3D Rendering Code on CPUs & Possible GPU Designs

Goal: Find something better than CPU for 3D rendering pipeline code.

Plan:

By hand analysis find potential performance of rp (rendering pipeline) code.

Determine how closely CPU achieves potential.

Consider minor CPU modifications.

Consider new GPU design.

Rendering Pipeline Software CPU Execution Analysis

Analysis Performed Using

CPU-only "Demo" routines from course.

Compiled for SPARCV9 with visual and multiply/add insn.

Use Sun Studio 12 compiler with aggressive optimizations.

Simulated execution on Fujitsu SPARC64 VI.

Use RSIML for simulation and PSE for visualization.

SPARCV9

RISC ISA, developed by Sun Microsystems.

Has 32 64-bit floating point registers.

SPARC64 VI

Four-way superscalar. (Can execute up to 4 insn / cycle.)

Dynamically scheduled. (Can execute insn out of prog. order.)

Can start two FP operations per cycle.

Can keep 64 insn in flight (32 int, 32 fp).

RSIML and PSE

RSIML used to simulate approximate model of SPARC64 VI.

PSE shows detailed results of simulations.

PSE will be used to find execution performance limiters.

RP Software Characteristics

Important RP software characteristics to look for:

Substantial independence.

High floating-point density.

Relaxed FP precision requirements.

Vertex Transformation Code Analysis

Start with something simple, vertex transformation.

```
for ( pVertex_Iterator ci = vtx_list.begin(); ci < vtx_list.end(); ci++ )
  {
    pVertex& v = **ci;
    v *= transform;     // Multiply matrix by coordinate (vector).
    v.homogenize();     // Some divisions.
  }
```

Performance Measures

More descriptive: Vertices / Second

Easier for this analysis: CPU Instructions / Vertex

Vertex Transformation Code Analysis

Computation requirements per vertex.

Last vertex test: 3 insn (load, comparison, conditional branch).

Vertex load: 4 loads.

Matrix mult: 4 mult, 12 multiply/add insn.

Homogenize: 1 divide, 3 multiplies.

Vertex store: 4 stores.

Total: 31 instructions.

Peak rate: $\frac{31}{4} = 7.75$ cycles per vertex.

Will CPU realize peak rate? Could something else be faster?

Vertex Transformation Code Analysis

Determine if 31 insn / vtx and 7.75 cyc / vtx achieved.

Compiled Code

First try: over 150 instructions per vertex.

With minor code changes compiler generated $\approx$ 31 insn / vtx.

Execution on Simulated Processor

Achieves 1.8 IPC (insn / cycle), less than half of possible 4 IPC.

Lost performance because ROB (window) fills.

With 256-entry ROB: 3.3 IPC, but that's impractically expensive.

Software pipelining may improve execution without larger ROB.

Vertex Transformation Code: Minor CPU Modifications

Minor CPU Modifications

Since operating at close to full efficiency, not really needed.

A lower-precision, faster divide might ease coding (sw pipelining).

Vertex Transformation Code: Alternative Design

Plan

Examine parts of CPU implementation, looking at chip floorplan.

Eliminate parts that are not really needed.

Use free space for more of parts that are needed.

SPARC64V floorplan:

FP unit is less than 3.2% of chip area.

Integer (FX) unit: 5.2% of chip area.

L1 data cache: 5.5%

L2 cache: 46%.

Dynamic scheduling: 15% (roughly).

Remainder: Buffering, memory management.

Ideas for Chip just for VTX Transformation

Plan A: Just FP Unit

Since 3.2% of area, can have over 31 FP units: 31× faster!

Could this work?

Plan B: Eliminate dynamic scheduling and L2 Cache

Reduction in area: L2 -46%, DS -15%: have 61% area to fill.

Could fit second core: 2× faster.