The Graphics Pipeline Revisited

(thanks to David Luebke, University of Virginia)

Johns Hopkins Department of Computer Science
Course 600.460: Interactive Graphics and Games, Spring 2005, Professor: Jonathan Cohen

---

Classic Rendering Pipeline
(as taught in Intro Graphics courses)

3D Geometric Primitives

- Modeling Transformation — Transform into 3D world coordinate system
- Lighting — Illuminate according to lighting and reflectance
- Viewing Transformation — Transform into 3D camera coordinate system
- Projection Transformation — Transform into 2D screen coordinate system
- Clipping — Clip primitives outside camera's view
- Scan Conversion — Draw pixels (includes texturing, hidden surface, ...)

Image

Wrong

Classic Rendering Pipeline

3D Geometric Primitives

- Modeling Transformation
- Lighting
- Viewing Transformation
- Projection Transformation
- Clipping
- Scan Conversion

Image

What's wrong with this model (for an OpenGL system)?
- Model/view transforms combined
- Really "vertices" not "primitives"
  —Making this the *vertex pipeline*
- There's a lot going on in the "scan conversion" stage!
  —Primitive assembly
  —Rasterization
  —Texture mapping
  —Per-pixel lighting
  —Visibility (Z-buffer)
- We refer to these collectively as the *pixel* or *fragment pipeline*

High-Level Pipeline

Back up & think about the larger picture:

Application Processing → Geometry Processing → Rasterization

CPU | GPU

• What sort of tasks does each stage perform?

High-Level Pipeline

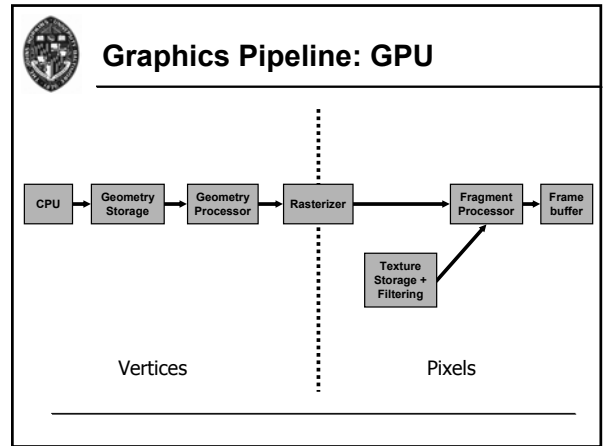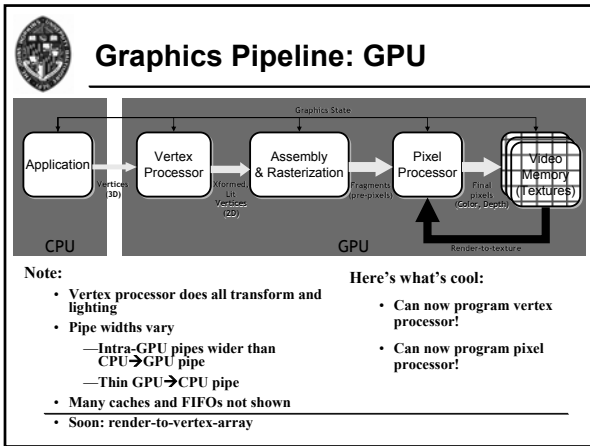| Application	Geometry (a.k.a. "vertex pipeline")	Rasterization (a.k.a. "pixel pipeline" or "fragment pipeline")
Handle input	Transform	Rasterize (fill pixels)
Simulation & AI	Lighting	Interpolate vertex parameters
		Look up/filter textures
Culling	Skinning	Z- and stencil tests
LOD selection	Calculate texture coords	Blending
Prefetching		

Graphics Pipeline: CPU

SIM

CULL

DRAW

DISPLAY

Graphics Pipeline: GPU

Graphics State

Application → Vertex Processor → Assembly & Rasterization → Pixel Processor → Video Memory (Textures)

Vertices (3D) → Xformed Lit Vertices (2D) → Fragment (pre-pixels) → Final pixels (Color, Depth)

Render-to-texture

CPU | GPU

Note:
- Vertex processor does all transform and lighting
- Pipe widths vary
  - Intra-GPU pipes wider than CPU→GPU pipe
  - Thin GPU→CPU pipe
- Many caches and FIFOs not shown
- Soon: render-to-vertex-array

Here's what's cool:
- Can now program vertex processor!
- Can now program pixel processor!

Graphics Pipeline: GPU

CPU → Geometry Storage → Geometry Processor → Rasterizer → Fragment Processor → Frame buffer

Texture Storage + Filtering

Vertices | Pixels

Graphics Pipeline: GPU

Don't forget! Not your father's (or professor's) GPU

Possible Bottlenecks

CPU transfer transform raster texture fragment frame buffer

CPU → Geometry Storage → Geometry Processor → Rasterizer → Fragment Processor → Frame buffer

Texture Storage + Filtering

CPU/Bus Bound | Vertex Bound | Pixel Bound