Johns Hopkins
Engineering for Professionals

605.767 Applied Computer Graphics

Brian Russin

Module 11G
Pipelining

Pipelining

- Pipeline consists of several stages in succession
  - Example: oil pipeline
    - Oil cannot move from 1st stage into 2nd until oil in 2nd stage has moved to the third (and so on)
    - Speed of pipeline is determined by its slowest stage
- Ideally dividing a process into n pipeline stages would produce a speed-up factor=n
  - Pipeline stages execute in parallel
  - Are stalled until slowest stage finishes its task
- Geometry stage is a good example of a pipeline

Pipelining

- Graphics hardware is easier to pipeline than a CPU
  - Pixels are most often independent of each other
  - Few branches, much fixed functionality
  - Don’t need high clock frequency
    - Bandwidth to memory is bottleneck
  - Majority of memory accesses are reads
    - Easier to predict memory access pattern and do prefetching
- Examples
  - Pentium IV has 20 pipeline stages, 42 million transistors
  - NVIDIA GeForce3 GPU has 600-800 pipeline stages, 57 million transistors
  - Newer GPUs
    - ATI Radeon 9700: 110M transistors
    - NVidia GeForce FX 5800: 125 M transistors, 500 MHz
  - Early 2000s number of transistors in GPUs began exceeding CPUs

Parallelism

- Can achieve faster graphics performance by parallelizing operations
  - Can be done in both geometry and rasterizer stages
  - Compute n results in parallel then merge results
- Example
  - Application sends data to a set of geometry units that work in parallel
  - Forward their results to a set of rasterizer units
- Multiple results computed in parallel
  - Some method of sorting must be done to get intended image

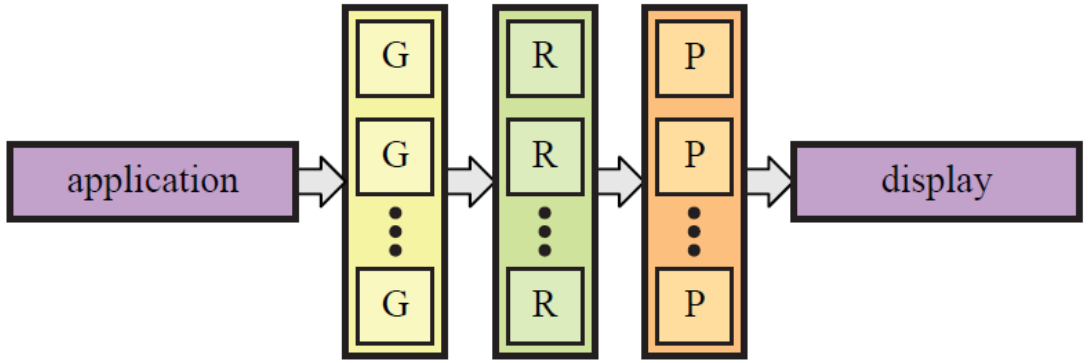


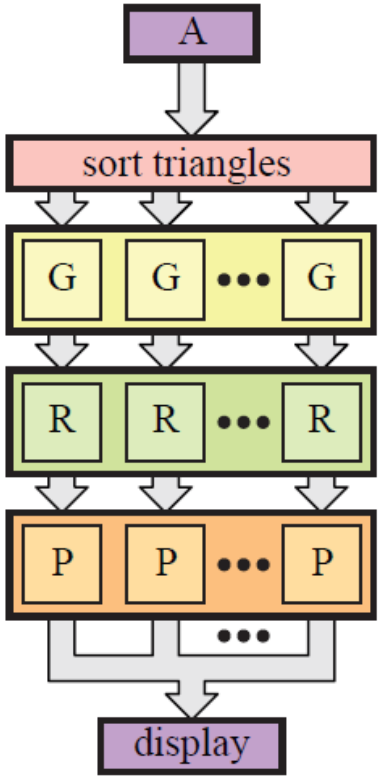
Figure 23.16

Taxonomy of Parallel Graphics Architectures

- Need to sort from model space to screen space
- Four major architectures (presented by Eldridge, et al.)
  - Sort-First
  - Sort-Middle
  - Sort-Last Fragment
  - Sort-Last Image
- Geometry stage is broken into geometry units (G)
- Rasterization stage broken into two units based on **fragments**
  - A fragment is all the generated information for a pixel on a triangle
  - R is a Rasterization units
    - Finds which pixels are inside triangle – i.e. generates spans
  - P is a Pixel Processing unit
    - Merges the fragments with various buffers: depth, color
  - Texturing is done during P
- Text focuses mostly on Sort-Middle and Sort-Last Fragment
  - Used in commercial hardware

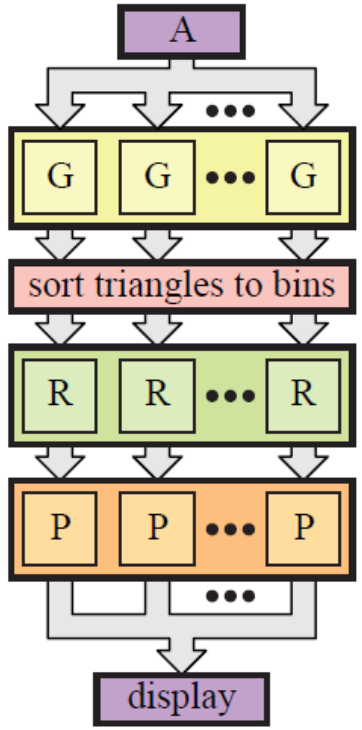
Sort-First

- Sorts primitives before geometry stage
- Screen is divided into large regions
  - A separate pipeline is responsible for each region
- Triangles may overlap regions
  - Triangle sent to all intersected regions
  - Clipping per region can be performed
- Most practical in a system with multiple displays
  - Chromium is a cluster-based rendering system which could use this architecture
- Not explored much at all



