Graphics Hardware

Mike Gleicher
12/06/2006
Lecture Notes – not projected!

Graphics Hardware

• Why?
  – Need lots of computation to do graphics
  – Lots of pixels, lots of polygons, lots of texels, …
• A few standard things done very often
  – Pipeline provides a standard set of abstractions
  – Break everything into triangles
• Regular computations + pipelineable
• Moving target – changing faster than processors!

History

• 1980s – first workstation 3D hardware (SGI)
• 1990s – extension of abstraction set
  – Texture mapping, compositing, multi-buffering
• 1990s – first PC graphics hardware
  – Low end (Apple’s white magic project)
  – High end (3D solutions – expensive)
• 2000s – consumer graphics hardware
  – Driven by gaming market
  – Extensive use of the abstractions
• 2002++ - programmable graphics hardware
  – Better abstractions, generality, use as GP processor

Graphics Pipeline

• Fixed set of abstractions
  – Doesn’t really change
  – Can optimize
  – Fits a programming model
• Early Graphics Hardware
  – 4x4 transform engines
  – Fill Engines
  – Scanline hardware (Apple)

Working with the Pipeline

• Where is your bottleneck?
• Get your triangles fast
  – Vertex sharing schemes
  – Display lists / v-buffers
• Filling pixels
  – Lots of z tests (read/write)
  – Texture accesses per pixel
• Limitations
  – Set operations for each phase

Early Extensions to Pipeline

• Texture Mapping
• Accumulation Buffer
  – More light sources
  – Compositing
  – Anti-Aliasing / Motion Blur
• Stencil Buffer
Pipelining in conventional processors

• Start step 2 before step 1 completes
• Unless step 2 depends on step 1
• Pipe Stall

C = A * B
F = D * E
J = G * H

Pipelines in graphics processors

• Conventional processors – stalls are bad
  – Need shorter pipelines
• Pixels and vertices are independent
• Pipes can be long
• Parallelism is easy
  – Start as many at a time as you want

Programming the Pipeline

• Vertex programs
  • Given the info about a vertex
    – Local coords, transform matrices, colors
    – Lights
  • Figure out the color and position
    – Typical: standard lighting model
    – Give a little program
• Parallel – all vertices happen at once
• Deeply pipelined (not intervention till end)

Fragment Shaders

• Fragment = Pixel (?)
  – Multiple fragments = 1 pixel if anti-aliasing
• Given “context” figure out color to write
  – Pixel (fragment) position already known
  – Gets control over z-test, …
• Highly parallel
• All pixels run the same program
  – SIMD – single instruction multiple data

Why is graphics hardware fast?

• Highly parallel
  – Simple parallel model
  – Lots of little processors
• Deeply pipelined
  – Results are independent
• Multiple processors on a chip is way of future
  – Speeds can’t get faster
  – Chips can’t get bigger (cross chip latencies)