0-1)(dinp) # pick off first field
```

```
44
45                  .end    doIMM
46    #
47    #
48    #
49    ###############################################################################
50
51
52   #ifdef LINE3D
53
54    ###############################################################################
55    #
56    # This code handles G_LINE3D. It's basically modified from Steve's triangle
57    # code above. It picks off the two vertex indicies and calls the line code
58    #
59    #
60   .name v0,       $1
61   .name v1,       $2
62   .name wd,       $3
63   .name vn,       $4
64   .name n,        $5
65
66
67
68                  .ent    case_G_LINE3D
69
70     case_G_LINE3D:
71          # pick off flag field
72                  lbu     n, (0-4)(dinp)  # which normal?
73          # dinp points to next dl cmd, so back up to get line indices
74                  lbu     v0, (0-3)(dinp)
75                  lbu     v1, (0-2)(dinp)
76          # which normal?
77                  sll     n, n, 2                 # word-size offset
78
79          # this is a hack. we get the width, add 3 (0 means 'min' width)
80          # and store it back to an unused DMEM location. We'll retrieve
81          # it later and use it...
82                  lbu     wd, (0-1)(dinp)
83                  addi    wd, wd, 3
84                  sh      wd, CLIP_STATE_TABLE(zero)
85
86          # translate indices into DMEM offsets. Point buffer entries are
87          # 40 bytes (yuk!) each... The interface (mbi.h) pre-multiplies
88          # the indices by 10, so we just have to multiply by 4.
```

```
 89                 sll     v0, v0, 2
 90                 sll     v1, v1, 2
 91
 92
 93                 addi    v0, v0, RSP_POINTS_OFFSET
 94                 addi    v1, v1, RSP_POINTS_OFFSET
 95
 96                 sw      v0, (0+RSP_SCRATCH_OFFSET)(zero)
 97                 sw      v1, (4+RSP_SCRATCH_OFFSET)(zero)
 98                 lw      vn, RSP_SCRATCH_OFFSET(n)
 99
100                 jal     doClip
101                 nop     # delay slot, might do something useful here later
102                 j       GfxDone
103                 nop     # delay slot, might do something useful here later
104
105                 .end    case_G_LINE3D
106
107   .unname v0
108   .unname v1
109   .unname wd
110   .unname vn
111   .unname n
112    #
113    #
114    #
115    ###############################################################################
116
117   #else /* LINE3D */
118
119    ###############################################################################
120    #
121    # This code handles G_TRI1. It picks off the three vertex indicies and
122    # calls the triangle setup code.
123    #
124    #
125   .name v0,               $1
126   .name v1,               $2
127   .name v2,               $3
128   .name vn,               $4
129   .name n,                $5
130   .name tmp,              $6
131
132                 .ent    case_G_TRI1
133
```

```
134     case_G_TRI1:
135           # pick off flag field
136                   lbu     n, (0-4)(dinp)  # which normal?
137           # dinp points to next dl cmd, so back up to get tri indices
138                   lbu     v0, (0-3)(dinp)
139                   lbu     v1, (0-2)(dinp)
140                   lbu     v2, (0-1)(dinp)
141           # which normal?
142                   sll     n, n, 2                 # word-size offset
143
144           # translate indices into DMEM offsets. Point buffer entries are
145           # 40 bytes (yuk!) each... The interface (mbi.h) pre-multiplies
146           # the indices by 10, so we just have to multiply by 4.
147                   sll     v0, v0, 2
148                   sll     v1, v1, 2
149                   sll     v2, v2, 2
150
151                   addi    v0, v0, RSP_POINTS_OFFSET
152                   addi    v1, v1, RSP_POINTS_OFFSET
153                   addi    v2, v2, RSP_POINTS_OFFSET
154
155                   sw      v0, (0+RSP_SCRATCH_OFFSET)(zero)
156                   sw      v1, (4+RSP_SCRATCH_OFFSET)(zero)
157                   sw      v2, (8+RSP_SCRATCH_OFFSET)(zero)
158                   lw      vn, RSP_SCRATCH_OFFSET(n)
159
160                   j       clipAndSetup
161                   lh      return_save,GFXDONE(zero)       # return to GfxDone:
162                   .end    case_G_TRI1
163
164   .unname v0
165   .unname v1
166   .unname v2
167   .unname vn
168   .unname n
169   .unname tmp
170    #
171    #
172    #
173    ###############################################################################
174
175   #endif /* LINE3D */
176
177
178    ###############################################################################
```

```
179   #
180   # This code handles G_POPMTX. It checks the stack depth, backs up the
181   # stack pointer, then DMA's the matrix into DMEM, updates the state,
182   # and loads the registers.
183   #
184   #
185  .name param,            $1
186  .name mstack_p,         $19
187  .name mstack_max,       $3
188  .name mat_sz,           $18
189  .name mat_p,            $20
190
191                 .ent    case_G_POPMTX
192
193    case_G_POPMTX:
194          # 'param' already filled in but not used
195
196          # we can only pop the MODELVIEW stack
197
198          # get pointer and stack size
199          lw      mstack_p,  RSP_STATE_MTX_STACK_P(rsp_state)    # stack ptr
200          lw      mstack_max, RSP_STATE_MTX_STACK_MAX(rsp_state) # end of stack
201          addi    mat_p, zero, RSP_CURR_MTX_OFFSET       # where to DMA matrix
202          addi    mat_sz, zero, 63        # DMA expects sz-1      # DMA length -1
203          sub     mstack_max, mstack_max, mstack_p               # size of stack
204          addi    mstack_max, mstack_max, (-10*64)       #  ... - max size of stck
205
206          # check matrix stack depth, bail if == 0
207          bgez    mstack_max, GfxDone                    # anything on stack?
208          addi    mstack_p, mstack_p, -64                # stack is 1 mtx smaller
209   ### BRANCH OCCURS TO GfxDone: IF NOTHING ON STACK
210
211          jal     DMAproc                                # DMA matrix from stack
212          addi    $17, zero, 0                           # DMA is a READ
213
214          jal     DMAwait                        # wait for DMA to finish
215          addi    $3, zero, RSP_CURR_MPMTX_OFFSET # where to put MP matrix
216
217          # update state, then jump to pre-multiply MxP
218          j       mtx_MxP                        # mult model * proj mtx
219                                                 # store new stack size
220          sw      mstack_p,  RSP_STATE_MTX_STACK_P(rsp_state)
221
222
223                 .end    case_G_POPMTX
```

```
224   .unname param
225   .unname mstack_p
226   .unname mstack_max
227   .unname mat_sz
228   .unname mat_p
229    #
230    #
231    #
232    ###############################################################################
233
234    ###############################################################################
235    #
236    # this handles the G_MOVEWORD command, moving 1 word into dmem
237    #
238    #
239   .name target,   $1
240   .name outptr,   $5
241   .name offset,   $2
242
243                   .ent    case_G_MOVEWORD
244
245     case_G_MOVEWORD:
246                   lbu     target, (0-5)(dmp)              # index to address
247                   lhu     offset, (0-7)(dmp)              # offset from address
248                   lh      outptr,(MOVEWORD_TBL)(target) # actual address
249                   add     outptr,outptr,offset            #   ...plus offset
250                   j       GfxDone                         #
251                   sw      gfx1, 0(outptr)                 # store @ addr + off
252
253                   .end    case_G_MOVEWORD
254
255   .unname target
256   .unname outptr
257   .unname offset
258    #
259    #
260    ###############################################################################
261
262
263    ###############################################################################
264    #
265    # This code handles the G_TEXTURE.
266    #
267    #
268   .name   rmode,          $2
```

```
269   .name   mask,           $3
270   .name   sscale,         $4
271   .name   tscale,         $5
272   .name   tile,           $6
273
274                   .ent    case_G_TEXTURE
275
276     case_G_TEXTURE:
277
278           # turn texture on or off:
279                   sw      gfx0, RSP_STATE_TEX_CMD(rsp_state)
280                   sw      gfx1, RSP_STATE_TEX_SCALE_S(rsp_state)
281                   lh      rmode, RSP_STATE_RENDER_L(rsp_state)
282                   andi    rmode, rmode, 0xfffd    # clear texture state
283                   andi    mask, gfx0, 0x01        # on bit
284                   sll     mask, mask, 1
285                   or      rmode, rmode, mask      # set texture on (maybe)
286                   j       GfxDone
287                   sh      rmode, RSP_STATE_RENDER_L(rsp_state)
288
289                   .end    case_G_TEXTURE
290
291   .unname rmode
292   .unname mask
293   .unname sscale
294   .unname tscale
295   .unname tile
296
297    ###############################################################################
298    #
299    # This code handles G_SETOTHERMODE_*.
300    #
301    #
302   .name modewd,   $3
303   .name mask,     $2
304   .name lenth,    $5
305   .name shft,     $6
306   .name waddr,    $7
307   .name minus1,   $8
308
309                   .ent    case_G_OTHERMODE
310
311     case_G_SETOTHERMODE_H:
312                   j       doOtherMode
313                   addi    waddr, rsp_state, RSP_STATE_OTHER_H # delay slot
```

```
314
315     case_G_SETOTHERMODE_L:
316                 addi    waddr, rsp_state, RSP_STATE_OTHER_L
317
318                 # this code is the same for both OTHERMODE commands...
319           doOtherMode:
320                 lw      modewd, 0(waddr)
321                 addi    minus1, zero, -1
322
323                 lbu     lenth, (0-5)(dinp)
324                 lbu     shft,  (0-6)(dinp)
325
326                 addi    mask, zero, 0x01
327                 sllv    mask, mask, lenth
328                 addi    mask, mask, -1
329                 sllv    mask, mask, shft
330                 xor     mask, mask, minus1
331                 and     mask, mask, modewd
332                 or      modewd, mask, gfx1
333                 sw      modewd, 0(waddr)
334
335           # output to RDP
336           # writes 64-bits at once. cmd byte already there.
337           # use the regular RDP output routine, sharing code.
338                 lw      gfx0, RSP_STATE_OTHER_H(rsp_state)
339                 j       doRDPSend
340                 lw      gfx1, RSP_STATE_OTHER_L(rsp_state)
341
342                 .end    case_G_OTHERMODE
343
344   .unname modewd
345   .unname mask
346   .unname lenth
347   .unname shft
348   .unname waddr
349   .unname minus1
350    #
351    #
352    #
353    ###############################################################################
354
355
356    ###############################################################################
357    #
358    # This code handles G_CULLDL.
```

```
359   #
360   # Ends display list if vertices n through m are mutually trivially rejected
361   # (ie the volume described by these vertices is completely outside of the
362   # trivial reject volume).
363   #
364   .name cc,       $2
365   .name tmp,      $3
366
367                   .ent    case_G_CULLDL
368   case_G_CULLDL:
369                   andi    gfx0, gfx0, 0x3ff               # vertex to start on
370                   ori     cc, zero, 0xffff                # initialize cc
371   VolCulLoop:
372                   lh      tmp, (RSP_POINTS_OFFSET+RSP_PTS_CC)(gfx0)
373                   addi    gfx0, gfx0, RSP_PTS_LEN
374                   bne     gfx0, gfx1, VolCulLoop          # loop through vtx's
375                   and     cc, cc, tmp                     # is this vtx clipped?
376    ### LOOP OCCURS if not all points have been checked
377
378                   beq     cc, zero, GfxDone       # continue if not culled
379    # NOTE Delay Slot!!
380                                                   # If culled End DL
381
382                   .end    case_G_CULLDL
383   .unname cc
384   .unname tmp
385    #
386    #
387    #
388    ###############################################################################
389
390    ### IMPORTANT!!!!  Do not place any code betweeen case_G_CULLDL and G_ENDDL
391
392    ###############################################################################
393    #
394    # This code handles G_ENDDL.
395    #
396    # Causes a 'pop' of the display list stack. If we pop an empty
397    # display list stack, that's an error and we end.
398    #
399   .name stack_sz, $2
400   .name stack_p,  $3
401
402                   .ent    case_G_ENDDL
403
```

```
404     case_G_ENDDL:
405           # pop display list
406                   lb      stack_sz, RSP_STATE_DL_N(rsp_state)
407                   addi    stack_sz, stack_sz, -4
408                   bltz    stack_sz, TaskDone              # empty stack
409                   addi    stack_p, stack_sz, RSP_DLSTACK_OFFSET
410                   lw      inp,      0(stack_p)      # pointer of DL
411                   sb      stack_sz, RSP_STATE_DL_N(rsp_state)
412                   j       GfxDone
413                   addi    dlcount, zero, 0
414
415                   .end    case_G_ENDDL
416
417   .unname stack_sz
418   .unname stack_p
419    #
420    #
421    #
422    ###############################################################################
423
424
425    ###############################################################################
426    #
427    # This code handles G_SETGEOMETRYMODE
428    #
429    # Any bit set 'on' in the incoming command is 'set' in the state.
430    # Assumes gfx1 is all 0's, except some of the lower 16 bits.
431    #
432   .name   rmode,          $2
433
434                   .ent    case_G_SETGEOMETRYMODE
435
436     case_G_SETGEOMETRYMODE:
437                   lw      rmode, RSP_STATE_RENDER(rsp_state)
438                   or      rmode, rmode, gfx1
439                   j       GfxDone
440                   sw      rmode, RSP_STATE_RENDER(rsp_state) # delay slot
441
442                   .end    case_G_SETGEOMETRYMODE
443
444   .unname rmode
445    #
446    #
447    #
448    ###############################################################################
```

```
449
450
451   ###############################################################################
452   #
453   # This code handles G_CLEARGEOMETRYMODE
454   #
455   # Any bit set 'on' in the incoming command is 'cleared' in the state.
456   # Assumes gfx1 is all 0's, except some of the lower 16 bits.
457   #
458  .name   rmode,          $2
459  .name   mask,           $3
460
461                 .ent    case_G_CLEARGEOMETRYMODE
462
463    case_G_CLEARGEOMETRYMODE:
464                 lw      rmode, RSP_STATE_RENDER(rsp_state)
465                 addi    mask, zero, -1
466                 xor     mask, mask, gfx1
467                 and     rmode, rmode, mask
468                 j       GfxDone
469                 sw      rmode, RSP_STATE_RENDER(rsp_state) # delay slot
470
471                 .end    case_G_CLEARGEOMETRYMODE
472
473  .unname rmode
474  .unname mask
475   #
476   #
477   #
478   ###############################################################################
479
480   ###############################################################################
481   #
482   # This code handles G_PERSPNORM
483   #
484   # This magic number is needed to fix the transformation and clip
485   # math, extracting the most precision. Grab the scale from gfx1
486   # and save it for later.
487   #
488                 .ent    case_G_PERSPNORM
489
490      case_G_PERSPNORM:
491                 j       GfxDone
492                 sh      gfx1, RSP_STATE_PERSPNORM(rsp_state) # delay slot
493
```

```
494                 .end    case_G_PERSPNORM
495     #
496     #
497     #
498     ###############################################################################
499
500     ###############################################################################
501     #
502     # This code handles G_RDPHALF_1
503     #
504     # This received the 3rd quarter of a texrect or texrectflip command
505     #
506                 .ent    case_G_RDPHALF_1
507
508        case_G_RDPHALF_1:
509                 j       noYield                     # don't yield mid cmd
510                 sw      gfx1, RSP_STATE_RDPHALF(rsp_state) # save for later...
511
512                 .end    case_G_RDPHALF_1
513     #
514     #
515     #
516     ###############################################################################
517
518     ###############################################################################
519     #
520     # This code handles G_RDPHALF_CONT
521     #
522     # This received the 2nd 32 bits of a 64 bit string to send to the RDP.
523     # It sends it (using the G_RDPHALF_2 code below) AND disables yield
524     # so the data will not get interrupted by a yield.  This shoulb be used
525     # only to send data which will be followed by more data from a G_RDPHALF_1
526     # and G_RDPHALF_2 pair (or a G_RDPHALF_1 and G_RDPHALF_CONT pair).
527     #
528                 .ent    case_G_RDPHALF_CONT
529
530        case_G_RDPHALF_CONT:
531                 ori     $2, zero, 0             # disable yield
532
533                 .end    case_G_RDPHALF_CONT
534     #
535     #
536     #
537     ###############################################################################
538
```

```
539   ##############################################################################
540   #
541   # This code handles G_RDPHALF_2
542   #
543   # This received the 4rd quarter of a texrect or texrectflip command
544   # and sends it and the 3rd quarter (ie the 2nd half) to the rdp.
545   #
546                 .ent    case_G_RDPHALF_2
547
548      case_G_RDPHALF_2:
549                 j       doRDPSend                   # jmp to send routine
550                 lw      gfx0, RSP_STATE_RDPHALF(rsp_state) # retrieve 3rd qtr
551
552                 .end    case_G_RDPHALF_2
553   #
554   #
555   #
556   ##############################################################################
557
```

ginit.s

```
 1
 2  /**************************************************************************
 3   *                                                                        *
 4   *             Copyright (C) 1994, Silicon Graphics, Inc.                 *
 5   *                                                                        *
 6   *  These coded instructions, statements, and computer programs  contain  *
 7   *  unpublished  proprietary  information of Silicon Graphics, Inc., and  *
 8   *  are protected by Federal copyright  law.  They  may not be disclosed  *
 9   *  to  third  parties  or copied or duplicated in any form, in whole or  *
10   *  in part, without the prior written consent of Silicon Graphics, Inc.  *
11   *                                                                        *
12   **************************************************************************/
13
14  /*
15   * File:               ginit.s
16   * Creator:            acorn@sgi.com
17   * Create Date:        3/14/95
18   *
19   * This in the init code for the 'graphics task' which used to be in gmain.s
20   * but was moved to here so it could be overlayed with clip and light code.
21   *
22   */
23
24
25   ###########################################################################
26   #
27   #  Begin task initialization
28   #
29   #  Register $1 holds the task header address
30   # format for the task header:
31   #      (see task.h)
32   #
33   #
34                  .ent    doInit
35  doInit:
36          # set RSP yield flags to FALSE (IMPORTANT: don't clear yield flag)
37                  ori     $2,zero,(SP_CLR_YIELDED|SP_CLR_TASKDONE)
38                  mtc0    $2, SP_STATUS
39
40          #
41          # important constants:
42          #
43                  lqv     vconst[0], VCONST_OFFSET(zero)
```

```
44                  lqv     vconst1[0], VCONST1_OFFSET(zero)
45
46          #
47          # The task header should tell us if this is a restart
48          # of a previously checkpointed run. If so, do some special
49          # init sequence.
50          #
51  .name   yield, $4
52                  lw      yield, (RSP_TASK_OFFSET+OS_TASK_OFF_FLAGS)(zero)
53                  andi    yield, yield, OS_TASK_YIELDED
54                  bne     yield, zero, RSPYieldRestart
55                  nop
56  .unname yield
57
58
59  #ifdef OUTPUT_DRAM
60          # reset output pointers
61                  lw      outp, OS_TASK_OFF_OUTBUFF($1)
62                  lw      $3, OS_TASK_OFF_OUTBUFF_SZ($1)
63                  sw      outp, RSP_STATE_DRAM_OUTP(rsp_state)
64                  sw      $3, RSP_STATE_DRAM_OUT_LENP(rsp_state)
65
66                  addi    $4, zero, DPC_CLR_XBUS_DMEM_DMA
67                  mtc0    $4, CMD_STATUS
68                  addi    outp, zero, RSP_OUTPUT_OFFSET
69
70  #else
71  #ifdef OUTPUT_FIFO
72  .name   bufend, $3
73          # reset output pointers
74                  lw      outp, OS_TASK_OFF_OUTBUFF($1)
75                  lw      bufend, OS_TASK_OFF_OUTBUFF_SZ($1)
76                  sw      outp, RSP_STATE_FIFO_BUF_TOP(rsp_state)
77                  sw      bufend, RSP_STATE_FIFO_BUF_END(rsp_state)
78
79                  mfc0    $4, CMD_STATUS
80                  andi    $4, $4, DPC_STATUS_XBUS_DMEM_DMA
81                  bne     $4, zero, restart_fifo
82                  mfc0    $4, CMD_END
83                  sub     outp, outp, $4
84                  bgtz    outp, restart_fifo
85
86          mfc0    $5, CMD_CURRENT
87          beq     $5, zero, restart_fifo
88          nop
```

