ATI IMAGEON

Media Processors for Mobile Phones

Optimizing Games for IMAGEON™ 3D – Enabled Handsets

Claude Benoit
Technical Evangelist

August 31, 2006

CEDEC 2006
CESA DEVELOPERS CONFERENCE 2006

OVERVIEW

- IMAGEON™ 3D Feature Set
- Optimizing 3D Games for Performance

IMAGEON™ 3D Feature Set

IMAGEON™ 3D Feature Set

	W230x	MSM75xx	W238x
	OpenGL ES 1.0	OpenGL ES 1.0 + Extensions	OpenGL ES 1.1 + Extension Pack + Extensions
Performance			
Triangle Rate (specs)	1 M triangles/sec	4 M triangles / sec	4 M triangles/sec
3D Pixel Fill Rate (specs)	100 M pixels/sec	125 M pixels/sec	125 M pixels/sec
Frame Buffer			
Supported Resolutions	QVGA Double-Buffered	QVGA / VGA Double-Buffered	VGA Double-Buffered
LCD Orientation	Portrait & Landscape	Portrait & Landscape	Portrait & Landscape
Color	16 bits/pixel	16 bits/pixel	16 bits/pixel
Z buffer	16 bits/pixel	16 bits/pixel	16 bits/pixel
Stencil	-	4 bits/pixel	8 bits/pixel
FBO: Frame Buffer Objects	-	-	Yes (1)

(1) Support for color, Z, stencil; all sub-formats according to OES_framebuffer_object

IMAGEON™ 3D Feature List

	W230x	MSM75xx	W238x
Geometry Engine			
Hardware Transforms	Yes	Yes	Yes
Vertex Buffer Objects	Yes	Yes	Yes
DX8-style Vertex Shaders	-	-	Yes
Hardware Point Sprites	-	Yes	Yes
Distance-based size factor	-	-	Yes
Vertex Skinning	-	Yes	Yes
Number of matrices/vertex	-	4	4
Total number of matrices	-	32	32
Hardware Lighting	-	-	Yes
Spotlight	-	-	Yes
Directional	-	-	Yes
Point	-	-	Yes