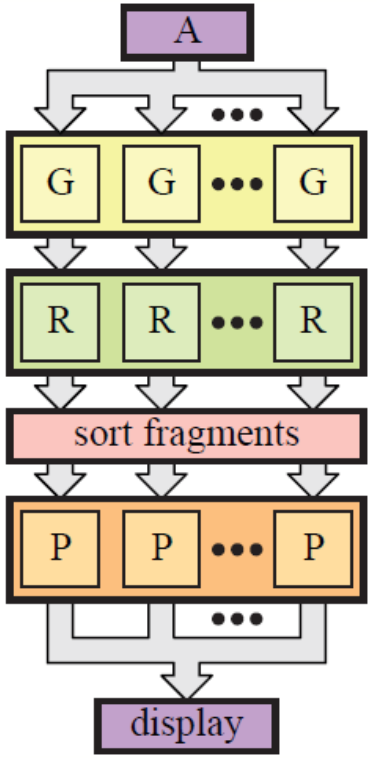
Sort-Middle

- Sorts between geometry and rasterizer stages
  - Know the screen-space positions of the triangles after geometry stage
- Spread work arbitrarily among geometry stages
- Sort results to different rasterizers depending on screen-space position
  - Screen can be split into ”tiles”
    - Rectangular blocks (8x8 pixels)
    - Every n scanlines
- Each rasterizer is responsible for rendering inside a tile
  - A triangle can be sent to many R’s depending on overlap (over tiles)
- Most commercial hardware uses this architecture
  - Text has case studies of a sort-middle architecture
    - ARM Mali G71 Bifrost



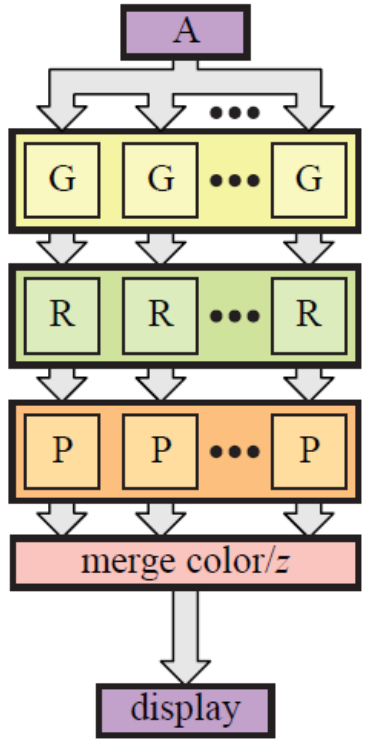
Sort Last Fragment

- Sorts between R and P
  - XBox architecture
- Spread work among geometry stages
- Generated work is sent to Rs
- Sort fragments to Ps
  - Each P is responsible for a tile of pixels
- A triangle is only sent to one R
  - Avoids doing the same work twice
    - In Sort-Middle if a triangle overlaps several tiles then the triangle is sent to all Rs responsible for these tiles
    - Results in extra work



Sort-Last Image

- Sorts after entire pipeline
  - Can be seen as a set of independent pipelines
- Each R and P has a separate frame buffer for entire screen (Z and color)
- After all primitives have been sent to pipeline, the Z buffers and color buffers are merged into one color buffer
- Can be seen as a set of independent pipelines
- Huge memory requirements
- Used in research
  - PixelFlow
- This architecture cannot fully support OpenGL
    - OpenGL requires primitives be rendered in order they are sent



Geometry Unit

- Geometry stage can be located on host (CPU), on special geometry acceleration hardware, or both
- Commonly implemented in dedicated hardware today
  - Special hardware can be implemented on custom chip
    - Pipelined and parallelized
  - Called fixed-function implementations
    - Not much flexibility
- Programmability has been introduced into geometry stage
  - Vertex shaders – small programs that can be executed in geometry stage
    - Executed for each vertex sent
  - Vertex shader units can be implemented in custom hardware
    - Limited form of a CPU

Rasterizer Unit

- Rasterizer implemented on custom chips for speed
- Triangle setup
  - Computes deltas and slopes based on triangle vertex information
  - Interpolate color, depth, texture coordinates
    - Interpolation is not always linear (e.g. perspective correct textures)
- Setup and interpolation produces a fragment
  - Data for a pixel: color, depth, alpha, texture coordinate
  - Rasterizer is also programmable in modern hardware: pixel shaders
- Some rasterizer chips only support triangles
  - Do not support points and lines directly
    - Draw rectangles to represent them

Pixel Processing Unit

- Programmable in modern hardware: fragment shaders
  - Lighting
  - Fog
  - Composite colors
  - Texture access
- Fixed-function operations
  - Depth and Stencil tests
  - Blending
  - Texture filtering
    - Up to 40 times faster to use dedicated hardware
      - Fetching
      - Filtering
      - Decompression