```
 89          beq     $5, $4, restart_fifo
 90
 91
 92                  nop
 93                  j       set_fifo
 94                  ori     bufend, $4, 0
 95
 96  restart_fifo:
 97                  mfc0    $4, CMD_STATUS
 98                  andi    $4, $4, DPC_STATUS_START_VALID
 99                  bne     $4, zero, restart_fifo
100                  addi    $4, zero, DPC_CLR_XBUS_DMEM_DMA
101   ### LOOP TO restart_fifo: UNTIL START_VALID=0
102                  mtc0    $4, CMD_STATUS
103                  mtc0    bufend, CMD_START
104                  mtc0    bufend, CMD_END
105  set_fifo:
106                  sw      bufend, RSP_STATE_FIFO_OUTP(rsp_state)
107                  addi    outp, zero, RSP_OUTPUT_OFFSET
108  .unname bufend
109
110  #else /* OUTPUT_XBUS: */
111          # reset output pointers
112                  lw      outp, OS_TASK_OFF_OUTBUFF($1)
113                  lw      $3, OS_TASK_OFF_OUTBUFF_SZ($1)
114                  sw      $0, RSP_STATE_DRAM_OUTP(rsp_state)
115                  sw      $3, RSP_STATE_DRAM_OUT_LENP(rsp_state)
116
117                  addi    $4, zero, DPC_SET_XBUS_DMEM_DMA
118                  addi    outp, zero, 0x1000      # DP initial conditions
119                  mtc0    $4, CMD_STATUS
120                  mtc0    outp, CMD_START
121                  mtc0    outp, CMD_END
122
123  #endif /* OUTPUT_XBUS */
124  #endif /* OUTPUT_XBUS */
125
126
127      ###########################################################################
128      #
129      # code overlays:
130      #
131          # update table to be real DRAM address:
132          # (awkward, we pipeline the load delays)
133  NoRestart:
```

```
134                 lw      $5, OS_TASK_OFF_UCODE($1)       # ucode base pointer
135
136     #ifdef FASTLIGHT3D
137             # PATCH NEWTON, CLIP, AND LIGHTING
138                 lw      $2, (OVERLAY_1_OFFSET + OVERLAY_OFFSET)(zero)
139                 lw      $3, (OVERLAY_2_OFFSET + OVERLAY_OFFSET)(zero)
140                 lw      $4, (OVERLAY_3_OFFSET + OVERLAY_OFFSET)(zero)
141                 lw      $6, (OVERLAY_4_OFFSET + OVERLAY_OFFSET)(zero)
142                 add     $2, $2, $5
143                 add     $3, $3, $5
144                 add     $4, $4, $5
145                 add     $6, $6, $5
146                 sw      $2, (OVERLAY_1_OFFSET + OVERLAY_OFFSET)(zero)
147                 sw      $3, (OVERLAY_2_OFFSET + OVERLAY_OFFSET)(zero)
148                 sw      $4, (OVERLAY_3_OFFSET + OVERLAY_OFFSET)(zero)
149                 sw      $6, (OVERLAY_4_OFFSET + OVERLAY_OFFSET)(zero)
150
151     #else /* FASTLIGHT3D */
152             # PATCH NEWTON ONLY
153                 lw      $2, (OVERLAY_1_OFFSET + OVERLAY_OFFSET)(zero)
154                 add     $2, $2, $5
155                 sw      $2, (OVERLAY_1_OFFSET + OVERLAY_OFFSET)(zero)
156
157     #endif /* FASTLIGHT3D */
158
159             # load newton code:
160                 jal     loadOverlaySR
161                 addi    $30, zero, OVERLAY_NEWTON
162     #ifndef FASTLIGHT3D
163                 .symbol NewtonDiv, 0x04001000           # for call-forward
164                 .symbol Newtons, 0x04001020             # for call-forward
165     #endif /* FASTLIGHT3D */
166
167         #
168         #
169         ###########################################################################
170
171             #
172             # get the first display list out of the header and
173             # start the DMA of the display list into DMEM.
174                 jal     LoadDL
175                 lw      inp, OS_TASK_OFF_DATA($1)
176
177             #
178             # while DMA is going on, do some initializations:
```

```
179         #
180         # MOST initialization is done at compile time, and loaded
181         # in during the PData DMA.
182         #
183
184         # initialize DRAM stack pointer in RSP state
185   .name dram_ptr, $2
186                 lw      dram_ptr, OS_TASK_OFF_STACK($1)
187                 sw      dram_ptr, RSP_STATE_DRAM_STACK(rsp_state)
188
189         # initialize matrix stacks to empty
190                 sw      dram_ptr, RSP_STATE_MMTX_STACK_P(rsp_state)
191                 addi    dram_ptr, dram_ptr, (10*64)
192                 sw      dram_ptr, RSP_STATE_MMTX_STACK_MAX(rsp_state)
193
194         # save RSP_STATE_YIELD_STORE address from task header
195                 lw      dram_ptr, -8(zero)
196                 sw      dram_ptr, RSP_STATEP_YIELD_STORE(zero)
197   .unname dram_ptr
198
199    #
200    #
201    #
202    # end of initialization section
203    #
204    ##############################################################################
205                 .end    doInit
```

gmain.s

```
 1
 2  /*****************************************************************************
 3   *                                                                           *
 4   *              Copyright (C) 1994, Silicon Graphics, Inc.                   *
 5   *                                                                           *
 6   *  These coded instructions, statements, and computer programs  contain     *
 7   *  unpublished  proprietary  information of Silicon Graphics, Inc., and     *
 8   *  are protected by Federal copyright  law.  They  may not be disclosed     *
 9   *  to  third  parties  or copied or duplicated in any form, in whole or     *
10   *  in part, without the prior written consent of Silicon Graphics, Inc.     *
11   *                                                                           *
12   *****************************************************************************/
13
14  /*
15   * File:               gmain.s
16   * Creator:            hsa@sgi.com
17   * Create Date:        Mon Jun 20 11:48:48 PDT 1994
18   *
19   * This is the 'graphics task' for the RSP. it takes blocks of display
20   * list commands and interprets them:    transforming, shading, clipping,
21   * DDA setup, etc., preparing geometry to be rasterized. It is also the
22   * center of the graphics system from the game's point of view, and
23   * receives all of the RDP control commands, which it passes along.
24   *
25   * In the absence of a linker, the different source code modules of the
26   * graphics task will be include'd here during compilation. See the Makefile
27   * for details.
28   *
29   */
30
31
32  #include <rsp.h>
33  #include <rcp.h>
34  #include <os.h>
35  #include <sptask.h>
36  #include "mbi.h"
37
38  #ifdef FASTLIGHT3D
39                  .text   RSP_IMEM_BASE   # this is coordinated with rspboot.s
40  #else /* FASTLIGHT3D */
41                  .text   TASKBASE        # this is coordinated with rspboot.s
42  #endif /* FASTLIGHT3D */
43                  .data   0
```

```
44
45  #define MAIN
46  #include "gdmem.h"
47  #include "gfx_regs.h"
48
49                  .text
50
51          ##########################################################################
52          #
53          #  Begin task initialization
54          #
55          #  Register $1 holds the task header address
56          # format for the task header:
57          #
58          # (see task.h)
59          #
60
61   #================================================
62   #======= in FASTLIGHT doInit is an overlay =======
63   #================================================
64  #ifdef FASTLIGHT3D
65
66  #include "newt.s" */
67   #              j       doInit          # initialization routine
68          # set state pointer:
69   #              addi    rsp_state, zero, RSP_STATE_OFFSET
70   ### JUMP OCCURS here to doInit:
71
72   #================================================
73   #======= ...Otherwise it is #included here =======
74   #================================================
75  #else /* FASTLIGHT3D */
76
77          # set state pointer:
78                  addi    rsp_state, zero, RSP_STATE_OFFSET
79
80  #include "ginit.s"
81
82  #endif /* FASTLIGHT3D */
83
84                  .ent    main
85
86          ##########################################################################
87          #
88          # Note about display list command processing:
```

```
 89          #
 90          #    - 'inp' always points to the "next" DL command in DRAM. This
 91          #       is the pointer pushed on the stack and used to retrieve the
 92          #       next block of DL commands.
 93          #
 94          #    - 'dinp' is the local pointer in DMEM to the part of the display
 95          #       list in DMEM.
 96          #
 97          #    - 'dlcount' parallels dinp, counting down the remaining DL
 98          #       commands in DMEM.
 99          #
100          # So when (dlcount == 0) we need to fetch more display list.
101          #
102          # NOTE:
103          # It would be useful to support an "immediate" mode from the host.
104          # This would allow a better performance/space trade-off using
105          # generated data, etc. In order to do this we need to modify the
106          # display list processing -- basically allow the gtask to go idle
107          # and wait for more display list, rather than exit. Needs a couple
108          # status registers to sync with the host (GO, IDLE, etc.)
109          #
110          # Fri Oct  7 10:45:04 PDT 1994
111          #
112          ######################################################################
113          #
114          #
115  DMAWaitDL:
116                  jal     DMAwait
117                  # note delay slot...
118
119          # process gfx list:
120  .name   dlcmd,  $1
121                  addi    dinp, zero, RSP_DLINPUT_OFFSET  # delay
122  DecodeDL:       lw      gfx0, 0(dinp)
123                  lw      gfx1, 4(dinp)
124                  srl     dlcmd, gfx0, 29
125                  andi    dlcmd, dlcmd, 0x06      # GFX op type (jump offset)
126
127          # advance these pointers after consuming gfx0/1
128                  addi    inp, inp, 8
129                  addi    dinp, dinp, 8
130                  addi    dlcount, dlcount, -8
131
132          #
133          # Get a head start on the next command, if it's a DMA.
```

```
134         # The register in_bufp points to the fetched data, and
135         # should implment the double-buffering...
136         #
137   .name dma_len,  $18
138   .name dram_addr, $19
139   .name dmem_addr, $20
140                 bgtz    dlcmd, ContDecode       # not a DMA...
141                 # note delay slot
142         #
143         # despite being a 16-bit field, the maximum DMA transfer
144         # is 256 bytes (16 vertices). Since the G_DL command will
145         # also come through here, with a much larger DMA length, we
146         # protect the length by masking to 0x01ff...
147         #
148
149         # WARNING!
150         # We are passing the wrong length to the DMA routine here.
151         # We should pass 'length-1'. We choose not to spend the extra
152         # instruction and correct this, because the place in DMEM
153         # that will potentially be trashed by the extra DMA is
154         # guarenteed to be trash-able. (setup tmp area)
155         #
156                 andi    dma_len, gfx0, 0x01ff   # pick off DMA length
157                 addi    in_bufp, zero, RSP_INPUT_OFFSET
158
159                 jal     AddrFixup
160                 add     dram_addr, gfx1, zero           # delay slot
161
162                 add     dmem_addr, zero, in_bufp
163                 jal     DMAproc
164                 addi    $17, zero, 0    # delay slot
165   .unname dma_len
166   .unname dram_addr
167   .unname dmem_addr
168
169         # we do the wait for the DMA in the gdma module. This
170         # lets us do some useful work while we wait...
171
172   .name op_addr, $2
173      ContDecode:
174                                 # compute jump address
175                 lh      op_addr, OPTYPE_JMP_OFFSET(dlcmd)
176                 jr      op_addr
177         # this delay slot is the first thing the other routines do...
178                 srl     $2, gfx0, 23            # table byte offset...
```

```
179  .unname op_addr
180  .unname dlcmd
181
182
183
184  .name   yield,          $2
185  .name   overeturn,      $21     # return address from loadOverlay
186
187  GfxDone:        # we're done with this one, do the next one (if available)...
188          #
189          # stick our head up, see if we need to yield the SP. If so,
190          # checkpoint everything then exit.
191          #                                               #
192                  mfc0    yield, SP_STATUS               # need to yield?
193                  andi    yield, yield, SP_STATUS_YIELD  #
194                  bne     yield, zero, RSPYield          #
195                  lh      overeturn, TASKYIELD(zero)     # return to RSPYield
196                                                         #
197  .unname yield                                          #
198                                                         #
199  noYield:                                               #
200                  bgtz    dlcount, DecodeDL              #
201                  nop                                    #
202   ### BRANCH OCCURS to DecodeDL if dlcount is not zero yet
203                  j       LoadDL                         #
204                  lh      return, DMAWAITDL(zero)        # return to DMAWaitDL
205   ### BRANCH OCCURS to LoadDL: and returns to DMAWaitDL:
206                                                         #
207                  .end    main                           #
208                                                         #
209   #=================================================================
210   #======= in FASTLIGHT TaskDone & Yield are overlays - load them =======
211   #=================================================================
212  #ifdef FASTLIGHT3D
213  .unname return_save
214  .name   overp,          $30     # which overlay to load
215  TaskDone:
216                  lh      overeturn, TASKDONE(zero)       # return to start
217  RSPYield:
218                  j       loadOverlay             # Load the overlay
219                  ori     overp, zero, OVERLAY_DONE # Load gdone overlay
220   ### BRANCH OCCURS HERE TO loadOverlay:
221
222  .unname overp
223  .unname overeturn
```

```
224   .name   return_save,    $30
225    #=====================================================================
226    #======= ...Otherwise include Done and Yield here (Yield is in done) ==
227    #=====================================================================
228   #else /* FASTLIGHT3D */
229   .unname overeturn
230
231   #include "gdone.s"
232   #endif /* FASTLIGHT3D */
233           #
234           #
235           ######################################################################
236
237           ######################################################################
238           #
239           # start the DMA of the display list into DMEM.
240           #
241           # Always reads RSP_DLINPUT_SIZE8 bytes into the buffer. We
242           # require lists to be terminated with G_ENDDL, so this might be
243           # reading a little beyond the actual end.
244           #
245           # Registers upon call:
246           #               inp     pointer to read from
247           #               return  where to go when we're done
248           #
249           # Registers upon return:
250           #               dlcount size actually read (bytes)
251           #               dinp    pointer to data we read in
252           #
253           # Registers used:
254           #               return, zero, inp, dlcount, dinp
255           #
256
257                   .ent    LoadDL
258   LoadDL:
259                   addi    dlcount, zero, RSP_DLINPUT_SIZE8
260                   add     $21, zero, return
261                   addi    $20, zero, RSP_DLINPUT_OFFSET
262                   add     $19, zero, inp
263                   addi    $18, zero, RSP_DLINPUT_SIZE8 - 1
264                   jal     DMAproc
265                   addi    $17, zero, 0
266                   jr      $21
267                   addi    dinp, zero, RSP_DLINPUT_OFFSET # delay slot
268
```

```
269                 .end LoadDL
270         #
271         #
272         ######################################################################
273
274         ######################################################################
275         #
276         # This routine loads one of the overlays from the DMEM table.
277         # Register $4 has the index into the overlay table to load.
278         #
279         # Assumes overlay table is the first thing in DMEM.
280         #
281   .unname return_save
282   .name   overp,          $30
283   .name   imem_addr,      $20
284   .name   dram_addr,      $19
285   .name   dma_len,        $18
286   .name   iswrite,        $17
287   .name   overeturn,      $21
288
289         loadOverlaySR:
290                 add     overeturn, zero, return
291         loadOverlay:
292                 lw      dram_addr, OVERLAY_OFFSET(overp)
293                 lh      dma_len,   OVERLAY_SIZE(overp)
294                 lh      imem_addr, OVERLAY_DEST(overp)
295                 jal     DMAproc
296                 addi    iswrite, zero, 0        # delay slot
297                 jal     DMAwait
298                 nop
299                 jr      overeturn
300                 # note delay slot
301   .unname overp
302   .unname imem_addr
303   .unname dram_addr
304   .unname dma_len
305   .unname iswrite
306   .unname overeturn
307   .name   return_save,    $30
308         #
309         #
310         ######################################################################
311
312         ######################################################################
313         #
```

```
314         # Address Fix-up routine.
315         #
316         # Takes a segment/offset address in register $19, computes
317         # the proper DRAM address using the segment table. Returns
318         # the answer in register $19
319         #
320         # (The use of register $19 is NOT random... It's the DRAM
321         # address used in DMAproc, often the next thing called.)
322         #
323         # This code is shared among several routines, the registers
324         # used are chosen not to conflict with those.
325         #
326 .name dma_addr, $19
327 .name mask,     $11
328 .name seg_id,   $12
329 .name seg_ptr,  $13
330                 .ent    AddrFixup
331     AddrFixup:
332                 lw      mask, SEGADDR_MASK_OFFSET(zero)
333                 srl     seg_id, dma_addr, 22
334                 andi    seg_id, seg_id, 0x3c    # Seg num only
335                 and     dma_addr, dma_addr, mask
336                 add     seg_ptr, zero, seg_id
337                 lw      seg_id, RSP_SEG_OFFSET(seg_ptr)
338                 jr      return
339                 add     dma_addr, dma_addr, seg_id      # delay slot
340                 .end    AddrFixup
341 .unname dma_addr
342 .unname mask
343 .unname seg_id
344 .unname seg_ptr
345         #
346         ######################################################################
347
348
349         ######################################################################
350         #
351         # Procedure to do DMA reads/writes.
352         #
353         # Registers:
354         #
355         #       $20     mem_addr
356         #       $19     dram_addr
357         #       $18     dma_len
358         #       $17     iswrite?
```

```
359         #
360         #       $11     used as tmp
361         #
362   .name   mem_addr,      $20
363   .name   dram_addr,     $19
364   .name   dma_len,       $18
365   .name   iswrite,       $17
366   .name   tmp,           $11
367
368   DMAproc:
369   #if 0
370   /* this code is not needed, we rely on the CPU's good
371    * behavior to not be using the DMA engine.
372    */
373         # request DMA access: (get semaphore)
374                 mfc0    tmp, SP_RESERVED
375                 bne     tmp, zero, DMAproc
376                 # note delay slot
377   #endif
378
379                 mfc0    tmp, DMA_FULL
380                 bne     tmp, zero, DMAproc
381                 nop
382         # set DMA registers:
383                 mtc0    mem_addr, DMA_CACHE
384         # handle writes:
385                 bgtz    iswrite, DMAWrite
386                 mtc0    dram_addr, DMA_DRAM
387
388                 jr      return
389                 mtc0    dma_len, DMA_READ_LENGTH
390
391         DMAWrite:
392                 jr      return
393                 mtc0    dma_len, DMA_WRITE_LENGTH
394   .unname mem_addr
395   .unname dram_addr
396   .unname dma_len
397   .unname iswrite
398   .unname tmp
399         #
400         #
401         ######################################################################
402
403         ######################################################################
```