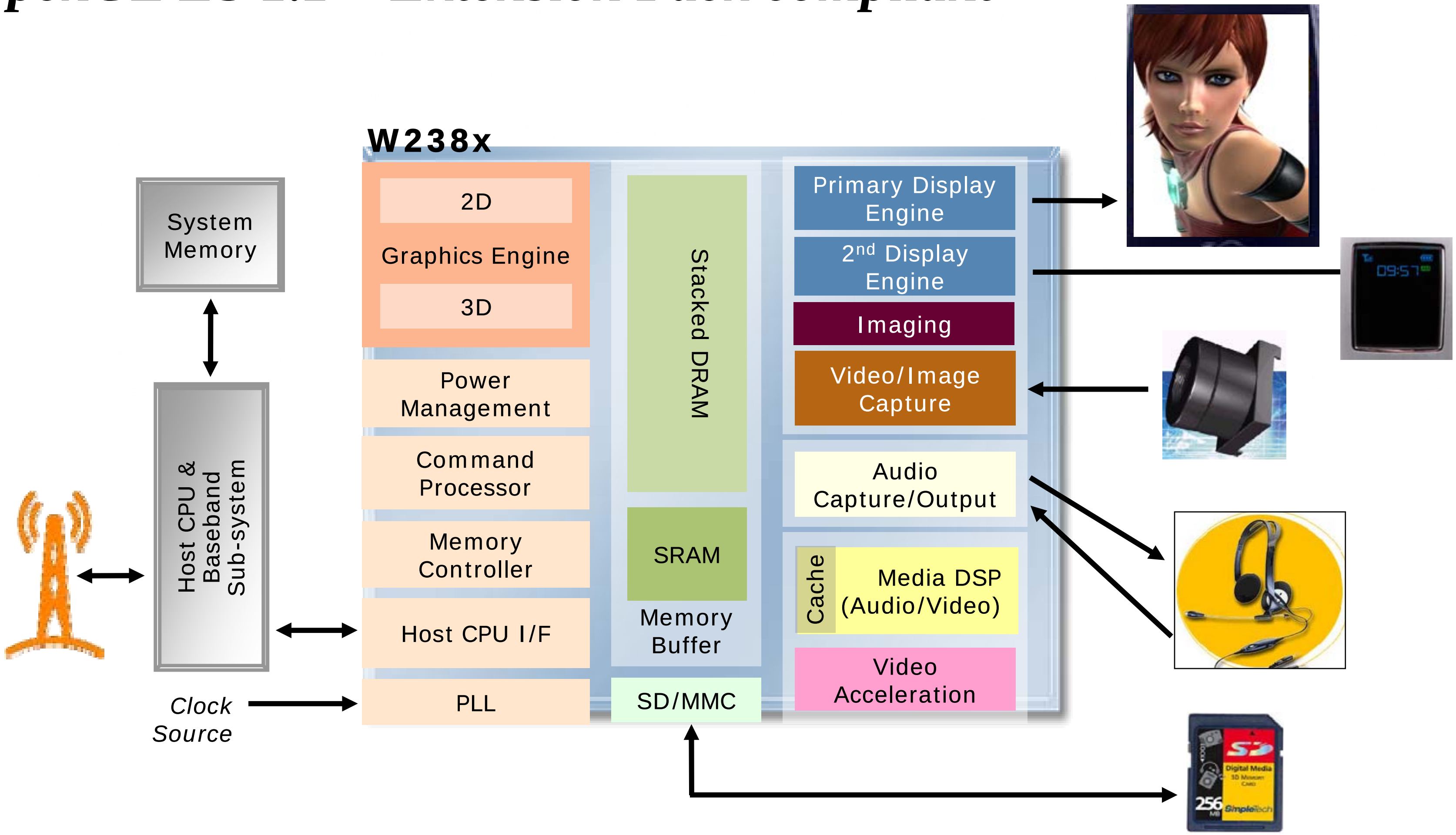
IMAGEON™ 3D Feature List

	W230x	MSM75xx	W238x
Texturing			
Texture pipelines	1	2	2
Texture Crossbar	-	Yes	Yes
Texture Compression (1)	-	Yes	Yes
Extended Texture Data Formats (2)	-	Yes	Yes
DOT3 Bump Mapping	-	Yes	Yes
Projective Textures	-	-	Yes
Cubic Mapping	-	-	Yes
Early Z Culling	-	-	Yes

