Architecture Studies 1
an architecturally mediocre presentation engine

Silicon Graphics
Akeley and Jermoluk
DN10000VS
Kirk and Voorhies
1990 (Paleozoic)
Leo
Deering and Nelson
1993 (Brenden finishes 6th grade)

When you think of hardware, think of rectangles.
© 2002 Brenden Schubert

High-Performance Polygon Rendering
every presentation should tell a story:

• Once upon a time, there were two guys named Kurt Akeley and Tom Jermoluk.
• Kurt was a genius. He went to quality schools such as the Stanford engine.
• Tom was not so bright – he simply had an propensity for corporate elevation. He went to a second-tier school in Blacksburg, Va.

(These slides aren’t going on the web, are they?)

Anyway the System Overview

• Transfer data to graphics subsystem
• Transform vertex coordinates
• Transform vertex normals
• Light each vertex
• Clip each vertex
• Project each vertex
• Map projected coordinates to screen
• Fill screen-space polygon in framebuffer

• Scan framebuffer engine at screen refresh rates
• Be prepared to switch contexts
• Do all this, like, really fast

Data Transfer to Graphics Hardware

• Data specified by function calls
  – i.e. glVertex3f
• 400,000 vertexes per second = 10 MB/s
• System bus engine capable of 64 MB/s
• Use glVertex3fv to send larger blocks of data, improve bus efficiency
Geometry Subsystem

- Comprised of 5 Geometry Engines
  - Specialized 20 Mflops floating-point processors
- The Geometry Engine
  - FIFO input buffer (4 entries)
  - Full pipelined concurrent microcode operation execution
- Geometry Engine pipeline operates at 40-50 Mflops

Geometry Subsystem: Geometry Engine Pipeline

1. Vertex and Normal transformation
2. Lighting calculations
3. Clip testing
4. Perspective division
5. Viewport transformation, color clamping, depthcue calculations

Scan Conversion Engine

- Polygon Decomposer splits polygon into triangles and trapezoids with vertical edges
- Edge Slope Calculators calculate slopes
- Edge Interpolators find endpoints of each span to 1/8 pixel accuracy
- Span Slope Calculators calculate color and depth slopes at span endpoints
- Span Interpolators generate 40 Mpixels/s

Raster Subsystem

- 20 Image Engines TM
- Receive pixels from Scan Conversion Engine
- Perform Z-buffer operations, replace, high-speed clear
- Write to the framebuffer in 5x4 tiles

<table>
<thead>
<tr>
<th>Operation</th>
<th>Fill Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replace</td>
<td>80 Mpix/s</td>
</tr>
<tr>
<td>Z-buffer</td>
<td>40 Mpix/s</td>
</tr>
<tr>
<td>Z-buffer/blend</td>
<td>10 Mpix/s</td>
</tr>
<tr>
<td>High-speed clear</td>
<td>160 Mpix/s</td>
</tr>
</tbody>
</table>

Display Subsystem

- Reads pixel data from framebuffer
- Interprets and routes color/alpha values to DA converters
- Drives the display engine
Special Features

• Pan and Zoom
• Window ID masking
• Realtime Video
• Alpha Blending
• Antialiased Lines

Nolan used to have a desktop like this:

Performance

• 400,000 vertexes/s
• Geometry limited around 140,000 tris/s
• Fill limited around 40 Mpixels/s
• 210,000 antialiased lines/s
• Screen clear in 8.3 ms
  – @30hz 25% of frame time is clearing buffer engine

The End.