```
404         #
405         # Procedure to do DMA waits.
406         #
407         # Registers:
408         #
409         #       $11     used as tmp
410         #
411  #.name tmp,    $20
412  #.name tmp,    $16 !! $16 needed as clr2 in lighting
413 .name   tmp,    $11
414
415 DMAwait:
416 #if 0
417 /* this code is not needed, we rely on the CPU's good
418  * behavior to not be using the DMA engine.
419  */
420         # request DMA access: (get semaphore)
421                 mfc0    tmp, SP_RESERVED
422                 bne     tmp, zero, DMAwait
423                 # note delay slot
424 #endif
425                 mfc0    tmp, DMA_BUSY
426                 bne     tmp, zero, DMAwait
427                 nop
428                 jr      return
429                 nop
430 .unname tmp
431         #
432         #
433         ######################################################################
434
435         ######################################################################
436         #
437         # other "modules" are appended by #include
438         #
439
440  #=============================================================================
441  #==== If FASTLIGHT Yield Restart is in init overlay, otherwise it's here =====
442  #=============================================================================
443 #ifndef FASTLIGHT3D
444 #       define YIELD_RESTART
445 #       define YIELD_STOP
446 #       include "gyield.s"
447 #       undef YIELD_STOP
448 #       undef YIELD_RESTART
```

```
449  #endif /* FASTLIGHT3D */
450
451   #============================================================================
452   #==== Output module (either DRAM or directly to XBUS) ======================
453   #============================================================================
454  #ifdef OUTPUT_DRAM
455  #   include "goutdram.s"
456  #else
457  #ifdef OUTPUT_FIFO
458  #   include "goutfifo.s"
459  #else
460  #   include "goutxbus.s"
461  #endif
462  #endif
463
464   #============================================================================
465   #==== Command parsing modules for immediate, rdp, and dma commands ==========
466   #============================================================================
467  #include "gimm.s"
468
469  #include "grdp.s"
470
471  #include "gdma.s"
472
473   #============================================================================
474   #==== Clip and/or light and/or setup code for lines or polygons =============
475   #============================================================================
476  #ifdef LINE3D
477  #   include "gclipl.s"
478  #   include "gline.s"
479  #else
480  #   include "goverlays.s"
481  #   ifndef NODATA
482  #ifndef _HW_VERSION_1
483  #       include "gsetup.s"
484  #else
485  #       include "gsetup1.s"
486  #endif
487  #   endif
488  #endif
489
490          EndOfProg:
491                  .print  __FILE__
492                  .print  " : IMEM used %d instructions ", (EndOfProg - 0x04001000)/4
493                  .print  "(must be < 1024).\n"
```

```
494                     .dmax   4096
495             #
496             #
497             ###################################################################
```

gmtx.s

```
 1
 2  /*
 3   * Copyright 1995, Silicon Graphics, Inc.
 4   * ALL RIGHTS RESERVED
 5   *
 6   * UNPUBLISHED -- Rights reserved under the copyright laws of the United
 7   * States.   Use of a copyright notice is precautionary only and does not
 8   * imply publication or disclosure.
 9   *
10   * U.S. GOVERNMENT RESTRICTED RIGHTS LEGEND:
11   * Use, duplication or disclosure by the Government is subject to restrictions
12   * as set forth in FAR 52.227.19(c)(2) or subparagraph (c)(1)(ii) of the Rights
13   * in Technical Data and Computer Software clause at DFARS 252.227-7013 and/or
14   * in similar or successor clauses in the FAR, or the DOD or NASA FAR
15   * Supplement.  Contractor/manufacturer is Silicon Graphics, Inc.,
16   * 2011 N. Shoreline Blvd. Mountain View, CA 94039-7311.
17   *
18   * THE CONTENT OF THIS WORK CONTAINS CONFIDENTIAL AND PROPRIETARY
19   * INFORMATION OF SILICON GRAPHICS, INC. ANY DUPLICATION, MODIFICATION,
20   * DISTRIBUTION, OR DISCLOSURE IN ANY FORM, IN WHOLE, OR IN PART, IS STRICTLY
21   * PROHIBITED WITHOUT THE PRIOR EXPRESS WRITTEN PERMISSION OF SILICON
22   * GRAPHICS, INC.
23   *
24   */
25
26   ##############################################################################
27   #
28   # Process the G_MTX command.
29   #
30   #      in_bufp holds pointer to data
31   #
32
33  .name param,            $1
34
35  .name do_load,          $7
36  .name do_proj,          $8      # flag for if we're doing PROJECTION...
37
38  .name mstack_sz,        $12     # whichever matrix stack size we need...
39  .name mstack_p,         $19     # pointer to whichever matrix stack we need...
40  .name mat_p,            $20     # pointer to whichever mat_p we are using...
41
42  .name   mtx0,           $v29
43  .name   mtx1,           $v28
```

```
44  .name   mtx2,           $v27
45  .name   mtx3,           $v26
46
47  case_G_MTX:
48          sbv     vconst[6],RSP_STATE_L_LEN(rsp_state)    # hi bit causes light recalc
49          andi    do_proj, param, G_MTX_PROJECTION        # doing projection mtx?
50          bne     do_proj, zero, mtx_StartProj            # if yes, do projection
51          andi    do_load, param, G_MTX_LOAD              # do load or mul?
52   ### BRANCH OCCURS TO startProj: IF DOING PROJECTION MATRIX
53
54  .unname do_proj
55  .name   do_push,        $8
56
57          addi    mat_p, zero, RSP_CURR_MMTX_OFFSET       # point to model matrix
58          andi    do_push, param, G_MTX_PUSH              # do push or not?
59          beq     do_push, zero, mtx_PushDone             # skip push if not
60          lqv     mtx3[0], 48(in_bufp)                    # get part 4/4 of mtx
61   ### BRANCH OCCURS TO mtx_PushDone: IF NOT PUSHING
62
63  .unname do_push
64  .name   mstack_max,     $8
65  .unname param
66  .name   mstack_next,    $1
67
68          lw      mstack_p, RSP_STATE_MMTX_STACK_P(rsp_state)     # top of matrix stack
69          lw      mstack_max, RSP_STATE_MMTX_STACK_MAX(rsp_state) # end of matrix stack
70          addi    $17, zero, 1                            # do a DMA WRITE
71          addi    mstack_next,mstack_p, 64                # point to next spot on stack
72          beq     mstack_p, mstack_max, mtx_PushDone      # skip if stack full
73          addi    mstack_sz, zero, 63                     # 64 byte matrix
74   ### BRANCH OCCURS TO mtx_PushDone IF STACK FULL
75          jal     DMAproc                                 # dma current matrix to stack
76          sw      mstack_next, RSP_STATE_MMTX_STACK_P(rsp_state)  # update stack pointer
77   ### BRANCH OCCURS TO SUBROUTINE DMAproc:
78          jal     DMAwait                                 # wait for DMA to finish
79
80  .unname mstack_next
81  .name   Mp,             $1              ############################
82  .name   Tp,             $2              ##       IMPORTANT        ##
83  .name   Np,             $3              ## These register names  ##
84  .name   Ni,             $v5             ## should match the ones ##
85  .name   Nf,             $v6             ## in the MatCat routine ##
86                                          ############################
87  mtx_PushDone:
88          lqv     mtx1[0], 16(in_bufp)            # get part 2/4 of matrix
```

```
 89         beq     do_load, zero, mtx_Mul          # branch to mul if multiplying
 90         lqv     mtx2[0], 32(in_bufp)            # get part 3/4 of matrix
 91  ### BRANCH OCCURS TO mtx_Mul: IF MULTIPLYING NEW MATRIX
 92         sqv     mtx3[0], 48(mat_p)              # store part 4/4 of matrix
 93         lqv     mtx0[0], 0(in_bufp)             # load part 1/4 of matrix
 94         sqv     mtx1[0], 16(mat_p)              # store part 2/4 of matrix
 95 mtx_Store:
 96         addi    $3, zero, RSP_CURR_MPMTX_OFFSET # where to put MP matrix
 97         sqv     mtx2[0], 32(mat_p)              # store part 3/4 of matrix
 98         sqv     mtx0[0], 0(mat_p)               # store part 1/4 of matrix
 99 mtx_MxP:
100         addi    $1, zero, RSP_CURR_MMTX_OFFSET  # Model matrix ptr for MP mult
101         addi    $2, zero, RSP_CURR_PMTX_OFFSET  # Project mtx ptr for MP mult
102         j       MatCat                          # concatenate M and P matrix
103         lh      return, GFXDONE(zero)           # return to GfxDone
104
105
106 mtx_StartProj:
107         lqv     mtx3[0], 48(in_bufp)            # get part 4/4 of matrix
108         j       mtx_PushDone                    # return to load & multiply proj mtx
109         addi    mat_p, zero, RSP_CURR_PMTX_OFFSET # point to projection matrix
110  ### BRANCH OCCURS TO mtx_PushDone:
111
112
113 mtx_Mul:
114         addiu   $3, zero, ((RSP_SCRATCH_OFFSET+15) & 0xfffffff0)        # put multiplied mtx here
115         addu    $1, zero, in_bufp       # ptr to mew matrix
116         jal     MatCat                  # concatenate new and old matrix
117         addu    $2, zero, mat_p         # old matrix pointer
118  ### BRANCH OCCURS TO SUBROUTINE MatCat:
119         sqv     Nf[0], 48(mat_p)        # store part 4/4 of mult'd matrix
120         sqv     Ni[0], 16(mat_p)        # store part 2/4 of mult'd matrix
121         lqv     mtx2[0],  0(Np)         # get part 3/4 of mult'd matrix
122         j       mtx_Store               # continue storing mult'd matrix
123         lqv     mtx0[0], -32(Np)        # get part 1/4 of mult'd matrix
124  ### BRANCH OCCURS TO mtx_Store:
125
126 .unname do_load
127 .unname mstack_sz
128 .unname mstack_p
129 .unname mat_p
130 .unname mtx0
131 .unname mtx1
132 .unname mtx2
133 .unname mtx3
```

```
134   .unname mstack_max
135   .unname Mp
136   .unname Tp
137   .unname Np
138   .unname Ni
139   .unname Nf
140
141   /******* utility routines *****/
142
143
144           ##########################################################################
145           #
146           # 4x4 matrix multiply routine:  N = MT
147           # At this point,
148           #       $1 holds DMEM pointer to M  (row major)
149           #       $2 holds DMEM pointer to T  (row major)
150           #       $3 holds DMEM pointer to N  (row major)
151           #
152           # Uses registers:       $18, $19, $16
153           #                       $v1, $v2, $v3, $v4, $v5, $v6
154           #
155           # WARNING: Does the write while we compute, so don't let
156           # the destination be the same as one of the sources.
157           #
158           # WARNING: Be careful these registers used don't overlap with
159           # any global registers.
160           #
161           # the following implementation is optimized for code space.
162           #
163
164
165
166
167   .name Mp,       $1      # pointer to M matrix                   ##########################
168   .name Tp,       $2      # pointer to T matrix                   ##      IMPORTANT       ##
169   .name Np,       $3      # pointer to N matrix                   ## These register names ##
170   .name Ni,       $v5     # N element integer (summed vector)     ## must match the names ##
171   .name Nf,       $v6     # N element frac (summed vector)        ## in the mtx routine   ##
172                                                                   ##########################
173   .name Mdone,    $18     # loop ends when this == Mp (4 loops)
174   .name ALLdone,  $19     # function ends when this == Np (2 loops)
175   .name Mi,       $v1     # M row integer (vector)
176   .name Mf,       $v2     # M row frac (vector)
177   .name Ti,       $v3     # T col integer (vector)
178   .name Tf,       $v4     # T col frac (vector)
```

```
179
180                 .ent    MatCat
181   MatCat:
182         addi    ALLdone,Np,16           # did whole thing when Np == Np0 + 16 (2 loops
183   matCatHalf:
184          vmudh  Ni,vconst,vconst[0]     # zero accumulator
185         addi    Mdone,Mp,8              # inner loop ends when Mp == Mp0 + 8 (4 loops)
186   matCatRow:
187         ldv     Ti[0],  0(Tp)           # load row of T mtx (2 times)
188         ldv     Tf[0], 32(Tp)           #
189         lqv     Mi[0], 0(Mp)            # load same column element of 2 rows of M mtx
190         lqv     Mf[0], 32(Mp)           #
191         ldv     Ti[8],  0(Tp)           # load another copy of same T mtx row
192         ldv     Tf[8], 32(Tp)           #
193
194          vmadl  Nf, Tf, Mf[0h]          # multiply 2 M elemnts * 2 T rows
195         addi    Mp, Mp, 2                       # next Mp element
196          vmadm  Nf, Ti, Mf[0h]          #
197         addi    Tp, Tp, 8                       # next Tp element
198          vmadn  Nf, Tf, Mi[0h]          #
199          vmadh  Ni, Ti, Mi[0h]          #
200         bne     Mp, Mdone, matCatRow            # Loop over 4 column elements
201          vmadn  Nf, vconst, vconst[0]   # needed to obtain sign of frac
202    ### LOOP OCCURS 4 TIMES TO matCatRow:
203
204
205         addi    Tp, Tp, -32             # start at top of T matrix again
206         addi    Mp, Mp, 8               # do bottom half of mtx M & mtx N
207         sqv     Ni[0], 0(Np)            # store half of completed matrix
208         sqv     Nf[0], 32(Np)           # store half of completed matrix
209         bne     Np, ALLdone, matCatHalf # finished after 2nd half done
210         addi    Np, Np, 16              # ready to calculate  bottom half of Matrix N
211    ### LOOP OCCURS HERE 2 TIMES TO matCatHalf:
212
213
214         jr      return
215         nop
216
217                 .end    MatCat
218
219   .unname Mp
220   .unname Tp
221   .unname Np
222   .unname Mdone
223   .unname ALLdone
```

```
224   .unname Mi
225   .unname Mf
226   .unname Ti
227   .unname Tf
228   .unname Ni
229   .unname Nf
230           #
231           # end of matrix multiply...
232           #
233           ###############################################################################
234
235
236    # ############################ CALLED BY VTX AND CLIP ######################
237    # This routine loads vpscale and vptrans with the screen scalefactors.
238
239   .name   gmode,          $3
240   .name   voutp,          $7
241   .name   vtmp,           $v3
242   .name   vpscale,        $v0
243   .name   vptrans,        $v1
244   .name   vp,             $8
245   .name   wscl,           $v19
246
247                   .ent getScaleTrans
248   getScaleTrans:
249           # get OpenGL scale
250                   addi    vp, zero, RSP_VIEWPORT_OFFSET     # dmembase repl by 0
251                   lqv     vtmp[0], VOPENGL_OFFSET(zero)     # dmembase repl by 0
252
253                   lsv     wscl[0], RSP_STATE_PERSPNORM(rsp_state)
254                   lh      gmode,(RSP_STATE_RENDER)(rsp_state) # for fog
255           # load the viewport. Remember that these guys have 1 bit of
256           # fraction, so we must account for that later...
257           #
258                   ldv     vpscale[0], RSP_VIEWPORT_SX(vp)
259                   ldv     vptrans[0], RSP_VIEWPORT_TX(vp)
260                   ldv     vpscale[8], RSP_VIEWPORT_SX(vp)
261                   ldv     vptrans[8], RSP_VIEWPORT_TX(vp)
262           # correct the vpscale to match OpenGL... (bogus)
263                   jr      return
264                   vmudh   vpscale, vpscale, vtmp
265
266                   .end getScaleTrans
267   .unname wscl
268   .unname vp
```

```
269   .unname gmode
270   .unname voutp
271   .unname vtmp
272   .unname vpscale
273   .unname vptrans
274
275
276    # ############################ CALLED BY VIX AND FLIGHT ###################
277    # This routine loads the matrix
278
279   .name tmp,      $8
280   .name mtx0,     $v8
281   .name mtx1,     $v9
282   .name mtx2,     $v10
283   .name mtx3,     $v11
284   .name mtf0,     $v12
285   .name mtf1,     $v13
286   .name mtf2,     $v14
287   .name mtf3,     $v15
288
289                   .ent getMatrix
290   getMatrix:
291           # get MP matrix:        (load twice)
292                   addi    tmp, zero, RSP_CURR_MPMIX_OFFSET
293                   ldv     mtx3[0], 24(tmp)
294                   ldv     mtx3[8], 24(tmp)
295                   ldv     mtf3[0], 56(tmp)
296                   ldv     mtf3[8], 56(tmp)
297   getMatrix3:
298                   ldv     mtx0[0],  0(tmp)
299                   ldv     mtx1[0],  8(tmp)
300                   ldv     mtx2[0], 16(tmp)
301                   ldv     mtf0[0], 32(tmp)
302                   ldv     mtf1[0], 40(tmp)
303                   ldv     mtf2[0], 48(tmp)
304                   ldv     mtx0[8],  0(tmp)
305                   ldv     mtx1[8],  8(tmp)
306                   ldv     mtx2[8], 16(tmp)
307                   ldv     mtf0[8], 32(tmp)
308                   ldv     mtf1[8], 40(tmp)
309                   jr      return
310                   ldv     mtf2[8], 48(tmp)
311
312                   .end getMatrix
313
```

```
314   .unname tmp
315   .unname mtx0
316   .unname mtx1
317   .unname mtx2
318   .unname mtx3
319   .unname mtf0
320   .unname mtf1
321   .unname mtf2
322   .unname mtf3
```

goutdram.s

```
 1
 2  /*
 3   * Copyright 1995, Silicon Graphics, Inc.
 4   * ALL RIGHTS RESERVED
 5   *
 6   * UNPUBLISHED -- Rights reserved under the copyright laws of the United
 7   * States.   Use of a copyright notice is precautionary only and does not
 8   * imply publication or disclosure.
 9   *
10   * U.S. GOVERNMENT RESTRICTED RIGHTS LEGEND:
11   * Use, duplication or disclosure by the Government is subject to restrictions
12   * as set forth in FAR 52.227.19(c)(2) or subparagraph (c)(1)(ii) of the Rights
13   * in Technical Data and Computer Software clause at DFARS 252.227-7013 and/or
14   * in similar or successor clauses in the FAR, or the DOD or NASA FAR
15   * Supplement.  Contractor/manufacturer is Silicon Graphics, Inc.,
16   * 2011 N. Shoreline Blvd. Mountain View, CA 94039-7311.
17   *
18   * THE CONTENT OF THIS WORK CONTAINS CONFIDENTIAL AND PROPRIETARY
19   * INFORMATION OF SILICON GRAPHICS, INC. ANY DUPLICATION, MODIFICATION,
20   * DISTRIBUTION, OR DISCLOSURE IN ANY FORM, IN WHOLE, OR IN PART, IS STRICTLY
21   * PROHIBITED WITHOUT THE PRIOR EXPRESS WRITTEN PERMISSION OF SILICON
22   * GRAPHICS, INC.
23   *
24   */
25
26   ############################################################################
27   #
28   # Output coordination routines:
29   #
30   # We support two cases:
31   #
32   #      1.) output directly to RDP using DMEM as buffer. Requires
33   #          producer/consumer synchronization.
34   #
35   #      2.) output routed to DRAM. In this case, we use DMEM as
36   #          a temporary buffer, then DMA out to DRAM.
37   #
38   # Goals:
39   #
40   #      - small code size.
41   #      - efficient and non-obtrusive
42   #      - flexible, can easily route to DRAM or RDP.
43   #
```