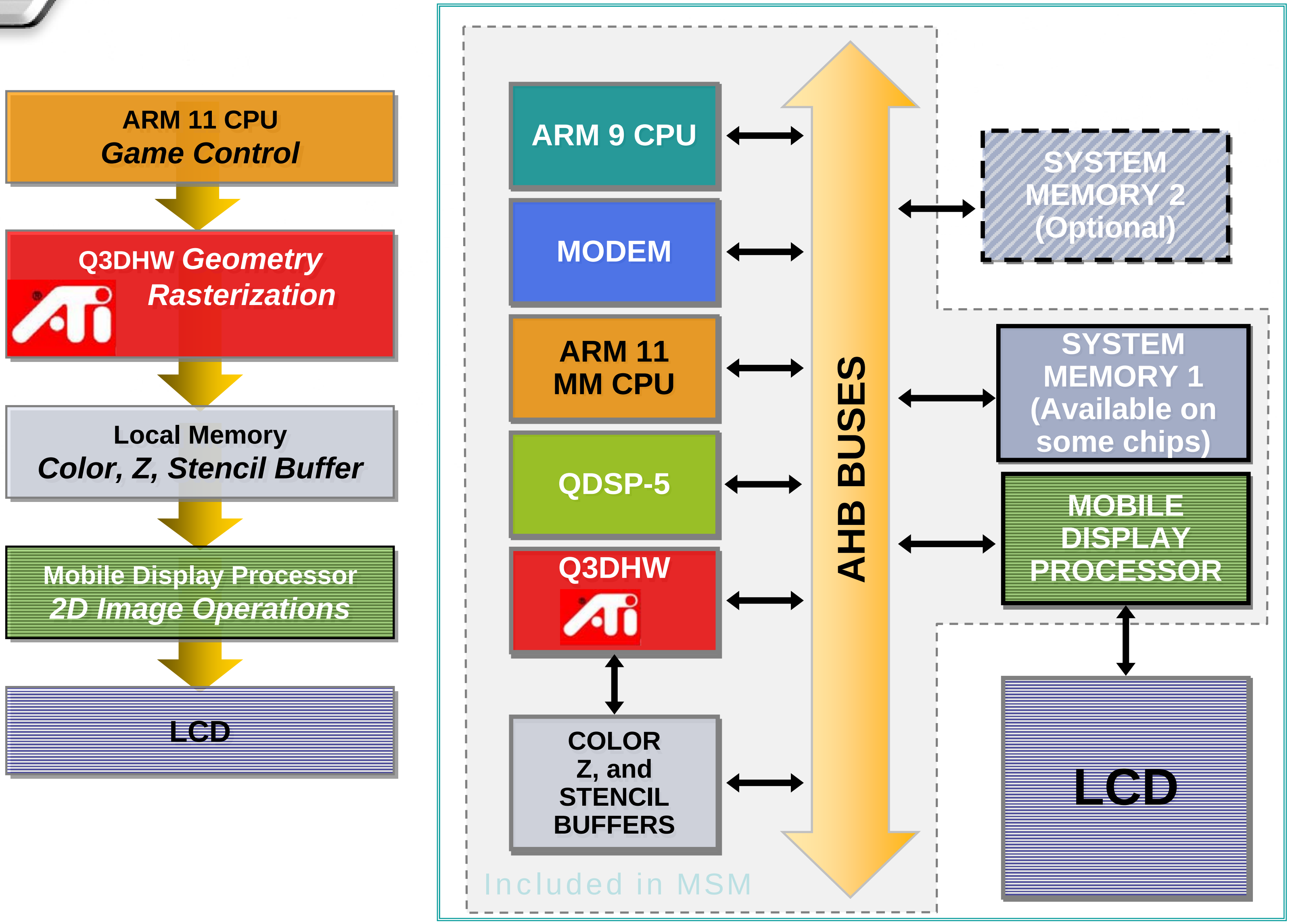
(1) ATI_TC for both RGB and RGBA
(2) Extended Texture Coordinate Data Formats: 4.4 / 8.8 / 4.12

