Lecture 20: Thursday, 18 November 1999

Next few weeks:
- Today, 18 Nov: Graphics Hardware
- T 23 Nov: Visible Surface Determination (VSD) I
- R 25 Nov: Thanksgiving Break, No class
- T 30 Nov: VSD II; HKN evaluations
- R 2 Dec: Summary; graphics and careers
  - This will be the final 6.837 lecture !
- F, 3 Dec – Thursday, 9 December
  - Final project presentations in 3-133

Specialized Graphics Hardware

Overview
- Bandwidth estimates
- UNC Pixel Planes 5
  - Fuchs et al, Siggraph '89
- UNC PixelFlow
  - Molnar et al, Siggraph '92
  - Commercialized by Division
- SGI Reality Engine
  - Akeley, Siggraph '93

Bottlenecks

Classical pipeline:
- Traverse database
- Transform, shade, clip
- Rasterize
- Write to frame buffer

Bottlenecks:
- Memory, I/O bandwidth
- Floating point speed
- Integer (fixed point) speed
- Frame buffer memory bandwidth
- Context switching (real-world)

Example

Sample scene with 100,000 10×10 triangles
- 25 visible pixels, on average (how?)

Ambient and local diffuse illumination

Gouraud interpolation

$1280 \times 1024 \times 24$ bit (rgb), 32 bit (depth)

30 Hz update rate

Per-Frame Geometry Calculations

Traversal
- 300,000 vertices, normals (6 load/store)
- 100,000 start/end tokens, (1 load/store)

Modeling transformation
- 300,000 vertices, normals (25 mults + 18 adds)
- 7.5M mults, 5.4M adds

Trivial accept/reject (gross clipping)
- Dot each vertex against six planes
- 24 mults + 18 adds per vtx: 7.4M mults, 5.4M adds

Lighting
- 12 mults + 5 adds per vtx: 3.6M mults, 1.5M adds

Per-frame Geometry Calculations (cont.)

Viewing transformation
- 8 mults + 6 adds per vtx: 2.4M mults and 1.8M adds

Clipping
- highly variable
- (assume all primitives within view frustum)

Division by w
- 3 divs per vertex, 0.9M divides

Map to NDC
- 2 mults + 2 adds per vertex: 0.6M mults, 0.6M adds

Totals:
- 22M mults/divides + 15M adds per frame
- At 30Hz, more than a GigaFlop!
  - (Not including application, system o/head, &c.)

Per-frame Rasterization Calculations

Visible pixels:
- Compute z (1 add)
- Read & compare depth (1 load, 1 compare)
- Compute r, g, b (3 adds)
- Write z, r, g, b to framebuffer (4 stores)

Invisible pixels:
- Compute z (1 add)
- Read & compare depth (1 load, 1 compare)

Each triangle covers 50 pixels, on average
- $100{,}000 \times 50 \times 5 = 15$M buffer accesses

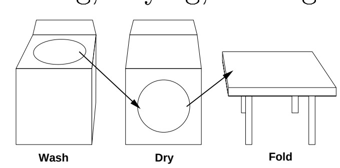
Double-buffer clear:
- 1.25Mpix r, g, b, z buffer accesses = 5Mword

600 Million frame buffer accesses per second!
- (didn't mention Phong lighting, texture mapping!)

How do we do 1200 MFlops, 600 M fb ops per second?

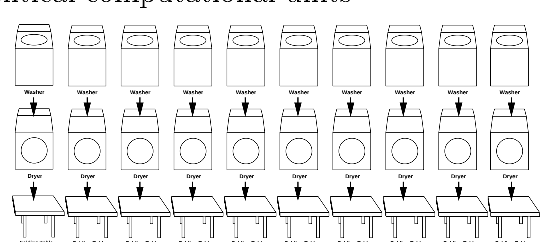
Pipelining and Parallelism

Pipelining
- Simultaneous, cascaded use of (typically)
  - distinct computational units
- Example: Washing, Drying, Folding laundry



- Issues: startup, stalling, context switch

Parallelism
- simultaneous use of (typically)
  - identical computational units



- Issues: variance, inherent data parallelism

SGI Reality Engine

Pipelined MIMD architecture
- 8 MIMD geometry engines
- 320 MIMD rasterizers

Triangle:
- Vertices (OS), colors, normal (OS), texture
  - queued to input FIFO
- Packet passed to geometry engine
  - → ES, lighted
  - → CS, clipped
  - → NDC
  - Texture coordinates → NDC
  - r, g, b, α, z, & texture slope info computed

Reality Engine

Triangle now ready for rasterization
- Broadcast on "triangle bus" to
  - (5, 10 or 20) "fragment generators"
  - Finely (5-, 10-, or 20-way) interleaved
  - Fragment: pixel with color, depth, texture info
- Communicated to "image engines" (16 per FG)
  - Local DRAM, subset of framebuffer
  - Does z compare, alpha combination, FB write

Observations
- Data stream merges only at triangle bus
- Large triangles processed by nearly every CPU!
- FIFO queues everywhere, to prevent stalls

Reality Engine

Points to note:
- Antialiasing (how?)
- Round-robin geometry scheduling
  - Implications?
- Pineda-style rasterization
- Scissoring
- Deep FIFO queues
  - Implications?
- Pixel interleaving
- Deferred shading, texturing (later)

Cost?

How scaleable?

RE Performance (1993)

Host: 1-8 MIPS CPUs

6-12 GEs, each a 50MHz i860XP (100 MFlops)
- with 2Mb of local memory

1-4 RM boards, each with:
- 5 FGs (each with texture memory!)
- 80 IEs (deliver video at 500 Mb/second), and
- $1280 \times 1024 \times 256$ bits/pixel framebuffer

ASIC for input, output FIFO, registers

One million triangles per second (transform)

140 million fragments per second (fill)

UNC Pixel Planes 5 (1989)

Tile screen space with rasterizers
  Assemble sub-images
Quadratic expression evaluator per pixel
Renders 1M polygons/spheres per second
Scaling problems:
  Network bandwidth
  Sorting
  Load imbalance

UNC PixelFlow

Compositing, "deferred shading" architecture
  1-128 SIMD Geometry/Rasterizer pairs
  Each computes a full-screen "image"
    of a fraction of primitives!
  Deferred shading after visibility
  Simple programming model
  SIMD linear expression evaluators
    evaluate $Ax + By + C$ at each pixel
  Deferred shading
    Each pixel shaded exactly once
    Implication?

UNC PixelFlow

Discussion
  Every pixel must be transmitted every frame
  Restrictive visibility/transparency algorithms
  Frame of storage required per renderer
  Composition network causes latency
How scaleable?

Pixel Flow Performance (1993, 1995)

Composition network bandwidth
  30 GigaBits / second
Geometry processor (i860XP)
  150,000 triangles / second
Rasterizer
  1.4 million Gouraud triangles / second
  0.8 million Phong, texture-mapped
  Divide by supersampling rate (here, 5)
Deferred Shader
  10,000 128×128 regions per second
  Mip-map textures: 3,700 regions per second
  Remember: one primitive per region!
Bottlenecks
  Linear scaling until composition time dominates
  Need **large** datasets to scale linearly
    up to 128 renderers: 400,000 triangles/sec!