```
44   # In order to do this, and make it transparent to the parts of the
45   # code which do the writing, we implement an open/close interface.
46   #
47   # FUNCTION:        OUTPUT TO RDP:                  OUTPUT TO DRAM:
48   #-----------------------------------------------------------------------
49   # open(size)    - wait for size avail in        - does nothing
50   #                 ring buffer.
51   #               - possibly handle wrap
52   #               - wait for 'current' to get
53   #                 out of the way
54   #
55   # close()       - advance CP0 pointer           - do DMA of cmd to DRAM
56   #                                               - increment pointers
57   #                                               - reset DMEM buffer
58   #
59   # In between the open() and close(), the ucode writes the data to DMEM.
60   # (to either the RDP ring buffer or a temp buffer to be DMA'd)
61   #
62   ###############################################################################
63
64           #
65           # These registers used by both routines.
66           #
67   .name   dmemp,          $20
68   .name   dramp,          $19
69   .name   outsz,          $18 # set by caller to max size to write
70
71   ###############################################################################
72   #
73   #
74   #
75   #if !(defined(OUTPUT_DRAM)||defined(OUTPUT_FIFO))
76                   .ent    OutputOpen
77
78   OutputOpen:
79                   jr      return
80                   nop
81
82                   .end    OutputOpen
83   #endif /* !(OUTPUT_DRAM || OUTPUT_FIFO) */
84   #
85   #
86   #
87   ###############################################################################
88
```

```
 89
 90
 91   ##############################################################################
 92   #
 93   #
 94   #
 95                   .ent    OutputClose
 96
 97   OutputClose:
 98                   lw      dramp, RSP_STATE_DRAM_OUTP(rsp_state)
 99                   addi    dmemp, zero, RSP_OUTPUT_OFFSET
100
101   .name   outlen,         $17
102                   lw      outlen, (RSP_STATE_DRAM_OUT_LEN+4)(rsp_state)
103                   sub     outsz, outp, dmemp
104                   add     outlen, outlen, outsz
105                   sw      outlen, (RSP_STATE_DRAM_OUT_LEN+4)(rsp_state)
106                   addi    outsz, outsz, -1
107   .unname outlen
108
109           # DEBUG:
110           # safety precaution, only do DMA if outsz >= 0
111                   bltz    outsz, OutputDone
112                   add     $21, zero, return
113           # if outsz < 0, we will drop data on the floor
114
115           OutputData:
116                   jal     DMAproc
117                   addi    $17, zero, 1    # set 'iswrite'
118                   jal     DMAwait
119                   nop
120
121           # update pointers...
122           # (dmemp gets trashed in DMAwait routine...)
123           OutputDone:
124                   addi    outp, zero, RSP_OUTPUT_OFFSET
125                   add     dramp, dramp, outsz
126                   addi    dramp, dramp, 1
127                   jr      $21
128                   sw      dramp, RSP_STATE_DRAM_OUTP(rsp_state)
129
130                   .end    OutputClose
131
132   #
133   #
```

```
134   #
135   ###############################################################################
136
137   .unname outsz
138   .unname dramp
139   .unname dmemp
140
141   ###################################################################
142   ##  Padding                                                      ##
143   ###################################################################
144   #make code the same length for all 3 versions of output
145
146                 nop
147                 nop
148                 nop
149                 nop
150                 nop
151                 nop
152                 nop
153                 nop
154                 nop
155                 nop
156                 nop
157                 nop
158                 nop
159                 nop
160                 nop
161                 nop
162                 nop
163                 nop
164
```

goutfifo.s

```
 1
 2  /*
 3   * Copyright 1995, Silicon Graphics, Inc.
 4   * ALL RIGHTS RESERVED
 5   *
 6   * UNPUBLISHED -- Rights reserved under the copyright laws of the United
 7   * States.   Use of a copyright notice is precautionary only and does not
 8   * imply publication or disclosure.
 9   *
10   * U.S. GOVERNMENT RESTRICTED RIGHTS LEGEND:
11   * Use, duplication or disclosure by the Government is subject to restrictions
12   * as set forth in FAR 52.227.19(c)(2) or subparagraph (c)(1)(ii) of the Rights
13   * in Technical Data and Computer Software clause at DFARS 252.227-7013 and/or
14   * in similar or successor clauses in the FAR, or the DOD or NASA FAR
15   * Supplement.  Contractor/manufacturer is Silicon Graphics, Inc.,
16   * 2011 N. Shoreline Blvd. Mountain View, CA 94039-7311.
17   *
18   * THE CONTENT OF THIS WORK CONTAINS CONFIDENTIAL AND PROPRIETARY
19   * INFORMATION OF SILICON GRAPHICS, INC. ANY DUPLICATION, MODIFICATION,
20   * DISTRIBUTION, OR DISCLOSURE IN ANY FORM, IN WHOLE, OR IN PART, IS STRICTLY
21   * PROHIBITED WITHOUT THE PRIOR EXPRESS WRITTEN PERMISSION OF SILICON
22   * GRAPHICS, INC.
23   *
24   */
25
26   # DMEM is utilized as follows for this output routine:
27   # RSP_STATE_FIFO_OUTP:        pointer in dram to next empty ring buffer entry
28   # RSP_STATE_FIFO_BUF_TOP:     pointer in dram to beginning of ring buffer
29   # RSP_STATE_FIFO_BUF_END:     pointer in dram to end of ring buffer
30   #                                (ie the byte following the ring buffer)
31
32          #
33          # These registers used by both routines.
34          #
35   .name   dmemp,          $20
36   .name   dramp,          $19
37   .name   outsz,          $18 # set by caller to max size to write
38
39    ###############################################################################
40    #
41    #
42    #
43   #if !(defined(OUTPUT_DRAM)||defined(OUTPUT_FIFO))
```

```
44                  .ent    OutputOpen
45
46  OutputOpen:
47                  jr      return
48                  nop
49
50                  .end    OutputOpen
51  #endif /* !(OUTPUT_DRAM || OUTPUT_FIFO) */
52   #
53   #
54   #
55   ###############################################################################
56
57   ###############################################################################
58   #
59   #
60   #
61                  .ent    OutputClose
62
63  OutputClose:
64   # check if the packet will fit in the buffer
65                  add     $21, zero, return
66                  lw      dramp, RSP_STATE_FIFO_OUTP(rsp_state)
67                  addi    outsz, outp, (-1*RSP_OUTPUT_OFFSET)
68                  lw      outp,  RSP_STATE_FIFO_BUF_END(rsp_state)
69                  blez    outsz, CloseDone
70                  add     dmemp, dramp, outsz             # end data in dram
71                  sub     dmemp, outp, dmemp              # bigger than buffer?
72                  bgez    dmemp, CurrentFit
73
74  WrapBuffer:
75   # packet will not fit, wait for current to wrap
76                  mfc0    dmemp, CMD_STATUS
77                  andi    dmemp, dmemp, 0x0400
78                  bne     dmemp, zero, WrapBuffer
79                  # note delay slot
80
81   # wait for current to advance
82  AdvanceCurrent:
83                  mfc0    outp, CMD_CURRENT               # outp = current
84                  lw      dramp, RSP_STATE_FIFO_BUF_TOP(rsp_state) # buffer start
85                  beq     outp, dramp, AdvanceCurrent
86                  nop
87                  mtc0    dramp, CMD_START        # reset START
88  CurrentFit:
```

```
 89                 # done if current_address <= dramp
 90                 mfc0    outp, CMD_CURRENT              # outp = current
 91                 sub     dmemp, dramp, outp
 92                 bgez    dmemp, CloseDMA
 93                 # note delay slot
 94
 95                 # loop if current_address <= (outp + outsz)
 96                 add     dmemp, dramp, outsz
 97                 sub     dmemp, dmemp, outp
 98                 bgez    dmemp, CurrentFit
 99                 nop
100   CloseDMA:
101                 add     outp, dramp, outsz
102                 addi    outsz, outsz, -1
103                 addi    dmemp, zero, RSP_OUTPUT_OFFSET
104                 jal     DMAproc
105                 addi    $17, zero, 1    # set 'iswrite'
106                 jal     DMAwait
107                 sw      outp, RSP_STATE_FIFO_OUTP(rsp_state)
108                 mtc0    outp, CMD_END
109
110   CloseDone:
111                 jr      $21
112                 addi    outp, zero, RSP_OUTPUT_OFFSET
113
114                 .end    OutputClose
115    #
116    #
117    #
118    ###############################################################################
119
120   .unname outsz
121   .unname dramp
122   .unname dmemp
123
124
```

goutxbus.s

```
 1
 2  /*
 3   * Copyright 1995, Silicon Graphics, Inc.
 4   * ALL RIGHTS RESERVED
 5   *
 6   * UNPUBLISHED -- Rights reserved under the copyright laws of the United
 7   * States.   Use of a copyright notice is precautionary only and does not
 8   * imply publication or disclosure.
 9   *
10   * U.S. GOVERNMENT RESTRICTED RIGHTS LEGEND:
11   * Use, duplication or disclosure by the Government is subject to restrictions
12   * as set forth in FAR 52.227.19(c)(2) or subparagraph (c)(1)(ii) of the Rights
13   * in Technical Data and Computer Software clause at DFARS 252.227-7013 and/or
14   * in similar or successor clauses in the FAR, or the DOD or NASA FAR
15   * Supplement.  Contractor/manufacturer is Silicon Graphics, Inc.,
16   * 2011 N. Shoreline Blvd. Mountain View, CA 94039-7311.
17   *
18   * THE CONTENT OF THIS WORK CONTAINS CONFIDENTIAL AND PROPRIETARY
19   * INFORMATION OF SILICON GRAPHICS, INC. ANY DUPLICATION, MODIFICATION,
20   * DISTRIBUTION, OR DISCLOSURE IN ANY FORM, IN WHOLE, OR IN PART, IS STRICTLY
21   * PROHIBITED WITHOUT THE PRIOR EXPRESS WRITTEN PERMISSION OF SILICON
22   * GRAPHICS, INC.
23   *
24   */
25
26  .name   dmemp,          $20
27  .name   dramp,          $19
28  .name   outsz,          $18     # set by caller to max size to write
29
30   ###############################################################################
31   #
32   # Output coordination routines:
33   #
34   # We support two cases:
35   #
36   #      1.) output directly to RDP using DMEM as buffer. Requires
37   #          producer/consumer synchronization.
38   #
39   #      2.) output routed to DRAM. In this case, we use DMEM as
40   #          a temporary buffer, then DMA out to DRAM.
41   #
42   # Goals:
43   #
```

```
44  #      - small code size.
45  #      - efficient and non-obtrusive
46  #      - flexible, can easily route to DRAM or RDP.
47  #
48  # In order to do this, and make it transparent to the parts of the
49  # code which do the writing, we implement an open/close interface.
50  #
51  # FUNCTION:        OUTPUT TO RDP:                  OUTPUT TO DRAM:
52  #-----------------------------------------------------------------------
53  # open(size)   - wait for size avail in        - does nothing
54  #                ring buffer.
55  #              - possibly handle wrap
56  #              - wait for 'current' to get
57  #                out of the way
58  #
59  # close()      - advance CP0 pointer           - do DMA of cmd to DRAM
60  #                                              - increment pointers
61  #                                              - reset DMEM buffer
62  #
63  # In between the open() and close(), the ucode writes the data to DMEM.
64  # (to either the RDP ring buffer or a temp buffer to be DMA'd)
65  #
66  ###############################################################################
67
68                .ent    OutputOpen
69
70  OutputOpen:
71   # check if the packet will fit in the buffer
72                addi    dramp, zero, (RSP_OUTPUT_OFFSET + RSP_OUTPUT_SIZE8)
73                add     dmemp, outp, outsz
74                sub     dramp, dramp, dmemp
75                bgez    dramp, CurrentFit
76                nop
77
78  WrapBuffer:
79   # packet will not fit, wait for current to wrap
80                mfc0    dramp, CMD_STATUS
81                andi    dramp, dramp, 0x0400
82                bne     dramp, zero, WrapBuffer
83                # note delay slot
84
85   # wait for current to advance
86  AdvanceCurrent:
87                mfc0    dramp, CMD_CURRENT
88                addi    outp, zero, RSP_OUTPUT_OFFSET
```

```
 89                 beq     dramp, outp, AdvanceCurrent
 90                 nop
 91                 mtc0    outp, CMD_START         # reset START
 92   CurrentFit:
 93                 # done if current_address <= outp
 94                 mfc0    dramp, CMD_CURRENT
 95                 sub     dmemp, outp, dramp
 96                 bgez    dmemp, OpenDone
 97                 # note delay slot
 98
 99                 # loop if current_address <= (outp + outsz)
100                 add     dmemp, outp, outsz
101                 sub     dramp, dmemp, dramp
102                 bgez    dramp, CurrentFit
103                 nop
104   OpenDone:
105                 jr      return
106                 nop
107
108                 .end    OutputOpen
109
110    ######################################################################
111    ##  OutputClose                                                     ##
112    ######################################################################
113                 .ent    OutputClose
114
115   OutputClose:
116          #
117          # XBUS RDP output
118          #
119                 jr      return
120                 mtc0    outp, CMD_END
121
122                 .end    OutputClose
123
124   .unname outsz
125   .unname dramp
126   .unname dmemp
127
128    ######################################################################
129    ##  Padding                                                         ##
130    ######################################################################
131    #make code the same length for all 3 versions of output
132
133                 nop
```

```
134             nop
135             nop
136             nop
137             nop
138             nop
139             nop
140             nop
141
```

goverlays.s

```
 1
 2  /*
 3   * Copyright 1995, Silicon Graphics, Inc.
 4   * ALL RIGHTS RESERVED
 5   *
 6   * UNPUBLISHED -- Rights reserved under the copyright laws of the United
 7   * States.   Use of a copyright notice is precautionary only and does not
 8   * imply publication or disclosure.
 9   *
10   * U.S. GOVERNMENT RESTRICTED RIGHTS LEGEND:
11   * Use, duplication or disclosure by the Government is subject to restrictions
12   * as set forth in FAR 52.227.19(c)(2) or subparagraph (c)(1)(ii) of the Rights
13   * in Technical Data and Computer Software clause at DFARS 252.227-7013 and/or
14   * in similar or successor clauses in the FAR, or the DOD or NASA FAR
15   * Supplement.  Contractor/manufacturer is Silicon Graphics, Inc.,
16   * 2011 N. Shoreline Blvd. Mountain View, CA 94039-7311.
17   *
18   * THE CONTENT OF THIS WORK CONTAINS CONFIDENTIAL AND PROPRIETARY
19   * INFORMATION OF SILICON GRAPHICS, INC. ANY DUPLICATION, MODIFICATION,
20   * DISTRIBUTION, OR DISCLOSURE IN ANY FORM, IN WHOLE, OR IN PART, IS STRICTLY
21   * PROHIBITED WITHOUT THE PRIOR EXPRESS WRITTEN PERMISSION OF SILICON
22   * GRAPHICS, INC.
23   *
24   */
25
26   ############################################################################
27   #
28   # This code #includes clipping, lighting, and init code, and ensures
29   # the proper overlay is in place when needed.
30   #
31   ############################################################################
32
33
34  #if defined(CLIP_ALONE)||defined(FLIGHT)
35  #include <rsp.h>
36  #include "mbi.h"
37
38                  .text   startClip
39
40  #include "gdmem.h"
41  #include "gfx_regs.h"
42  #endif /* CLIP_ALONE or FLIGHT */
43
```

```
44
45
46                  .align  8
47
48  #if defined(FLIGHT) || defined(FASTLIGHT3D)
49   ###############################################################################
50   ##################### LOAD CLIPPING ###########################################
51   ###############################################################################
52  .unname return_save
53  .name   overp,          $30     # which overlay to load
54  .name   overeturn,      $21     # return address from loadOverlay
55  startClip:
56          ori     overp, zero, OVERLAY_CLIP  # load clipping
57          beq     zero,zero,loadOverlay      # jump to the overlay loading routine
58  taskDone:                                      # taskDone here in DONE overlay
59          lh      overeturn, DOCLIP(zero)    # return to clip code
60   ###BRANCH OCCURS to loadOverlay: which will return to doClip:
61  .unname overp
62  .unname overeturn
63  .name   return_save,    $30
64
65  #endif /* FLIGHT or FASTLIGHT3D */
66
67  #ifdef FLIGHT
68   ###############################################################################
69   ##################### PERFORM LIGHTING ########################################
70   ###############################################################################
71  #include "gflight.s"                   # do lighting
72  #endif /* FLIGHT */
73
74  #ifdef CLIP_ALONE
75   ###############################################################################
76   ##################### JUMP TO CLIPPING ROUTINE ################################
77   ###############################################################################
78  startClip:
79          beq     zero, zero, doClip      # jump into clipping code
80          sh      return,RETURNJUMP2($0)  # save return address from setup
81          nop                             # delay slot
82          nop                             #
83  #endif /* CLIP_ALONE */
84
85  #if defined(CLIP_ALONE) || defined(FASTLIGHT3D)
86   ###############################################################################
87   ##################### LOAD LIGHTING ###########################################
88   ###############################################################################
```

```
 89  .unname return_save
 90  .name   overp,          $30     # which overlay to load
 91  .name   overeturn,      $21     # return address from loadOverlay
 92  doLight:
 93          ori     overp, zero, OVERLAY_LIGHT # load lighting
 94          beq     zero, zero, loadOverlay    # jump to the overlay loading routine
 95          lh      overeturn,DOLIGHT(zero)    # return to clip code
 96   ###BRANCH OCCURS to loadOverlay: which will return to doLight:
 97  .unname overp
 98  .unname overeturn
 99  .name   return_save,    $30
100  #endif /* CLIP_ALONE or FASTLIGHT3D */
101
102
103  #if !defined(FASTLIGHT3D) && !defined(FLIGHT)
104   ##############################################################################
105   ################### PERFORM CLIPPING #########################################
106   ##############################################################################
107  #include "gclip.s"
108
109  #endif /* CLIP_ALONE */
110
111  #ifdef FASTLIGHT3D
112   ##############################################################################
113   ################### PERFORM INITIALIZATION ###################################
114   ##############################################################################
115
116  #include "ginit.s"
117          j       DMAWaitDL               # return to main loop and begin
118          nop
119
120   ##############################################################################
121   ################### PERFORM YIELD RESTART ####################################
122   ##############################################################################
123  #define YIELD_RESTART
124  #include "gyield.s"
125  #undef YIELD_RESTART
126
127          # Align code following
128                  .align  8
129
130  #ifdef NODATA
131  clipAndSetup:
132                  nop
133  beginSetup:
```

```
134                 nop
135   #else /* NODATA */
136   #  ifdef OUTPUT_DRAM
137   #    include "gfillnops.dram.s"
138   #  else /* OUTPUT_DRAM */
139   #  ifdef OUTPUT_FIFO
140   #    include "gfillnops.fifo.s"
141   #  else /* OUTPUT_FIFO */
142   #    include "gfillnops.s"
143   #  endif /* OUTPUT_FIFO */
144   #  endif /* OUTPUT_DRAM */
145   #endif /* NODATA */
146
147   #endif /* FASTLIGHT3D */
148
```

grdp.s