W238x – Architecture Overview

OpenGL ES 1.1 + Extension Pack compliant



MSM7xxx -- Architecture Overview



Optimizing 3D Games for Performance

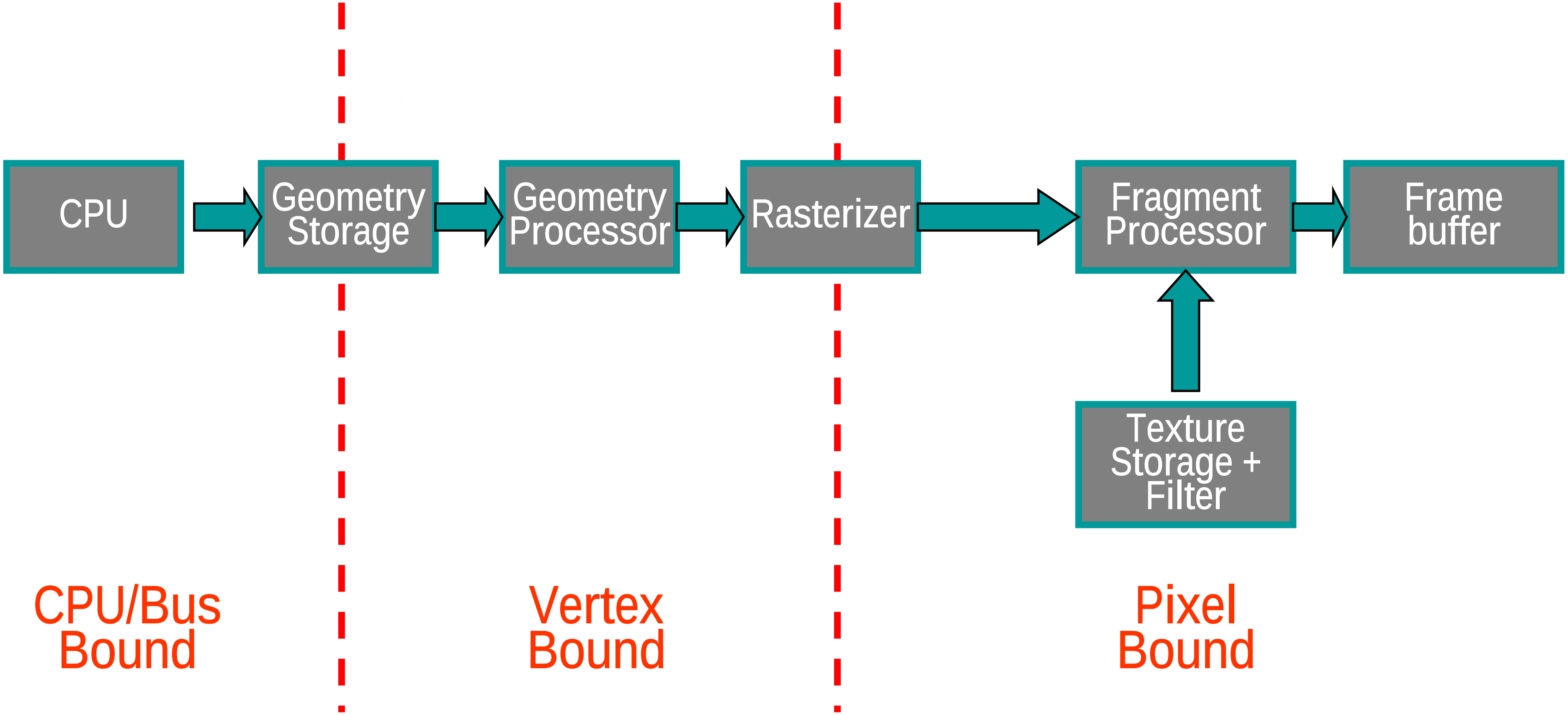
Battle Plan for Better Performance

- Locate the bottleneck(s)
- Eliminate the bottleneck (if possible)
    - Decrease workload of bottlenecked stage
- Otherwise, balance the pipeline
    - Increase workload of non-bottlenecked stage

Finding the Bottleneck

- Reduce the workload of different stages
- If performance does not change
  - Move on, this is not it
- If performance does change significantly
  - This is the bottleneck
  - Look to reduce workload
- Repeat
- TRICK: try finding the bottleneck that will give you the best ROI

Possible Pipeline Bottlenecks



Pixel Bottleneck

- Easiest to detect
  - Does performance scale with resolution?
- Multiple causes
  - Memory bandwidth
    - Disable blending
    - Reduce texture bit depth/size
  - Texture filtering
    - Turn off trilinear
  - Texture combine
    - Disable texture units