The DN10000VS

• "I'd like to use 'profligate' in a SIGGRAPH paper."
• "Let's make one of the Key Words 'quadratic'."
• "Maybe we can get away with software rendering if our cpu and memory are really fast"
• "Maybe if for some reason this looks like it might not work, we should go found a specialized graphics-hardware company instead"

System Overview

• Multiple CPUs used for transform and lighting calculations
  – CPUs optimized at isa level for vertex transformation calculations
  – Spin-locks used to control access to rendering data and rasterizing engine
• Rasterizing Engine driven via main system bus
  – Walks trapezoid scan lines, quadratically or linearly interpolating color, depth, and alpha
“Fast Phong” Shading

- Gouraud Shading produces Mach bands (retina doesn’t like discontinuities in first derivative of intensity)
- Quadratic interpolation + exponentiation texture map = ‘fast phong’ shading
  - Fast Phong shading is Gary Bishop’s approximation of the phong shading model
  - Requires lighting calculations at vertices and edge midpoints
- No Mach bands, curved highlights

“Fast Phong” Shading

No Mach bands, curved highlights:
[I’d planned to put the pictures here but even at hi-res the color versions suck. Seriously. So much that it wasn’t even worth putting them here anyway. Sure, I could have gone and found some other pictures of Mach bands. But this took less effort.]

Texture Mapping

- Texture must reside in framebuffer
- Re-use color interpolators to interpolate texture coordinates
- MIP-mapping used to prefilter textures
- Quadratic interpolation used for perspective-correct texture interpolation

Quadratic interpolation used for perspective-correct texture interpolation:

Conclusions

- Quadratic Interpolation is probably the biggest contribution
  - Not used by Akeley and Jermoluk
  - Used for both shading and perspective-correct texturing

Conclusions

- 1990 must just have been the wrong time to be designing graphics hardware, because David Kirk is a smart guy
- Let us count the follies:
  - Do graphics calculation on the CPU to ride the CPU performance curve and get better application-graphics load-balancing
  - Multiple processors process geometry at the primitive level, using spin-locks to interface to critical resources
  - Simplify hardware to the point of no programmability (not even microcode)
  - Memory is expensive – if possible, use main memory for the framebuffer
Conclusions

• Graphics Hardware is not supposed to be ‘holistic’. It is meant to be fast. Yoga is meant to be ‘holistic’.

The End.

Leo

• To digress for a moment: Do active vs. passive verbs in an abstract affect how exciting the paper sounds?

<table>
<thead>
<tr>
<th>Paper</th>
<th>Passive Verbs in Abstract</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kirk &amp; Voorhies</td>
<td>2</td>
</tr>
<tr>
<td>Deering &amp; Nelson</td>
<td>3</td>
</tr>
<tr>
<td>Akeley &amp; Jermoluk</td>
<td>3.5</td>
</tr>
</tbody>
</table>

Leo System Overview

• LeoCommand
  – FIFO buffer for graphics commands
• LeoFloat
  – Floating-point intensive geometry calculations
• LeoDraw
  – Screen space rendering to VRAM
• LeoCross
  – Reads framebuffer, color lookups, double-buffering, video timing

LeoCommand

• Decomposes triangle strips into individual triangles
  – D’oh!
• Compresses color components, normals, and texture coordinates into 16 bits while shoveling them around the chip
  – Sweet!

LeoFloat

• Custom microcodable ASIC specialized for 32-bit floating-point graphics calculations
• Double buffered asynchronous input and output packet hardware
• 288 internal registers (exactly the number needed to transform/clip/light a triangle)
• Horizontally microprogrammable (avoids complex out-of-order execution tables)
• “Kitchen sink” instruction
LeoDraw

- Figure 5: somehow justifies decision to use 5-way interleaved memory (and thus 5 LeoDraw chips)
- Broadcast triangle to all 5 rasterizers
  - Each thinks it’s rasterizing the same triangle, but onto memory 1/5 as dense

LeoDraw

- CAD features:
  - Antialiased lines and dots
  - Per-pixel depth cue
  - ‘Picking’ support
- Parallel 3D/2D contexts – multithreaded application can make simultaneous 2D and 3D graphics calls

LeoCross

- Video output
- Supports virtual reality

Results

- 210,000 100-pixel lit, Gouraud-shaded, z-buffered, depth-cued triangles/s
- 422,000 12 pixel long antialiased lines/s
- 730,000 antialiased points/s
- 10 Mpix/s drawpixels

The End.