```
 1
 2  /**************************************************************************
 3   *                                                                        *
 4   *               Copyright (C) 1994, Silicon Graphics, Inc.               *
 5   *                                                                        *
 6   *  These coded instructions, statements, and computer programs  contain  *
 7   *  unpublished  proprietary  information of Silicon Graphics, Inc., and  *
 8   *  are protected by Federal copyright  law.  They  may not be disclosed  *
 9   *  to  third  parties  or copied or duplicated in any form, in whole or  *
10   *  in part, without the prior written consent of Silicon Graphics, Inc.  *
11   *                                                                        *
12   **************************************************************************/
13
14  /*
15   * File:               grdp.s
16   * Creator:            hsa@sgi.com
17   * Create Date:        Fri Jun 24 13:54:26 PDT 1994
18   *
19   * This file holds the top-level of the RDP command processing, and
20   * related routines.
21   *
22   */
23
24
25   ##############################################################################
26   #
27   # The following code processes the RDP type display list commands.
28   # Registers on input:
29   #      gfx0    - first word of display list command
30   #      gfx1    - second word of display list command
31   #
32   # RDP command processing is a little different, we only check the
33   # command to identify it as having an address to patch up. We don't
34   # need a switch table with an entry for each command...
35   #
36
37   #
38   # NOTE: We need to detect the FULLSYNC (and other sync?) command
39   # and then spin until some bit in cp0 (?) tells us the RDP pipeline
40   # has drained (about 40 clocks?). Otherwise a race condition is
41   # possible when we write the next output...
42   #
43
```

```
44  .name mask,     $2
45  .name addr,     $19
46  .name   temp,   $8
47
48                  .ent    doRDP
49  doRDP:
50
51  #ifdef LINE3D
52          # check to see if we have a scissor command
53
54                  # shift down the command word
55                  srl     mask,   gfx0,   24
56
57                  # now compare against scissor command
58                  ori     temp,   zero,   G_SETSCISSOR
59                  bne     temp,   mask,   notScissor
60
61                  # we have a scissor command
62
63                  #store YH
64                  andi    temp,   gfx0,   0xfff
65                  sh      temp,   RSP_STATE_SCISSOR_YH(rsp_state)
66
67                  # store XH
68                  srl     temp,   gfx0,   12
69                  andi    temp,   temp,   0xfff
70                  sh      temp,   RSP_STATE_SCISSOR_XH(rsp_state)
71
72                  #store YL
73                  andi    temp,   gfx1,   0xfff
74                  sh      temp,   RSP_STATE_SCISSOR_YL(rsp_state)
75
76                  # store XL
77                  srl     temp,   gfx1,   12
78                  andi    temp,   temp,   0xfff
79                  sh      temp,   RSP_STATE_SCISSOR_XL(rsp_state)
80
81  notScissor:
82
83  #endif
84
85
86          # identify RDP command as having an address or not...
87                  sra     mask, gfx0, 24
88                  addi    mask, mask, G_RDP_ADDR_FIXUP
```

```
89                bltz    mask, doRDPSend
90                addi    mask, mask, (G_RDP_TEXRECT_CHECK-G_RDP_ADDR_FIXUP)
91    ### BRANCH OCCURS to doRDPSend: if no address in command
92          #
93          # RDP command has an address to patch...
94          #
95                jal     AddrFixup
96                add     addr, gfx1, zero              # delay slot
97                add     gfx1, addr, zero
98          #
99          # if RDP command doesn't have an addr, copy it verbatim.
100         #
101
102   .name   outsz,  $18     # used in OutputOpen
103
104   doRDPSend:
105   #if !(defined(OUTPUT_DRAM)||defined(OUTPUT_FIFO))
106               jal     OutputOpen              #
107               addi    outsz, zero, 8          # delay slot
108   #endif /* !(OUTPUT_DRAM || OUTPUT_FIFO) */
109                                               #
110               sw      gfx0, 0(outp)           #
111               sw      gfx1, 4(outp)           #
112                                               #
113               jal     OutputClose             #
114               addi    outp, outp, 8           # delay slot
115                                               #
116               bgtz    mask, GfxDone           # most commands return normally
117               nop                             #
118               j       noYield                 # Textured Rectangle commands
119                                               # do not allow a yield to occur
120                                               # before the second half of the
121                                               # command is sent
122
123    #NOTE! delay slot is in gdma.s
124
125               .end    doRDP
126   .unname outsz
127   .unname mask
128   .unname addr
129   .unname temp
130
131    #
132    #
133    #
```

```
134     ##############################################################################
135
```

gsetup.s

```
 1
 2  /*
 3   * Copyright 1995, Silicon Graphics, Inc.
 4   * ALL RIGHTS RESERVED
 5   *
 6   * UNPUBLISHED -- Rights reserved under the copyright laws of the United
 7   * States.   Use of a copyright notice is precautionary only and does not
 8   * imply publication or disclosure.
 9   *
10   * U.S. GOVERNMENT RESTRICTED RIGHTS LEGEND:
11   * Use, duplication or disclosure by the Government is subject to restrictions
12   * as set forth in FAR 52.227.19(c)(2) or subparagraph (c)(1)(ii) of the Rights
13   * in Technical Data and Computer Software clause at DFARS 252.227-7013 and/or
14   * in similar or successor clauses in the FAR, or the DOD or NASA FAR
15   * Supplement.  Contractor/manufacturer is Silicon Graphics, Inc.,
16   * 2011 N. Shoreline Blvd. Mountain View, CA 94039-7311.
17   *
18   * THE CONTENT OF THIS WORK CONTAINS CONFIDENTIAL AND PROPRIETARY
19   * INFORMATION OF SILICON GRAPHICS, INC. ANY DUPLICATION, MODIFICATION,
20   * DISTRIBUTION, OR DISCLOSURE IN ANY FORM, IN WHOLE, OR IN PART, IS STRICTLY
21   * PROHIBITED WITHOUT THE PRIOR EXPRESS WRITTEN PERMISSION OF SILICON
22   * GRAPHICS, INC.
23   *
24   */
25
26          /****** NOTE:
27           ******
28           ****** This code is the optimized version for HARDWARE 2!
29           ******
30           ****** It won't run on hardware 1.
31           ******
32           ******/
33
34   ############################################################################
35   #
36   # Triangle Setup Routine.
37   # When entering this code we have a points buffer full of points,
38   # and registers r1, r2, r3 point to the three vertices of a triangle.
39   #
40   ############################################################################
41
42  #ifdef SETUP_ALONE
43  #include <rsp.h>
```

```
44  #include "mbi.h"
45
46                  .text   beginSetup
47
48  #include "gdmem.h"
49  #include "gfx_regs.h"
50  #endif
51
52
53   ############################ CLIP TEST ##################################
54  .name   minp,           $1
55  .name   midp,           $2
56  .name   maxp,           $3
57  .name   tmp,            $8
58   #cc,name       rejectmask,     $6
59  .name   tmp2,           $9
60  .name   ccor,           $11     # OR of all points' clip codes
61  .name   ccand,          $12     # AND of all points' clip codes
62
63
64                  .ent    clipAndSetup
65  clipAndSetup:
66   ############################ CLIP TEST ##################################
67
68          lh      ccor, (RSP_PTS_CC)(maxp)        # or Clip Codes together &
69          lh      tmp, (RSP_PTS_CC)(midp) # and Clip Codes together
70          lh      tmp2, (RSP_PTS_CC)(minp)        #
71          and     ccand, ccor, tmp                #
72          or      ccor, ccor, tmp                 #
73          and     ccand, ccand, tmp2              #
74  #ifdef NEAR_CLIP_OFF
75          andi    ccand, ccand, 0x7030            # only see reject + xyz, - xy
76  #else /* NEAR_CLIP_OFF */
77          andi    ccand, ccand, 0x7070            # only see reject +/- xyz
78  #endif /* NEAR_CLIP_OFF */
79          bne     ccand, zero, GfxDone            # Trivial rejection ?
80          or      ccor, ccor, tmp2                #
81   ### BRANCH OCCURS TO GfxDone: IF TRIVIALLY REJECTED
82
83          andi    ccor, ccor, 0x4343              # only see clip/accept +/- xyz
84          bne     ccor, zero, startClip           # if ccor is 0, no clipping
85
86   ### JUMP OCCURS to doClip or startClip: IF clipping is neccessary
87   ### NOTE: delay slot is first instruction of beginSetup:
88
```

```
89                  .end    clipAndSetup
90
91  .unname ccor
92  .unname ccand
93  .unname minp
94  .unname midp
95  .unname maxp
96  .unname tmp
97  .unname tmp2
98   ############################## END CLIP TEST ##############################
99
100
101 #define one4th  vconst[4]
102
103
104 #define LOWX    0       /* used to index elements of edge vectors */
105 #define LOWY    1
106 #define MIDX    2
107 #define MIDY    3
108 #define HIGHX   4
109 #define HIGHY   5
110
111 /* scalar registers: */
112 .name   minp,           $1
113 .name   midp,           $2
114 .name   maxp,           $3
115 .name   miny,           $9
116
117 .name   tmp,            $7
118 .name   flatp,          $4
119 .name   rdp_cmd,        $5
120 .name   rdp_flg,        $6
121 .name   dscratchp,      $8
122 .name   midy,           $10
123 .name   maxy,           $11
124 .name   negR,           $12
125 .name   rendState,      $13
126
127 /* these are "global", used for both edge and attribute setup */
128 .name   DxXDyi,         $v0
129 .name   DxXDyf,         $v1
130 .name   yf,             $v2
131 .name   xHighf,         $v3
132 .name   EDel,           $v4
133 .name   invri,          $v27
```

```
134   .name   invrf,          $v26
135
136   /* these registers are dynamic, allocated and released as they are used */
137   .name   ri,             $v29
138   .name   rf,             $v28
139   .name   Hd,             $v9
140   .name   Md,             $v10
141   .name   Ld,             $v11
142   .name   td,             $v12
143   .name   vmin,           $v13
144   .name   vmid,           $v14
145   .name   vmax,           $v15
146
147   .name   frontrej, $14
148   .name   backrej,  $15
149   .name   doreject, $16
150   .name   bsignr,    $17
151
152   .name   jnk,    $v16
153   .name   t1i,    $v17
154   .name   t1f,    $v18
155   .name   t2i,    $v19
156   .name   t2f,    $v20
157
158   .name   allWi,  $v5
159   .name   allWf,  $v6
160   .name   wscl,   $v21
161
162                   .ent    beginSetup
163
164   beginSetup:
165
166           # load screen coordinates (pre-sort):
167           llv     vmin[0], RSP_PTS_XS(minp)
168           llv     vmid[0], RSP_PTS_XS(midp)       # element 1 is y
169           llv     vmax[0], RSP_PTS_XS(maxp)       # element 0 is x,
170           lw      rendState, RSP_STATE_RENDER(rsp_state)
171
172           addi    dscratchp, zero, RSP_SETUP_TMP_OFFSET
173
174           # load all the W's and Wscale for texture perspective while
175           # doing this.
176           lsv     wscl[0], RSP_STATE_PERSPNORM(rsp_state)
177           lsv     allWi[0], RSP_PTS_W_INT(minp)
178
```

```
179                         vsub    Md, vmid, vmin
180         lsv     allWf[0], RSP_PTS_W_FRAC(minp)
181                         vsub    Hd, vmax, vmin
182         lsv     allWi[2], RSP_PTS_W_INT(midp)
183                         vsub    td, vmin, vmid
184         lsv     allWf[2], RSP_PTS_W_FRAC(midp)
185
186         lsv     allWi[4], RSP_PTS_W_INT(maxp)
187         lsv     allWf[4], RSP_PTS_W_FRAC(maxp)
188
189         # compute the partial products...
190         # careful with the math here...
191                         vmudh   jnk, Hd, Md[1]
192         lh      miny, RSP_PTS_YS(minp)          # get the y's (BEGIN SETUP)
193                         vsar    t1f, t1f, t1f[1]
194         lh      midy, RSP_PTS_YS(midp)
195                         vsar    t1i, t1i, t1i[0]
196         lh      maxy, RSP_PTS_YS(maxp)
197                         vmudh   jnk, td, Hd[1]
198         andi    frontrej, rendState, G_CULL_FRONT
199                         vsar    t2f, t2f, t2f[1]
200         andi    backrej, rendState, G_CULL_BACK
201                         vsar    t2i, t2i, t2i[0]
202         addi    negR, zero, 0
203
204         # begin back-face test:
205         #
206         # Back-face test is the sign of the plane equation BEFORE VERTEX
207         # SORT, tested with the CULL_FRONT or CULL_BACK flags.
208         # We toggle a bit during the sort and possibly correct the
209         # pleq sign afterwards...
210         #
211         # Actual back-face computation is SU code weaved in among
212         # the VU code.
213
214         # y-sort. Remember, input screen coords are S11.2
215         #
216         # This sort code was written by Gudrun Achtenhagen, gudrun@engr.sgi.com
217         # Contact her if there are any bugs. :-)
218         #
219     swap1:      slt     tmp, midy, miny         #if midy>miny, tmp gets 0
220                 blez    tmp, swap2              #if tmp>0, branch
221                 add     tmp, midy, $0           #put midy in tmp
222                 add     midy, miny, $0          #put miny in midy
223                 add     miny, tmp, $0           #put tmp in miny
```

```
224                 addu    tmp, midp, $0           #put midp in tmp
225                 addu    midp, minp, $0          #put minp in midp
226                 addu    minp, tmp, $0           #put tmp in minp
227                 xori    negR, negR, 0x0001
228
229                 .align  8       # ensure dual-issue of branch target
230         swap2:
231                         vaddc   rf, t1f, t2f
232                 slt     tmp, maxy, midy         #if maxy>midy, tmp gets 0
233                         vadd    ri, t1i, t2i
234                 blez    tmp, sortDone           #if tmp>0, branch
235                 add     tmp, maxy, $0           #put maxy in tmp
236                 add     maxy, midy, $0          #put midy in maxy
237                 add     midy, tmp, $0           #put midy in tmp
238                 addu    tmp, maxp, $0           #put maxp in tmp
239                 addu    maxp, midp, $0          #put midp in maxp
240                 addu    midp, tmp, $0           #put tmp in midp
241                 j       swap1
242                 xori    negR, negR, 0x0001
243         sortDone:
244             # this branch target is aligned for dual-issue (see above)
245
246             # load screen coordinates:      (S11.2)
247                         vlt     invri, ri, vconst[0]
248                 llv     vmax[0], RSP_PTS_XS(maxp)       # element 0 is x,
249                         vor     invrf, ri, rf
250                 llv     vmid[0], RSP_PTS_XS(midp)       # element 1 is y
251                 llv     vmin[0], RSP_PTS_XS(minp)
252
253             # possibly negate R
254                 blez    negR, posiR
255                 vsub    EDel, vmax, vmid        # delay slot, low deltas
256                 vmudn   rf, rf, vconst[3]       # negate R
257                 vmadh   ri, ri, vconst[3]
258                 vmadn   rf, vconst, vconst[0]
259             posiR:
260
261             # compute edge deltas:          (S11.2)
262             # (Need to do this again after the sort)
263             # save out vertex pointers for attribute processing
264             # while doing this.
265                                             vsub    Md, vmid, vmin
266             mfc2    bsignr, invri[0]
267                                             vsub    Hd, vmax, vmin
268             mfc2    doreject, invrf[0]
```

```
269
270   .unname jnk
271   .unname t1i
272   .unname t1f
273   .unname t2i
274   .unname t2f
275   .unname td
276   .unname vmin
277   .unname vmid
278   .unname vmax
279
280           # if (r < 0) then triangle is a back-face.
281           # finish back-face processing in the SU.
282
283   /* DELAY HERE! (2 clocks) */
284                           # align these for Newton
285           sra     bsignr, bsignr, 31
286                           vmov    ri[3], ri[0]
287           and     backrej, backrej, bsignr
288                           vmov    rf[3], rf[0]
289
290           # If (r == 0), triangle is NULL, we should bail out completely.
291                           vmov    EDel[MIDX], Md[0]
292           beq     doreject, zero, SetupReject
293           # note delay slot
294
295                           # align these for speed later.
296           xori    bsignr, bsignr, 0xffff
297           # re-test the sign of r *after* the sort for left/right-ness
298                           vlt     invri, ri, vconst[0]
299           and     frontrej, frontrej, bsignr
300                           vmov    EDel[MIDY], Md[1]
301           or      doreject, backrej, frontrej
302                           vmov    EDel[HIGHX], Hd[0]
303           bgtz    doreject, SetupReject
304                           vmov    EDel[HIGHY], Hd[1]
305           mfc2    tmp, invri[0]
306
307   .unname negR
308
309   .unname Hd
310   .unname Md
311   .unname Ld
312
313   .unname ri
```

```
314   .unname rf
315
316   .name   invEDeli,       $v7
317   .name   invEDelf,       $v8
318   .name   EDeli,          $v9
319   .name   EDelf,          $v10
320
321           # these registers are for some attribute (texture) computation
322           # that we are going to sneak in during edge setup.
323   .name   nearWi, $v19
324   .name   nearWf, $v20
325
326           #
327           # compute 1/r
328           # R is about 10 bits accurate coming from the rcp table.
329           # We need to do a Newton's iteration pass here to get more
330           # precision. Each iteration should get another 10 bits...
331           #
332           # (r and invr registers are coordinated with Newton routine)
333           #
334                   jal     NewtonDiv
335                   addi    rdp_flg, zero, 0x80     # delay slot
336
337           # identify left- or right-major triangle:
338           # if (r < 0) dir = 0 else dir = 1
339           #
340                   bltz    tmp, rightMajor
341                   lb      rdp_cmd, RSP_STATE_TRI(rsp_state)       # delay slot
342                   addi    rdp_flg, zero, 0x0      # left-major
343       rightMajor:
344           # Ldx/Ldy, Mdx/Mdy, Hdx/Hdy:
345           #
346           # Since the rcp ROM is 10 bits, that's good enough for
347           # the edge slopes. Newton's doesn't help.
348           #
349           # Get triangle command from state and construct the proper RDP
350           # command while we do this.
351           #
352                           vmudm   EDeli, EDel, vconst[4]  # make S15.16
353                           vmadn   EDelf, vconst, vconst[0]
354
355                           vrcp    invEDelf[LOWY], EDel[LOWY]      # 1.0/Ldy
356                           vrcph   invEDeli[LOWY], vconst[0]
357           ori     rdp_cmd, rdp_cmd, G_TRI_FILL
358
```

```
359         # stick in tile number
360         lb      tmp, RSP_STATE_TEX_TILE(rsp_state)
361                         vrcp    invEDelf[MIDY], EDel[MIDY]      # 1.0/Mdy
362                         vrcph   invEDeli[MIDY], vconst[0]
363
364                         vrcp    invEDelf[HIGHY], EDel[HIGHY]    # 1.0/Hdy
365                         vrcph   invEDeli[HIGHY], vconst[0]
366         or      rdp_flg, rdp_flg, tmp   # 3 cycles after load
367
368         # open for output
369 #if !(defined(OUTPUT_DRAM)||defined(OUTPUT_FIFO))
370                 jal     OutputOpen
371                 addi    $18, zero, 176  # worst case guess (delay slot)
372 #endif /* !(OUTPUT_DRAM || OUTPUT_FIFO) */
373
374         #
375         # We used to shift down the rcp results all the way,
376         # then do the multiply. If we don't shift it down all the
377         # way, do the mult, then shift some more, we get better
378         # precision on the degenerate cases.
379         #
380 #if 0
381                         vmudl   invEDelf, invEDelf, vconst1[2]  # make S15.16
382         sb      rdp_cmd, 0(outp)        # output rdp command
383                         vmadm   invEDeli, invEDeli, vconst1[2]
384         sb      rdp_flg, 1(outp)        # output poly flag
385                         vmadn   invEDelf, vconst, vconst[0]
386 #else
387                         vmudl   invEDelf, invEDelf, vconst1[4]  # make S15.16
388         sb      rdp_cmd, 0(outp)        # output rdp command
389                         vmadm   invEDeli, invEDeli, vconst1[4]
390         sb      rdp_flg, 1(outp)        # output poly flag
391                         vmadn   invEDelf, vconst, vconst[0]
392 #endif
393
394         # Do some other work during the pipeline delay:
395         # We scale up EDel so that later, during the attribute computation,
396         # the 1/r multiply gives us the right S15.16 aligned answer.
397
398                 vmudh   EDel, EDel, vconst[5]   # mult by 4 for attributes
399
400 .name   xi,     $v12
401 .name   xf,     $v13
402
403 .unname frontrej
```