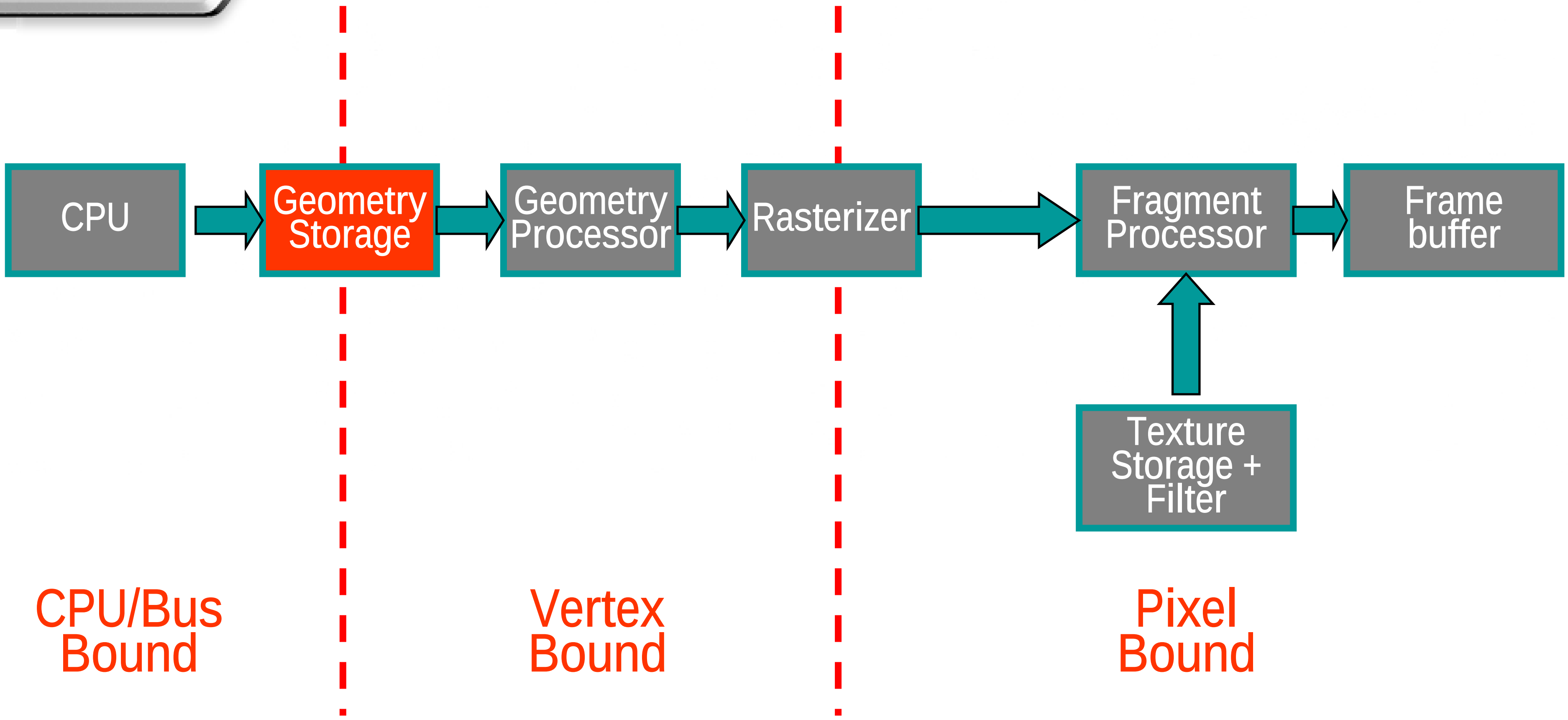
Vertex Bottleneck

- Harder to detect..
- Render ½ the triangles of each object
- Reduce the complexity of the vertex processing
  - Disable lights
  - Disable skinning
- If both scale performance
  - Vertex bottleneck
- If only reduced triangle count scales
  - Submission/fetch bottleneck

CPU Bottleneck

- Use profiler (if available)
- Find driver versus application time
- Turn off processing that does not affect rendering

