Johns Hopkins
Engineering for Professionals

605.767 Applied Computer Graphics

Brian Russin

Module 11B
Locating Bottlenecks

Pipeline Optimization

- Primarily from From Haines and Moller
  - Chapter 18 (15 in 3rd Edition)
  - Highly recommended!
- Nvidia presentations
  - https://docs.nvidia.com
- In a pipeline architecture good performance depends on finding and eliminating **bottlenecks**
  - Bottleneck varies over the course of a frame
  - Find where the bottleneck resides over most of the frame

Performance Bottlenecks

- CPU / bus limited
  - Vertex and data transfer from CPU to graphics board is the limiting factor
  - Often related to how data is stored for use
    - Inefficient access
  - Application cannot feed data to the geometry stage fast enough
- Geometry limited
  - Performance is limited by large amount of geometry transforms being performed
    - Primitives too high resolution
    - Complex lighting
- Fill / raster limited
  - Large polygons with numerous pixel level operations
    - e.g. Blending can reduce performance

Rendering Speed

- Slowest pipeline stage determines the **rendering speed**
  - Expressed in frames per second
    - Number of rendered images per second
  - Slowest stage is called the **bottleneck**
- Framebuffer rates with double buffering
  - Rendering is synchronized with the update rate of the monitor
    - E.g. monitor with 72 Hz refresh will take 1/72 sec = 0.01389 sec to scan and display the framebuffer
    - Framebuffer can only be swapped during vertical retrace
      - Time between scan of bottom line of framebuffer and rewinding to top
  - If drawing takes longer than scan converter update rate it has to wait until next vertical retrace
    - Limits frame rates to scan rate / n
    - Example: 72Hz scan rate
      - Frame rates can be 72Hz, 72/2=36Hz, 72/3=24Hz, 72/4=1Hz etc.

Rendering Speed (cont.)

- Use single buffering to measure total rendering time
- Average frame rate is unreliable measurement
  - e.g. Three seconds are analyzed with 50fps, 50fps, 20fps.
    - Average is **40fps**
  - **Incorrect** when measuring render time
    - 50fps = 20 ms / frame
    - 20fps = 50 ms / frame
    - Average: (20 + 20 + 50)/3 = 30 ms / frame = **33.3 fps!**

Application Stage

- Developer controls the application stage
  - Executes in software
- Two main phases
  - Main process and subsystems
    - Update scene graph elements
    - Culling: view frustum, occlusion, contribution
    - Collision detection
    - Animation: moving objects, texture animation, morphing
  - Graphics API calls
    - Feeds geometry to the geometry stage
    - Points, lines, triangles
    - Data access, processing/culling, etc.
    - Updates to uniform buffers, textures, etc.

Pipeline Optimization

- 3 stage process:
  - 1) Locate the bottleneck
  - 2) Perform some optimization on the bottleneck stage
    - Decreases the workload in the bottleneck stage
  - 3) Repeat step 1 if performance goal not achieved
    - Note: location of bottleneck may have changed!
- Do not spend too much effort on 2 (over-optimization)
  - If bottleneck moves you may waste effort
- When bottleneck stage cannot be further optimized you can increase work in other stages
  - Will not decrease frame rate since bottleneck stage is not altered
  - Examples
    - Use more complex lighting if geometry stage is not bottleneck
    - Add textures or blending if rasterization stage is not bottleneck

Locating the Bottleneck

- Cannot find bottleneck by simply timing the process
  - Tells total time for bottleneck stage, not which stage
- Must set up specific tests that affect one stage at a time
  - When total rendering time is lowered we have found the bottleneck
- Related method: reduce workload in other stages
  - Do not reduce workload on the stage being tested
  - If performance does not change then the bottleneck is in the stage not altered

Testing the Application Stage

- Use a utility to measure CPU work
  - UNIX: **top** or **osview**
  - Windows Task Manager
  - If near 100% CPU usage the application is likely the bottleneck
  - Caution: could be a **busy wait** condition
- Use a **null driver**
  - Accepts API calls but does nothing
    - Geometry and rasterization stages will do little or no work
  - Provides an upper limit on how fast the application can run
  - Can hide bottleneck that is due to communication between application and geometry stage

Testing the Geometry Stage

- Most difficult stage to test
  - Change of workload here can change load in other stages as well
- Lighting operations only affect the geometry stage
  - Disable or add light sources
    - Test is performance is impacted
- Increase/Decrease the vertex buffer sizes
- Add logic to vertex shader
  - Although changes that do nothing may be optimized away
- Eliminating the other stages
  - Test application and rasterization stages
    - If neither is the bottleneck then the geometry stage must be!

Testing the Rasterization Stage

- Easy test
- Decrease the resolution of the window/ viewport
  - Fewer pixels are touched – less texture / depth buffer tests
  - If rendering performance increases then the pipeline is **fill-limited**
    - Rasterizer stage is the bottleneck
- Other tests:
  - Turn off depth testing and blending
    - If performance increases then rasterization stage is the bottleneck
  - Vary texture sizes or filtering options
    - May be “fill-limited” due to texture access/filtering