```
404   .unname backrej
405   .unname doreject
406   .unname bsignr
407
408           # x setup:      (S15.16)
409           # we load these into unusual elements so we can group the
410           # multiplies later during the X adjust step...
411           # (finish edge slopes while we do this)
412           # The slope answer will end up in the Y element...
413           # scale W's down to match 1/w for texture perspective,
414           # while doing this.
415
416           lsv     xi[(LOWX*2)],  RSP_PTS_XS(midp)
417                                   vmudl   allWf, allWf, wscl[0]
418           lsv     xi[(MIDX*2)],  RSP_PTS_XS(minp) # same as high
419                                   vmadm   allWi, allWi, wscl[0]
420           lsv     xi[(HIGHX*2)], RSP_PTS_XS(minp)
421                                   vmadn   allWf, vconst, vconst[0]
422
423           # y setup:      (S11.2)
424           sll     tmp, miny, 14   # get frac part of high y-coord
425                           vmudl   DxXDyf, invEDelf, EDelf[0q]     # Ldx / Ldy
426                           vmadm   DxXDyf, invEDeli, EDelf[0q]     # Mdx / Mdy
427                           vmadn   DxXDyf, invEDelf, EDeli[0q]     # Hdx / Hdy
428                           vmadh   DxXDyi, invEDeli, EDeli[0q]
429           mtc2    tmp, yf[0]
430                           vmadn   DxXDyf, vconst, vconst[0]
431           sw      maxp, 0(dscratchp)
432
433   #if 0
434           # no downshift needed
435   #else
436           # shift down some more...
437           #
438           # we may be able to tolerate some slop, and not do a 2-part
439           # shift, once the bow-tie fix is in hardware
440           # Tue May 16 15:14:34 PDT 1995
441           #
442                   vmudl   invEDelf, invEDelf, vconst[4]
443                   vmadm   invEDeli, invEDeli, vconst[4]
444                   vmadn   invEDelf, vconst, vconst[0]
445
446                   vmudl   DxXDyf, DxXDyf, vconst[4]
447                   vmadm   DxXDyi, DxXDyi, vconst[4]
448                   vmadn   DxXDyf, vconst, vconst[0]
```

```
449  #endif
450
451
452
453  .unname EDeli
454  .unname EDelf
455
456  .name vtmp,     $v16
457
458          # clear lower bits of slope fractions to match the edgewalker
459          # only use this chopped frac to back up the starting point.
460          # pass the complete slope to the stepper.
461          sh      maxy, 2(outp)   # output y coords S11.2
462                          vand    vtmp, DxXDyf, vconst1[1]
463
464          # translate S11.2 x's to S15.16.
465          sh      miny, 6(outp)
466                  vmudm   xi, xi, one4th
467          sw      midp, 4(dscratchp)
468                  vmadn   xf, vconst, vconst[0]
469          sw      minp, 8(dscratchp)
470
471          # Check DxXDy for "nearly-horizontal". Make horizontal, if so.
472          # (only a single-precision clamp)
473          sh      midy, 4(outp)
474                          vcr     DxXDyi, DxXDyi, vconst1[6]
475
476  .unname miny
477  .unname midy
478  .unname maxy
479
480          # adjust x's to proper place:
481          #
482          # xHigh = xHigh - DxXHDy * yHigh.frac
483          # xMid  = xHigh - DxXMDy * yHigh.frac
484          # xLow  = xMid                  (already on sub-pixel grid)
485          #
486          # Start the output while we do this...
487          #
488  .name xHighi,   $v9
489  .name t1i,      $v10
490  .name t1f,      $v11
491
492          # Clever use of registers, careful where answer ends up.
493          # Remember, the DxXDy slopes are in the Y elements...
```

```
494          # Do the mult for both equations at once, since we
495          # lined up the registers that way.
496          #
497                  ssv     xi[(LOWX*2)],  8(outp)  # output xLow
498                                  vmudl   t1f, vtmp, yf[0]
499                  ssv     xf[(LOWX*2)], 10(outp)
500                                  vmadm   t1i, DxXDyi, yf[0]
501                  ssv     DxXDyi[(LOWY*2)], 12(outp)
502                                  vmadn   t1f, vconst, vconst[0]
503                  ssv     DxXDyf[(LOWY*2)], 14(outp)
504
505          # do both subtracts at the same time, since we sneakily
506          # lined up xi/xf that way...
507          # find the nearest W while we do this.
508
509          # pre-compute these in the stall delay
510   .name   ptpp,   $15
511   .name   toutp,  $16
512   .name   stmaxi, $v17
513   .name   stmaxf, $v18
514
515                  andi    tmp, rdp_cmd, G_RDP_TRI_TXTR_MASK
516
517          addi    ptpp, dscratchp, 8              # also the loop counter
518          addi    toutp, dscratchp, 16
519
520
521                                  vsubc   xHighf, xf, t1f[1q]
522                  ssv     DxXDyi[(HIGHY*2)], 20(outp)
523                                  vsub    xHighi, xi, t1i[1q]
524                  ssv     DxXDyf[(HIGHY*2)], 22(outp)
525                                  vsubc   wscl,  allWf, allWf[1]
526                  ssv     DxXDyi[(MIDY*2)], 28(outp)
527                                  vlt     nearWi, allWi, allWi[1]
528                  ssv     DxXDyf[(MIDY*2)], 30(outp)
529                                  vmrg    nearWf, allWf, allWf[1]
530                  ssv     xHighi[(HIGHX*2)], 16(outp)     # output xHigh
531                                  vsubc   wscl,  nearWf, allWf[2]
532                  ssv     xHighf[(HIGHX*2)], 18(outp)
533                                  vlt     nearWi, nearWi, allWi[2]
534                  ssv     xHighi[(MIDX*2)],  24(outp)     # output xMid
535                                  vmrg    nearWf, nearWf, allWf[2]
536                  ssv     xHighf[(MIDX*2)],  26(outp)
537                  addi    outp, outp, 32  # increment output pointer
538   .unname xHighi
```

```
539   .unname vtmp
540   .unname t1i
541   .unname t1f
542   .unname xi
543   .unname xf
544
545           #
546           # Begin attribute setup:
547           #
548   /*
549    * at this point, the only registers in use should be:
550    *
551    *      $v0     DxXDyi
552    *      $v1     DxXDyf
553    *      $v2     yf
554    *      $v3     xHighf
555    *      $v4     EDel
556    *      $v27    invri
557    *      $v26    invrf
558    *      $v7     invEDeli
559    *      $v8     invEDelf
560    *
561    * plus these texture things we already computed:
562    *
563    *      $v5     allWi
564    *      $v6     allWf
565    *      $v17    stmaxi
566    *      $v18    stmaxf
567    *      $v19    nearWi
568    *      $v20    nearWf
569    *      $v21    wscl
570    *
571    */
572
573
574           #
575           # Texture setup. If we aren't doing texturing, we can skip
576           # around this.
577           #
578           # This version is unrolled, pipelining two groups of
579           # calculations. This is 23 instructions more, but
580           # 50 clock cycles faster.
581           #
582                   blez    tmp, AttributeSetup
583                   # note delay slot.
```

```
584
585   .name   ptp1,   $14
586   .name   ptp2,   $17
587   .name   ptp3,   $18
588
589   .name   ptTX1i, $v9     # these registers hold S, T, 1/W
590   .name   ptTX1f, $v10    # for each vertex.
591   .name   invW1f, $v11
592   .name   invW1i, $v12
593   .name   vtmpi,  $v15
594   .name   vtmpf,  $v16
595
596   .name   ptTX2i, $v22    # these registers hold S, T, 1/W
597   .name   ptTX2f, $v23    # for each vertex.
598   .name   invW2f, $v24
599   .name   invW2i, $v25
600
601           # make sure nearW < 1.0. The reason we do this is
602           # to safeguard against numerical inaccuracies due to
603           # w divide, etc. If nearW/(nearest W) is not less than
604           # 1.0, the perspective scale will go the wrong direction,
605           # resulting in scrolling textures and wobbles.
606           # We've tried lots of methods to tune this, eventually
607           # settling on a scaling operation; we used to scale by 0.8
608           # but still see wobbles on large texture coordinates (repeated
609           # textures) so we now use 0.5.
610           #
611           # (get the point pointers while doing that)
612                           vmudl   nearWf, nearWf, vconst1[5]
613           lw      ptp1,  0(ptpp)
614                           vmadm   nearWi, nearWi, vconst1[5]
615           lw      ptp2, -4(ptpp)
616                           vmadn   nearWf, vconst, vconst[0]
617           lw      ptp3, -8(ptpp)
618
619           # load S and T:
620                   llv     ptTX1i[0], RSP_PTS_S(ptp1)
621                   llv     ptTX1i[8], RSP_PTS_S(ptp2)
622                   llv     ptTX2i[0], RSP_PTS_S(ptp3)
623
624           # Load 1/W saved from vertex transform.
625           # Stick a magic number in for w. Later during the vmult
626           # this will scale and shift the frac up where we want it
627           # for the attribute calculations.
628                   lsv     invW1f[0], RSP_PTS_INVW_FRAC(ptp1)
```

```
629                 lsv     invW1i[0], RSP_PIS_INVW_INT(ptp1)
630
631                 lsv     invW1f[8], RSP_PIS_INVW_FRAC(ptp2)
632                         vmov    ptTX1i[2], vconst1[0]
633                 lsv     invW1i[8], RSP_PIS_INVW_INT(ptp2)
634                         vmov    ptTX1i[6], vconst1[0]
635
636                 lsv     invW2f[0], RSP_PIS_INVW_FRAC(ptp3)
637                         vmov    ptTX2i[2], vconst1[0]
638                 lsv     invW2i[0], RSP_PIS_INVW_INT(ptp3)
639
640         # normalize the W's:
641         # (this is NW/W)
642         # (we can't cheat here; we need the double-precision multiply
643         # in order to handle all kinds of w's, including orthographic...)
644         # Redundant store of nearW by the SU in the loop saves time later.
645                         vmudl   allWf, invW1f, nearWf[0]
646                         vmadm   allWf, invW1i, nearWf[0]
647         ssv     nearWi[0], 68(dscratchp)
648                         vmadn   allWf, invW1f, nearWi[0]
649         ssv     nearWf[0], 76(dscratchp)
650                         vmadh   allWi, invW1i, nearWi[0]
651
652                         vmudl   vtmpf, invW2f, nearWf[0]
653                         vmadm   vtmpf, invW2i, nearWf[0]
654                         vmadn   nearWf, invW2f, nearWi[0]
655                         vmadh   nearWi, invW2i, nearWi[0]
656
657         # multiply (S, T, W) by normalized 1/W's
658                         vmudm   vtmpf, ptTX1i, allWf[0h]
659                         vmadh   ptTX1i, ptTX1i, allWi[0h]
660                         vmadn   ptTX1f, vconst, vconst[0]
661
662                         vmudm   vtmpf, ptTX2i, nearWf[0]
663                         vmadh   ptTX2i, ptTX2i, nearWi[0]
664                         vmadn   ptTX2f, vconst, vconst[0]
665
666         # output to scratch memory
667         sdv     ptTX1i[8],  (16)(toutp)
668         sdv     ptTX1f[8],  (24)(toutp)
669
670         sdv     ptTX1i[0],  (0)(toutp)
671         sdv     ptTX1f[0],  (8)(toutp)
672
673         sdv     ptTX2i[0],  (32)(toutp)
```

```
674             sdv     ptTX2f[0],  (40)(toutp)
675
676             # single precision, yuk
677                             vabs    ptTX1i, ptTX1i, ptTX1i
678             llv     nearWi[0], (16)(toutp)
679                             vabs    ptTX2i, ptTX2i, ptTX2i
680             llv     nearWf[0], (24)(toutp)
681                             vabs    nearWi, nearWi, nearWi
682
683             # find max S' and T' coordinates for LOD normalization
684                             vge     stmaxi, ptTX1i, ptTX2i
685                             vmrg    stmaxf, ptTX1f, ptTX2f
686
687             /* DELAY HERE! */
688                             vge     stmaxi, stmaxi, nearWi
689                             vmrg    stmaxf, stmaxf, nearWf
690
691             # store of stmax happens below
692
693     .unname ptp1
694     .unname ptp2
695     .unname ptp3
696     .unname ptpp
697     .unname toutp
698
699     .unname ptTX1i
700     .unname ptTX1f
701     .unname ptTX2i
702     .unname ptTX2f
703     .unname allWi
704     .unname allWf
705     .unname vtmpi
706     .unname vtmpf
707     .unname invW1f
708     .unname invW1i
709     .unname invW2f
710     .unname invW2i
711     .unname wscl
712
713
714     .name   Hdai,   $v9
715     .name   Hdaf,   $v10
716     .name   Mdai,   $v11
717     .name   Mdaf,   $v12
718     .name   adei,   $v13
```

```
719   .name   adef,   $v14
720   .name   amin,   $v15
721   .name   aminf,  $v16
722
723   .name   tMdai,  $v21
724   .name   tMdaf,  $v22
725   .name   amid,   $v23
726   .name   amidf,  $v24
727   .name   amax,   $v25
728   .name   amaxf,  $v5
729   .name   vjunk,  $v6
730   .name   vjunkf, $v28
731
732   AttributeSetup:
733
734           # store max S' and T' coordinates for LOD normalization
735                   slv     stmaxi[0], 64(dscratchp)
736                   slv     stmaxf[0], 72(dscratchp)
737   .unname stmaxi
738   .unname stmaxf
739   .unname nearWi
740   .unname nearWf
741
742   .name   ainiti, $v17
743   .name   ainitf, $v18
744   .name   tHdai,  $v19
745   .name   tHdaf,  $v20
746
747           #
748           # If we aren't doing any attributes at all, let's
749           # bail out early.
750           # (clear out all the bits while we do this)
751           #
752                   andi    tmp, rdp_cmd, (G_RDP_TRI_ZBUFF_MASK | G_RDP_TRI_TXTR_MASK  | G_RDP_TRI_SHADE_MASK)
753                   blez    tmp, SetupDone
754                           vxor    ainitf, vconst, vconst
755
756           #
757           # Collect all the attributes
758           # in a vector (r,g,b,a,s,t,w,z) with one left over (l).
759           # l (and z again) are computed in a second pass.
760
761           # load attributes:
762           # load smooth-shading colors first.
763           # RGBA, use fancy packed load, then shift.
```

```
764         # DMEM alignment is crucial here!
765
766         # add .5 to the colors in order to work around a hardware
767         # bug regarding span start color imprecision
768         # Thu Jun  8 18:49:52 PDT 1995
769
770                 luv     amax[0], RSP_PTS_R_NX(maxp)
771                                 vadd    aminf, ainitf, vconst1[5]
772                 luv     amin[0], RSP_PTS_R_NX(minp)
773                                 vadd    amidf, ainitf, vconst1[5]
774                 andi    tmp, rendState, G_SHADING_SMOOTH
775                                 vadd    amaxf, ainitf, vconst1[5]
776
777         # test for flat shading
778                 bgtz    tmp, smoothShade
779                 luv     amid[0], RSP_PTS_R_NX(midp)     # delay slot
780
781         # load flat-shading colors instead: (use same vertex)
782                 luv     amax[0], RSP_PTS_R_NX(flatp)
783                 luv     amin[0], RSP_PTS_R_NX(flatp)
784                 luv     amid[0], RSP_PTS_R_NX(flatp)
785
786         smoothShade:
787                 vmudm   amax, amax, vconst[7]   # multiply by 1/512.0 to
788                 vmudm   amin, amin, vconst[7]   # move things into lower byte.
789                 vmudm   amid, amid, vconst[7]
790
791         # load S, T, and W:
792         # These have been previously computed and stored in scratch memory.
793                 ldv     aminf[8], (16 +  8)(dscratchp)
794                 ldv     amin[8],  (16 +  0)(dscratchp)
795                 ldv     amidf[8], (16 + 24)(dscratchp)
796                 ldv     amid[8],  (16 + 16)(dscratchp)
797                 ldv     amaxf[8], (16 + 40)(dscratchp)
798                 ldv     amax[8],  (16 + 32)(dscratchp)
799
800         # load z's.
801         # Use the proper 'screen-space' Z.
802                 lsv     aminf[14], RSP_PTS_ZSF(minp)
803                 lsv     amin[14], RSP_PTS_ZS(minp)
804                 lsv     amidf[14], RSP_PTS_ZSF(midp)
805                 lsv     amid[14], RSP_PTS_ZS(midp)
806                 lsv     amaxf[14], RSP_PTS_ZSF(maxp)
807                 lsv     amax[14], RSP_PTS_ZS(maxp)
808
```

```
809         # compute attribute deltas: (S15.16) watch alignment!
810 /* DELAY HERE! */
811                 vsubc   Mdaf, amidf, aminf
812                 vsub    Mdai, amid, amin
813                 vsubc   tHdaf, aminf, amaxf
814                 vsub    tHdai, amin, amax
815                 vsubc   Hdaf, amaxf, aminf
816                 vsub    Hdai, amax, amin
817                 vsubc   tMdaf, aminf, amidf
818                 vsub    tMdai, amin, amid
819
820         #
821         # These multiplies use the full precision of the accumulator.
822         # They are basically 32-bit integer multiplies, but the
823         # fractional component is also included, although only the
824         # upper 32-bits of answer are used.
825         #
826         # See note up above about why EDel is being scaled up.
827         #
828         # S15.16 * S11.4 = SS26.20
829         # (we only use the upper SS26.4, which we'll multiply
830         # by 1/r below)
831         #
832                 # DxAtt = Mdy*Hda - Hdy*Mda
833                 vmudn   vjunk, Hdaf, EDel[MIDY]
834                 vmadh   vjunk, Hdai, EDel[MIDY]
835                 vmadn   vjunk, tMdaf, EDel[HIGHY]
836                 vmadh   vjunk, tMdai, EDel[HIGHY]
837                 vsar    Hdai, Hdai, Hdai[0]
838                 vsar    Hdaf, Hdaf, Hdaf[1]
839
840                 # DyAtt = Hdx*Mda - Mdx*Hda
841                 vmudn   vjunk, Mdaf, EDel[HIGHX]
842                 vmadh   vjunk, Mdai, EDel[HIGHX]
843                 vmadn   vjunk, tHdaf, EDel[MIDX]
844                 vmadh   vjunk, tHdai, EDel[MIDX]
845                 vsar    Mdai, Mdai, Mdai[0]
846                 vsar    Mdaf, Mdaf, Mdaf[1]
847
848         # divide by r (S4.27)
849         #
850         # This multiply results in the proper S15.16 attributes
851         # that we need (texture is S10.21)
852         #
853         # begin storing colors as we do this:
```

```
854         #
855                         vmudl   vjunk, Hdaf, invrf[3]
856                         vmadm   vjunk, Hdai, invrf[3]
857                         vmadn   Hdaf,  Hdaf, invri[3]
858                         vmadh   Hdai,  Hdai, invri[3]
859
860                         vmudl   vjunk, Mdaf, invrf[3]
861                         vmadm   vjunk, Mdai, invrf[3]
862                         vmadn   Mdaf,  Mdaf, invri[3]
863         sdv     Hdai[0],    8(outp)
864                         vmadh   Mdai,  Mdai, invri[3]
865         sdv     Hdaf[0],   24(outp)
866
867                 # convert to edge slope representation:
868                 #   de = dy + dx * DxXHDy
869 /* DELAY HERE! */
870                         vmudn   vjunk, Mdaf, vconst[1]
871                         vmadh   vjunk, Mdai, vconst[1]  # use accum for add...
872                         vmadl   vjunk, Hdaf, DxXDyf[HIGHY]
873                         vmadm   vjunk, Hdai, DxXDyf[HIGHY]
874                         vmadn   adef,  Hdaf, DxXDyi[HIGHY]
875         sdv     Mdai[0],   40(outp)
876                         vmadh   adei,  Hdai, DxXDyi[HIGHY]
877         sdv     Mdaf[0],   56(outp)
878
879 .unname vjunk
880 .unname vjunkf
881 .name   pp1i,   $v6
882 .name   pp1f,   $v28
883                 # attribute X adjust:
884                 #   att = att - (de * yHigh.frac)
885 /* DELAY HERE! */
886                                 vmudl   pp1f, adef,   yf[0]
887                 sdv     adei[0],   32(outp)
888                                 vmadm   pp1i, adei,   yf[0]
889                 sdv     adef[0],   48(outp)
890                                 vmadn   pp1f, vconst, vconst[0]
891 /* DELAY HERE! */
892                                 vsubc   ainitf, aminf, pp1f
893                                 vsub    ainiti, amin,   pp1i    # delay slot
894 .unname pp1i
895 .unname pp1f
896
897                 andi    tmp, rdp_cmd, G_RDP_TRI_SHADE_MASK
898         #
```

```
899          # All done.
900          # Write out the proper record to the RDP, based on the drawing
901          # modes.
902          #
903          # (get ready for next test in the branch delay slots)
904
905          # write out the rest of shade and increment outp
906                  blez    tmp, outputTXTR
907                  andi    tmp, rdp_cmd, G_RDP_TRI_TXTR_MASK       # delay
908
909                  addi    outp, outp, 64  # increment output pointer
910                  sdv     ainiti[0], (0-64)(outp) # 0
911                  sdv     ainitf[0], (16-64)(outp)        # 16
912
913          # write out texture
914    outputTXTR:   blez    tmp, outputZBUF
915                  andi    tmp, rdp_cmd, G_RDP_TRI_ZBUFF_MASK      # delay
916          #
917          # Scale texture parameters to ensure that they all remain
918          # in-bounds for the hardware LOD computation:
919          #
920
921          # free up some registers
922   .unname amaxf
923   .unname tHdai
924   .unname tHdaf
925   .unname tMdai
926   .unname tMdaf
927   .unname amid
928   .unname amidf
929   .unname amax
930   .unname invri
931   .unname invrf
932
933   .name scalei,   $v5
934   .name scalef,   $v6
935   .name vtmpf,    $v19
936   .name coordMi,  $v20
937   .name coordMf,  $v21
938   .name t1i,      $v22
939   .name t1f,      $v23
940   .name absdxi,   $v24
941   .name absdyi,   $v25
942   .name absdxf,   $v26
943   .name absdyf,   $v27
```

```
944
945         # shift >> 5 to get some guard bits for this computation:
946         addi    $16, zero, 0x0800
947         mtc2    $16, vtmpf[0]
948
949         # find abs() of all the slopes:
950         # (sloppy, single-precision test only)
951         # use the original fractional vector when needed
952         # load maxS', maxT', and nearW into vector
953                         vabs    absdxi, Hdai, Hdai
954         ldv     coordMi[8], 64(dscratchp)
955                         vabs    absdyi, Mdai, Mdai
956         ldv     coordMf[8], 72(dscratchp)
957
958         # shift >> 5 to get some guard bits for this computation:
959                         vmudm   absdxi, absdxi, vtmpf[0]
960                         vmadn   absdxf, vconst, vconst[0]
961
962                         vmudm   absdyi, absdyi, vtmpf[0]
963                         vmadn   absdyf, vconst, vconst[0]
964
965                         vmudl   coordMf, coordMf, vtmpf[0]
966                         vmadm   coordMi, coordMi, vtmpf[0]
967                         vmadn   coordMf, vconst, vconst[0]
968
969                         vmudn   absdxf, absdxf, vconst[2]
970                         vmadh   absdxi, absdxi, vconst[2]
971                         vmadn   absdxf, vconst, vconst[0]
972
973         # compute |coordMax| + 2*|d?dx| + |d?dy|
974         # add using the accumulator:
975                         vmadn   t1f, absdyf, vconst[1]
976                         vmadh   t1i, absdyi, vconst[1]
977
978         addi    $16, zero, 0x0040
979                         vmadn   scalef, coordMf, vconst[1]
980         mtc2    $16, vtmpf[0]
981                         vmadh   scalei, coordMi, vconst[1]
982
983         # find max of scale factors
984                         vsubc   t1f, scalef, scalef[5]
985                         vge     scalei, scalei, scalei[5]
986                         vmrg    scalef, scalef, scalef[5]
987                         vsubc   t1f, scalef, scalef[6]
988                         vge     scalei, scalei, scalei[6]
```

```
 989                         vmrg    scalef, scalef, scalef[6]
 990
 991         # shift >> 10 to get the scale ratio:
 992                         vmudl   scalef, scalef, vtmpf[0]
 993                         vmadm   scalei, scalei, vtmpf[0]
 994                         vmadn   scalef, vconst, vconst[0]
 995
 996         # compute 1/scalefactor
 997         # sloppy, Newton's not needed.
 998                         vrcph   t1f[0], scalei[4]
 999                         vrcpl   scalef[0], scalef[4]
1000                         vrcph   scalei[0], vconst[0]
1001
1002         # convert to s15.16
1003                         vmudn   scalef, scalef, vconst[2]
1004                         vmadh   scalei, scalei, vconst[2]
1005
1006         # if scale > 1.0, make 1.0 (don't want to scale)
1007                         vlt     scalei, scalei, vconst[1]
1008                         vmrg    scalef, scalef, vconst[0]
1009
1010  .unname absdxi
1011  .unname absdyi
1012  .unname absdxf
1013  .unname absdyf
1014  .unname coordMi
1015  .unname coordMf
1016  .unname t1i
1017  .unname t1f
1018  .unname vtmpf
1019
1020  .name tiniti,   $v19
1021  .name tinitf,   $v20
1022  .name tHdai,    $v21
1023  .name tHdaf,    $v22
1024  .name tMdai,    $v23
1025  .name tMdaf,    $v24
1026  .name tadei,    $v25
1027  .name tadef,    $v26
1028
1029         # scale init, dx, dy, de
1030  /* DELAY HERE! */
1031                         vmudl   tinitf, ainitf, scalef[0]
1032                         vmadm   tinitf, ainiti, scalef[0]
1033                         vmadn   tinitf, ainitf, scalei[0]
```

```
1034                        vmadh   tiniti, ainiti, scalei[0]
1035
1036                        vmudl   tHdaf, Hdaf, scalef[0]
1037                        vmadm   tHdaf, Hdai, scalef[0]
1038                        vmadn   tHdaf, Hdaf, scalei[0]
1039        sdv     tiniti[8],  0(outp)
1040                        vmadh   tHdai, Hdai, scalei[0]
1041        sdv     tinitf[8], 16(outp)
1042
1043                        vmudl   tMdaf, Mdaf, scalef[0]
1044                        vmadm   tMdaf, Mdai, scalef[0]
1045                        vmadn   tMdaf, Mdaf, scalei[0]
1046        sdv     tHdai[8],    8(outp)
1047                        vmadh   tMdai, Mdai, scalei[0]
1048        sdv     tHdaf[8],   24(outp)
1049
1050                        vmudl   tadef, adef, scalef[0]
1051                        vmadm   tadef, adei, scalef[0]
1052                        vmadn   tadef, adef, scalei[0]
1053        sdv     tMdai[8],   40(outp)
1054                        vmadh   tadei, adei, scalei[0]
1055        sdv     tMdaf[8],   56(outp)
1056
1057        # write out the rest of texture parameters and increment outp:
1058                addi    outp, outp, 64  # increment output pointer
1059                sdv     tadei[8],   (32-64)(outp)       # 32
1060                sdv     tadef[8],   (48-64)(outp)       # 48
1061
1062  .unname scalei
1063  .unname scalef
1064  .unname tiniti
1065  .unname tinitf
1066  .unname tHdai
1067  .unname tHdaf
1068  .unname tMdai
1069  .unname tMdaf
1070  .unname tadei
1071  .unname tadef
1072
1073    outputZBUF:
1074                blez    tmp, SetupDone
1075                # note delay slot
1076        #
1077        # Scale Z-values up, screen coordinates were limited
1078        # to 10 integer bits, but the hardware floating point format
```

```
1079          # needs valid bits in the upper range for best performance.
1080          #
1081                                  vmudn   adef, adef, vconst1[4]
1082                                  vmadh   adei, adei, vconst1[4]
1083                                  vmadn   adef, vconst, vconst[0]
1084
1085                                  vmudn   aminf, aminf, vconst1[4]
1086                                  vmadh   amin, amin, vconst1[4]
1087                                  vmadn   aminf, vconst, vconst[0]
1088
1089          ssv     adei[14],    8(outp)    # output z stuff.
1090                                  vmudn   Hdaf, Hdaf, vconst1[4]
1091          ssv     adef[14],   10(outp)
1092                                  vmadh   Hdai, Hdai, vconst1[4]
1093                                  vmadn   Hdaf, vconst, vconst[0]
1094
1095                                  vmudn   Mdaf, Mdaf, vconst1[4]
1096                                  vmadh   Mdai, Mdai, vconst1[4]
1097                                  vmadn   Mdaf, vconst, vconst[0]
1098
1099          # clamp dzdy if near zero, for decal mode:
1100                  lbu     tmp, RSP_STATE_TEX_LOD(rsp_state)
1101                  sub     tmp, zero, tmp
1102                  beq     tmp, zero, noZClamp
1103                  mtc2    tmp, $v6[0]
1104                  vch     Mdai, Mdai, $v6[0]
1105                  vcl     Mdaf, Mdaf, vconst[0]
1106
1107  .name   pp1i,   $v6
1108  .name   pp1f,   $v28
1109
1110      noZClamp:
1111                  # re-compute attribute X adjust after the Z scale:
1112                  #   att = att - (de * yHigh.frac)
1113          ssv     Hdai[14],    4(outp)
1114                                  vmudl   pp1f, adef,   yf[0]
1115          ssv     Hdaf[14],    6(outp)
1116                                  vmadm   pp1i, adei,   yf[0]
1117          ssv     Mdai[14],   12(outp)
1118                                  vmadn   pp1f, vconst, vconst[0]
1119          ssv     Mdaf[14],   14(outp)
1120                                  vsubc   ainitf, aminf, pp1f
1121                                  vsub    ainiti, amin,   pp1i
1122  .unname pp1i
1123  .unname pp1f
```

```
1124                addi    outp, outp, 16  # increment output pointer
1125                ssv     ainiti[14],  (0-16)(outp)       # 0
1126                ssv     ainitf[14],  (2-16)(outp)       # 2
1127
1128    SetupDone:                      # done or rejected. do any clean-up.
1129                jal     OutputClose
1130                # note delay slot
1131
1132    SetupReject:    # no OutputClose needed...
1133                nop
1134
1135                jr      return_save
1136                nop
1137
1138                .end    beginSetup
1139
1140    /* un-name scalar registers: */
1141    .unname minp
1142    .unname midp
1143    .unname maxp
1144    .unname flatp
1145    .unname rdp_cmd
1146    .unname rdp_flg
1147    .unname tmp
1148    .unname dscratchp
1149    .unname rendState
1150
1151    /* un-name vector registers: */
1152    .unname DxXDyi
1153    .unname DxXDyf
1154    .unname yf
1155    .unname xHighf
1156    .unname EDel
1157    .unname invEDeli
1158    .unname invEDelf
1159
1160    .unname Hdai
1161    .unname Hdaf
1162    .unname Mdai
1163    .unname Mdaf
1164    .unname adei
1165    .unname adef
1166    .unname ainiti
1167    .unname ainitf
1168    .unname amin
```

```
1169    .unname aminf
1170
```

gvtx.s

```
 1
 2  /*
 3   * Copyright 1995, Silicon Graphics, Inc.
 4   * ALL RIGHTS RESERVED
 5   *
 6   * UNPUBLISHED -- Rights reserved under the copyright laws of the United
 7   * States.   Use of a copyright notice is precautionary only and does not
 8   * imply publication or disclosure.
 9   *
10   * U.S. GOVERNMENT RESTRICTED RIGHTS LEGEND:
11   * Use, duplication or disclosure by the Government is subject to restrictions
12   * as set forth in FAR 52.227.19(c)(2) or subparagraph (c)(1)(ii) of the Rights
13   * in Technical Data and Computer Software clause at DFARS 252.227-7013 and/or
14   * in similar or successor clauses in the FAR, or the DOD or NASA FAR
15   * Supplement.  Contractor/manufacturer is Silicon Graphics, Inc.,
16   * 2011 N. Shoreline Blvd. Mountain View, CA 94039-7311.
17   *
18   * THE CONTENT OF THIS WORK CONTAINS CONFIDENTIAL AND PROPRIETARY
19   * INFORMATION OF SILICON GRAPHICS, INC. ANY DUPLICATION, MODIFICATION,
20   * DISTRIBUTION, OR DISCLOSURE IN ANY FORM, IN WHOLE, OR IN PART, IS STRICTLY
21   * PROHIBITED WITHOUT THE PRIOR EXPRESS WRITTEN PERMISSION OF SILICON
22   * GRAPHICS, INC.
23   *
24   */
25
26  #include "gfx_regs.h"
27
28   ##############################################################################
29   #
30   # Transform, project, and viewport map as we load into the points buffer.
31   #
32   # $1 holds n and v0
33   # in_bufp holds pointer to data
34   #
35   # This version does 2 points at a time, 2 per vector register,
36   # (could use software pipelining to do two more).
37   #
38   # WARNING: many of the constants, pipelining, etc. reflect the
39   # layout of the points buffer, etc. Be careful.
40   #
41         # v0 field done in the delay slot that brought us here.
42                 srl     $1, $1, 4       # pick off 'n' field
43                 addi    $5, $1, 1       # n is really (n-1)
```

```
44
45    # these can't be moved up because of the label/directive problem...
46    # don't use regs 1-4, so we can call MatCat...
47   .name gmode,    $3
48
49   .name n,        $5
50   .name v0,       $6
51   .name voutp,    $7
52   .name tmp,      $8
53   .name i,        $9
54   .name flg1,     $13
55   .name flg2,     $14
56   .name clr1,     $15
57   .name clr2,     $16
58
59   .name vpscale,  $v0
60   .name vptrans,  $v1
61   .name vin12,    $v2
62   .name vtmp,     $v3
63   .name persp12i, $v4
64   .name persp12f, $v5
65   .name scrn12i,  $v6
66   .name scrn12f,  $v7
67
68   .name mtx0,     $v8
69   .name mtx1,     $v9
70   .name mtx2,     $v10
71   .name mtx3,     $v11
72   .name mtf0,     $v12
73   .name mtf1,     $v13
74   .name mtf2,     $v14
75   .name mtf3,     $v15
76   .name txtscale, $v17
77   .name st12,     $v18
78   .name wscl,     $v19
79
80   .name voutbaki, $v21
81   .name voutbakf, $v20
82   .name vout12i,  $v29
83   .name vout12f,  $v28
84   .name invW12i,  $v27
85   .name invW12f,  $v26
86
87                   addi    i, n, 0                 # initialize loop counter
88                   ldv     vin12[0],  0(in_bufp)   # load first vert to transform
```

```
89                  ldv     vin12[8], 16(in_bufp)   # load second vert to transform
90
91          # handle loads where v0 is not zero...
92                  addi    voutp, zero, RSP_POINTS_OFFSET
93                  sll     tmp, v0, 5      # offset = v0 * sizeof(point_buffer)
94                  sll     v0, v0, 3
95                  add     tmp, v0, tmp    # (v0*32)+(v0*8)
96
97                  jal     getScaleTrans           # get screen scale & translate
98                  add     voutp, voutp, tmp       # delay slot
99
100         # get texture coordinate scale; jump to matrix load routine
101                 llv     txtscale[0], RSP_STATE_TEX_SCALE_S(rsp_state)
102                 jal     getMatrix
103                 llv     txtscale[8], RSP_STATE_TEX_SCALE_S(rsp_state)
104  ### JUMP OCCURS to getMatrix: subroutine
105
106
107 xfm_loop:
108         # do MP matrix multiplication:
109         # This is clever. We multiply each ROW of the matrix
110         # by one of the scalar point coordinates, using the
111         # accumulator to sum up the matrix columns.
112         #
113         # (could call MatCat, slower but less code...)
114         #
115         # do SU loads (colors and texture coords) while we do this...
116         #
117                 vmudn   vout12f, mtf0, vin12[0h]
118         llv     st12[0], ( 0+8)(in_bufp)
119                 vmadh   vout12f, mtx0, vin12[0h]
120         lw      clr1, ( 0+12)(in_bufp)
121                 vmadn   vout12f, mtf1, vin12[1h]
122         lw      clr2, (16+12)(in_bufp)
123                 vmadh   vout12f, mtx1, vin12[1h]
124 .name lighton,  $1
125         andi    lighton,gmode,(G_LIGHTING_H)
126                 vmadn   vout12f, mtf2, vin12[2h]
127                 vmadh   vout12f, mtx2, vin12[2h]
128                 vmadn   vout12f, mtf3, vconst[1]        # w = 1.0
129         llv     st12[8], (16+8)(in_bufp)
130                 vmadh   vout12i, mtx3, vconst[1]
131
132 #ifdef FASTLIGHT3D
133         bne     lighton, zero, doLight          # do lighting here if on
```

```
134  #endif /* FASTLIGHT3D */
135          addi    in_bufp, in_bufp, 32            # start working on next vtx
136
137   ### BRANCH OCCURS TO doLight: subroutine if lighting is on
138
139  .unname lighton
140
141  lightReturn:
142          # scale texture coordinates:
143                  vmudm   st12, st12, txtscale            # apply scale
144
145  screenCalc:
146
147
148          # save the vout12i and vout12f into voutbaki and voutbakf
149          # scale down 'w' by wscl[0] to keep 1/w reasonably large (DxF -> D)
150          # do SU load (vertex flags) during this compute.
151          # do CLIP store during this compute.
152          # CLIP CODES are 0000 0zyx 0000 0ZYX where zyx=-z,-y,-x and ZYX=+z,+y,+x
153          # The high 16 bit clipcodes are for trivial rejection and the low 16 bit
154          # clipcodes are for clipping and trivial accepting
155
156          lsv     voutbaki[0], (CLIP_SELECT+6)(zero)      # multiplier
157
158
159
160  #ifdef NEAR_CLIP_OFF
161    /* draw stuff even when it is in front of the near clipping plane */
162                  vmudn   voutbakf, vout12f, voutbaki[0]  # mult x frustrum
163          mfc2    flg1, vout12i[3]                        #n w comp of vtx 1
164          andi    flg1, flg1, 0x8000                      #n  neg bit
165          srl     flg1, flg1, 13                          #n  in place
166                  vmadh   voutbaki, vout12i, voutbaki[0]  #
167          mfc2    flg2, vout12i[7]                        # w comp of vtx 2
168          andi    flg2, flg2, 0x8000                      #  neg bit
169          srl     flg2, flg2, 9                           #  in place
170
171                  vch     vtmp, vout12i, vout12i[3h]      # rejection clipcodes
172          or      flg2, flg2, flg1                        #n  vtx 1 & 2
173                  vcl     vtmp, vout12f, vout12f[3h]      # rejection clipcodes
174          cfc2    flg1, $vcc                              # rejection clipcodes
175
176                  vch     vtmp, vout12i, voutbaki[3h]     # clip/accept clipcodes
177                  vcl     vtmp, vout12f, voutbakf[3h]     # clip/accept clipcodes
178
```

```
179         andi    tmp,  flg1, 0x0703              #n reject clipcodes - vtx 1
180         andi    flg1, flg1, 0x7030              #n reject clipcodes - vtx 2
181         or      flg1, flg1, flg2                #n complete with nearclip
182         sll     tmp,  tmp,  4                   # vtx 1 in 0x00007070
183         sll     flg1, flg1, 16                  # vtx 2 in 0x70700000
184         or      flg1, flg1, tmp                 # all reject clipcodes in place
185
186   #else /* NEAR_CLIP_OFF */
187     /* Don't draw anything in front of the near clipping plane */
188                 vmudn   voutbakf, vout12f, voutbaki[0]  # mult x frustrum
189                 vmadh   voutbaki, vout12i, voutbaki[0]  #
190
191                 vch     vtmp, vout12i, vout12i[3h]      # rejection clipcodes
192                 vcl     vtmp, vout12f, vout12f[3h]      # rejection clipcodes
193         cfc2    flg1, $vcc                              # rejection clipcodes
194
195                 vch     vtmp, vout12i, voutbaki[3h]     # clip/accept clipcodes
196                 vcl     vtmp, vout12f, voutbakf[3h]     # clip/accept clipcodes
197
198         andi    tmp,  flg1, 0x0707              # reject clipcodes - vtx 1
199         andi    flg1, flg1, 0x7070              # reject clipcodes - vtx 2
200         sll     tmp,  tmp,  4                   # vtx 1 in 0x00007070
201         sll     flg1, flg1, 16                  # vtx 2 in 0x70700000
202         or      flg1, flg1, tmp                 # all reject clipcodes in place
203
204   #endif /* NEAR_CLIP_OFF */
205
206
207         cfc2    flg2, $vcc                      # clip/accept clipcodes
208         andi    tmp,  flg2, 0x0707              # clip/accept clipcodes - vtx 1
209                                         vadd    voutbaki, vout12i, vconst[0]
210         andi    flg2, flg2, 0x7070              # clip/accept clipcodes - vtx 2
211                                         vadd    voutbakf, vout12f, vconst[0]
212         sll     flg2, flg2, 12                  # vtx 2 in 0x07070000
213                                         vmudl   vout12f, vout12f, wscl[0]
214         or      tmp,  tmp,  flg2        # all clip/accept clipcodes in place
215                                         vmadm   vout12i, vout12i, wscl[0]
216         or      tmp,  tmp,  flg1        # all clipcodes in place
217                                         vmadn   vout12f, vconst, vconst[0]
218         sh      tmp, ( 0+RSP_PIS_CC)(voutp)
219
220
221         # project (divide by w)
222         # W and invW registers are aligned for the Newton's call...
223                 jal     NewtonDiv
```

```
224                 lh      flg1, ( 0+6-32)(in_bufp)        # delay slot
225
226         # max 1/w if negative for screen calc
227                                     vge     scrn12i, invW12i, vconst[0]
228         sdv     voutbaki[0], ( 0+RSP_PTS_X_INT)(voutp)
229                                     vmrg    scrn12i, invW12i, vconst1[0]
230         sdv     voutbakf[0], ( 0+RSP_PTS_X_FRAC)(voutp)
231
232                                     vmudl   persp12f, voutbakf, invW12f[3h]
233                                     vmadm   persp12f, voutbaki, invW12f[3h]
234                                     vmadn   persp12f, voutbakf, scrn12i[3h]
235                                     vmadh   persp12i, voutbaki, scrn12i[3h]
236
237         # image space (viewport scale and translate)
238         # increment input pointer and load next verts to transform
239         # while we compute...
240         #
241         # The viewport scale and translate has a built-in multiplier
242         # of 4.0 which converts screen coords to S11.2.
243         #
244         addi    i, i, -1
245
246         # scale down x,y,z to compensate for prev scaling down of w (DxF -> D)
247                         vmudl   persp12f, persp12f, wscl[0]
248                         vmadm   persp12i, persp12i, wscl[0]
249                         vmadn   persp12f, vconst, vconst[0]
250
251   .name fogon,    $12
252         andi    fogon,gmode,G_FOG_H     # fog on?
253         ldv     vin12[0],  0(in_bufp)
254                                     # screen translate
255                                     vmudh   scrn12f, vptrans, vconst[1]
256         ldv     vin12[8], 16(in_bufp)
257                                     # screen scale
258                                     vmadn   scrn12f, persp12f, vpscale
259         # Clamp x,y,z to +/- 2K boundary (Z clamps 0...0x7FFF)
260         ldv     vout12i[0], VCONST_SCREENCLAMP(zero)
261                                     vmadh   scrn12i,  persp12i, vpscale
262         ldv     vout12i[8], VCONST_SCREENCLAMP(zero)
263                                     vmadn   scrn12f, vconst, vconst[0]
264
265                 vge     scrn12i,scrn12i,vout12i[1q]     # clamp xy -0x3fe, z 0
266         sw      clr1,       ( 0+RSP_PTS_R_NX)(voutp)
267         beq     fogon,zero,storeVertex  # skip fog if turned off
268                 vlt     scrn12i,scrn12i,vout12i[0q]     # clamp xy 0x3fe, z max
```

```
269
270   ### BRANCH OCCRS to storeVertex: if fog is off
271
272   #
273   # FOG
274   # screen z is a s10.16 number in scrn12i, scrn12f at this point
275   #
276           lqv     vtmp[0], RSP_FOG_FACTOR(zero)
277           vmudh   persp12f, persp12f, vtmp[0]
278           vmadh   persp12i, persp12i, vtmp[0]
279           vadd    persp12i, persp12i, vtmp[1]
280           vge     persp12i, persp12i, vconst[0]   # clamp: min=0x00
281           vlt     persp12i, persp12i, vtmp[2]     # clamp: max=0xff
282
283           sbv     persp12i[5],(0+RSP_PTS_R_NX+3)(voutp)  # store forg into vtx2 color
284           sw      clr2, ( 0+RSP_PTS_XS)(voutp)       # temp storage place
285           sbv     persp12i[13],( 0+RSP_PTS_XS+3)(voutp)  # insert fog into vtx1 color
286           lw      clr2, ( 0+RSP_PTS_XS)(voutp)       # load color with fog
287
288   .unname fogon
289
290
291       # output transformed vertex information:
292       #   3D points for clipping, w
293       #   screen points
294       #   vertex flag
295       #   clip code
296       #   texture coord
297       #   colors/normals
298
299           # always write the rest of the first point: (weird order to
300           # avoid stalls)
301
302   storeVertex:
303                   slv     st12[0],    ( 0+RSP_PTS_S)(voutp)
304                   sdv     scrn12i[0], ( 0+RSP_PTS_XS)(voutp)
305                   ssv     scrn12f[4], ( 0+RSP_PTS_ZSF)(voutp)
306                   ssv     invW12i[6], ( 0+RSP_PTS_INVW_INT)(voutp)
307                   ssv     invW12f[6], ( 0+RSP_PTS_INVW_FRAC)(voutp)
308
309           # maybe write the 2nd point:
310                   blez    i, xfm_done
311                   addi    i, i, -1
312                   sdv     voutbaki[8], (40+RSP_PTS_X_INT)(voutp)
313                   sdv     voutbakf[8], (40+RSP_PTS_X_FRAC)(voutp)
```

```
314                 slv     st12[8],    (40+RSP_PTS_S)(voutp)
315                 sw      clr2,       (40+RSP_PTS_R_NX)(voutp)
316                 sdv     scrn12i[8], (40+RSP_PTS_XS)(voutp)
317                 ssv     scrn12f[12], (40+RSP_PTS_ZSF)(voutp)
318                 ssv     invW12i[14], (40+RSP_PTS_INVW_INT)(voutp)
319                 ssv     invW12f[14], (40+RSP_PTS_INVW_FRAC)(voutp)
320                 sw      tmp,        (40+RSP_PTS_CC)(voutp)
321
322         # prepare for the next pair of points
323                 addi    voutp, voutp, 80        # increment output pointer
324                                                 # (with padding)
325                 bgtz    i, xfm_loop     # for (i=n_pts; i>0; i--) {
326
327         # transform normals and shade
328 .unname gmode
329
330 .unname n
331 .unname v0
332 .unname voutp
333 .unname tmp
334 .unname i
335
336 .unname flg1
337 .unname flg2
338 .unname clr1
339 .unname clr2
340
341 .unname vpscale
342 .unname vptrans
343 .unname vin12
344 .unname voutbaki
345 .unname voutbakf
346 .unname vout12i
347 .unname vout12f
348 .unname invW12i
349 .unname invW12f
350 .unname vtmp
351 .unname persp12i
352 .unname persp12f
353 .unname scrn12i
354 .unname scrn12f
355
356 .unname mtx0
357 .unname mtx1
358 .unname mtx2
```

```
359   .unname mtx3
360   .unname mtf0
361   .unname mtf1
362   .unname mtf2
363   .unname mtf3
364   .unname txtscale
365   .unname st12
366   .unname wscl
367
368    #
369    #
370    #
371    ##############################################################################
372
```

gyield.s

```
 1
 2  /**************************************************************************
 3   *                                                                        *
 4   *               Copyright (C) 1994, Silicon Graphics, Inc.               *
 5   *                                                                        *
 6   *  These coded instructions, statements, and computer programs  contain  *
 7   *  unpublished  proprietary  information of Silicon Graphics, Inc., and  *
 8   *  are protected by Federal copyright  law.  They  may not be disclosed  *
 9   *  to  third  parties  or copied or duplicated in any form, in whole or  *
10   *  in part, without the prior written consent of Silicon Graphics, Inc.  *
11   *                                                                        *
12   **************************************************************************/
13
14  /*
15   * File:               gyield.s
16   * Creator:            hsa@sgi.com
17   * Create Date:        Fri Sep 30 10:59:23 PDT 1994
18   *
19   * This module implements some overhead routines for load balancing.
20   * The strategy is very simple: Every so often, the graphics task
21   * looks to see if it should stop (CPU will tell it). If so, we
22   * save out our state and quit.
23   *
24   * Likewise, upon start-up, if this is a continuation of a stopped
25   * task, we do a special initialization, then jump directly to
26   * the processing.
27   *
28   */
29
30   ############################################################################
31   #  In FASTLIGHT3D this code is in overlays.  The RSPYieldRestart
32   #  function is in the doInit code which is in the basic microcode,
33   #  but gets overlayed by clip and lighting.  The RSPYieldOver function
34   #  is in the Done overlay which is loaded over clip or lighting.
35   #
36   #  In cases other than FASTLIGHT3D this code is included in 2 different
37   #  spots in the main code.  The RSPYield function is included by
38   #  defining YIELD_STOP and the RSPYieldRestart function is included by
39   #  defining YIELD_RESTART.
40   #
41
42   #============================================================================
43   #===== This code (RSPYieldOver in FASTLIGHT3D, otherwise RSPYield) is ======
```

```
44   #===== used to halt the microcode when a yield is requested and DMA   ======
45   #===== the dmem (including saved registers) to dram                   ======
46   #=============================================================================
47  #ifdef YIELD_STOP
48  .name   yield,  $2
49  .name   dmem_addr,      $20
50  .name   dram_addr,      $19
51  .name   dma_len,        $18
52  .name   iswrite,        $17
53
54
55  #ifndef FASTLIGHT3D
56  RSPYield:
57  #else /* FASTLIGHT3D */
58  RSPYieldOver:
59  #endif /* FASTLIGHT3D */
60          # save important registers to DMEM
61          # only the "constant" registers are important.
62                  ori     yield, zero, SP_SET_YIELDED     # indicate yield
63                  sw      dlcount,    (RSP_SETUP_TMP_OFFSET +  4)(zero)
64                  sw      dinp,       (RSP_SETUP_TMP_OFFSET +  8)(zero)
65                  sw      inp,        (RSP_SETUP_TMP_OFFSET + 12)(zero)
66                  sw      outp,       (RSP_SETUP_TMP_OFFSET + 16)(zero)
67
68                  lw      dram_addr, RSP_STATEP_YIELD_STORE(zero)
69                  ori     dmem_addr, zero,0
70                  ori     dma_len, zero, (RSP_YIELD_SAVE_LEN - 1)
71                  jal     DMAproc
72                  ori     iswrite, zero, 1
73   ### CALL TO SUBROUTINE DMAproc:
74                  jal     DMAwait
75                  nop
76   ### CALL TO SUBROUTINE DMAwait:
77                  j       TaskDone
78                  mtc0    yield, SP_STATUS                # indicate yield
79   ### JUMP OCCURS to TaskDone:
80
81  .unname dmem_addr
82  .unname dram_addr
83  .unname dma_len
84  .unname iswrite
85  .unname yield
86
87  #endif /* YIELD_STOP */
88
```

```
 89   #=============================================================================
 90   #===== This code (RSPYieldRestart) is used to restart from a previous ======
 91   #===== yield.  At this point the dmem should have been restored from  ======
 92   #===== dram (this is done by the 4300 CPU).                           ======
 93   #=============================================================================
 94   #ifdef YIELD_RESTART
 95
 96           #
 97           # $1 holds the task header
 98           #
 99   RSPYieldRestart:
100           # all 4K of DMEM is already initialized.
101
102   #ifndef TURBO3D
103           # load Newton overlay
104                   jal     loadOverlaySR
105                   addi    $30, zero, OVERLAY_NEWTON
106   #endif
107
108           # load some registers.
109                   lw      outp,       (RSP_SETUP_TMP_OFFSET + 16)(zero)
110                   lw      dlcount,    (RSP_SETUP_TMP_OFFSET +  4)(zero)
111                   lw      dinp,       (RSP_SETUP_TMP_OFFSET +  8)(zero)
112                   j       GfxDone
113                   lw      inp,        (RSP_SETUP_TMP_OFFSET + 12)(zero)
114
115   #endif /* YIELD_RESTART */
```

newt.s

```
 1
 2  /*
 3   * Copyright 1995, Silicon Graphics, Inc.
 4   * ALL RIGHTS RESERVED
 5   *
 6   * UNPUBLISHED -- Rights reserved under the copyright laws of the United
 7   * States.   Use of a copyright notice is precautionary only and does not
 8   * imply publication or disclosure.
 9   *
10   * U.S. GOVERNMENT RESTRICTED RIGHTS LEGEND:
11   * Use, duplication or disclosure by the Government is subject to restrictions
12   * as set forth in FAR 52.227.19(c)(2) or subparagraph (c)(1)(ii) of the Rights
13   * in Technical Data and Computer Software clause at DFARS 252.227-7013 and/or
14   * in similar or successor clauses in the FAR, or the DOD or NASA FAR
15   * Supplement.  Contractor/manufacturer is Silicon Graphics, Inc.,
16   * 2011 N. Shoreline Blvd. Mountain View, CA 94039-7311.
17   *
18   * THE CONTENT OF THIS WORK CONTAINS CONFIDENTIAL AND PROPRIETARY
19   * INFORMATION OF SILICON GRAPHICS, INC. ANY DUPLICATION, MODIFICATION,
20   * DISTRIBUTION, OR DISCLOSURE IN ANY FORM, IN WHOLE, OR IN PART, IS STRICTLY
21   * PROHIBITED WITHOUT THE PRIOR EXPRESS WRITTEN PERMISSION OF SILICON
22   * GRAPHICS, INC.
23   *
24   */
25
26  #include <rsp.h>
27  #include "mbi.h"
28
29                  .text   0x04001000
30
31  #ifndef FASTLIGHT3D
32
33  #include "gdmem.h"
34  #include "gfx_regs.h"
35
36                  .text
37  #endif /* FASTLIGHT3D */
38
39          ###########################################################################
40          #
41          # Newton-Raphson routine for the reciprocal approximation:
42          #
43          #       r2 = r * (2 - r * x)
```

```
44          #
45          # Does one pass of Newton's iteration.
46          #
47          # Does the *2 to S15.16 first.
48          #
49          # Uses registers:
50          #
51  .name   xi,     $v29    # original x
52  .name   xf,     $v28
53  .name   ri,     $v27    # approximation of 1/x  (also holds iterated answer)
54  .name   rf,     $v26
55  .name   r2i,    $v25    # intermediate value
56  .name   r2f,    $v24
57  .name   vtmpi,  $v23    # constant 2.0
58  .name   vtmpf,  $v22
59
60                  .ent    NewtonDiv
61
62                  # NewtonDiv does the divide too, then the Newtons.
63                  # use of element 3h favors the transform code.
64                  #
65          NewtonDiv:
66                          vrcph   ri[3], xi[3]
67                          vrcpl   rf[3], xf[3]
68                          vrcph   ri[3], xi[7]
69                          vrcpl   rf[7], xf[7]
70                          vrcph   ri[7], vconst[0]
71                          vmudn   rf, rf, vconst[2]
72                          vmadh   ri, ri, vconst[2]
73                          vmadn   rf, vconst, vconst[0]
74          Newtons:
75                          lqv     vtmpi[0], VNEWT_OFFSET(zero)
76                          vxor    vtmpf, vconst, vconst
77                          vmudl   r2f, rf, xf             #  R*X
78                          vmadm   r2f, ri, xf
79                          vmadn   r2f, rf, xi
80                          vmadh   r2i, ri, xi
81                          vsubc   r2f, vtmpf, r2f         #  2 - (R*X)
82                          vsub    r2i, vtmpi, r2i
83                          vmudl   vtmpf, rf, r2f          #  R * (2-R*X)
84                          vmadm   vtmpi, ri, r2f
85                          vmadn   rf, rf, r2i
86                          vmadh   ri, ri, r2i
87                          jr      return
88
```

```
 89                         nop
 90  #ifndef FASTLIGHT3D
 91                         nop
 92                         nop
 93                         nop
 94                         nop
 95                         nop
 96                         nop
 97  #endif /* FASTLIGHT3D */
 98
 99                 .end    NewtonDiv
100
101  .unname xi
102  .unname xf
103  .unname ri
104  .unname rf
105  .unname r2i
106  .unname r2f
107  .unname vtmpi
108  .unname vtmpf
109         #
110         ########################################################################
111
```