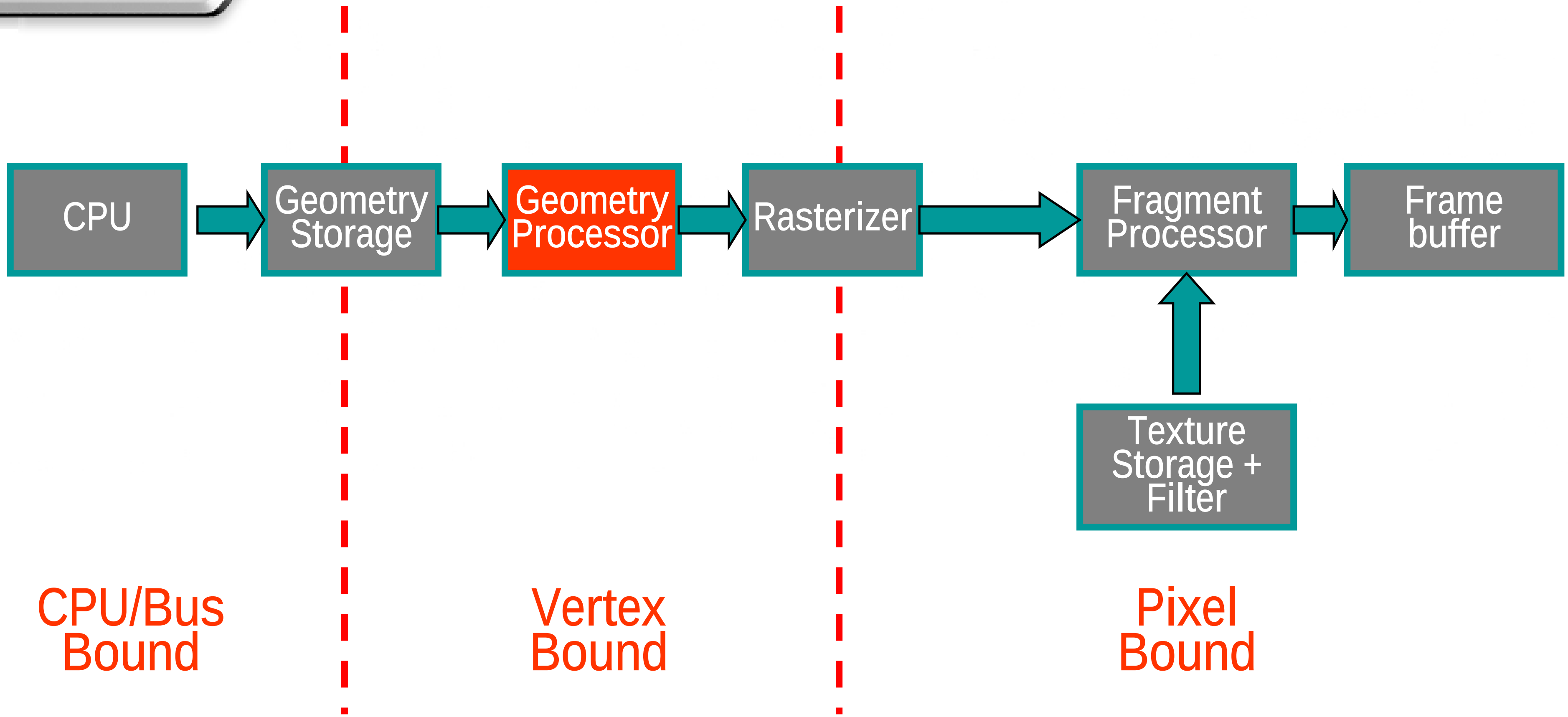
Geometry Transfer Bottlenecks



Geometry Transfer Bottlenecks

- Vertex data problems
  - Data size issues
  - Non-native types
- Using the wrong API calls
  - Not using vertex buffer objects or mesh lists
  - Non-indexed primitives
- Poor batching
  - Limit number of draw calls
  - Sort by material
- Too much dynamic vertex data

Geometry Transform Bottlenecks



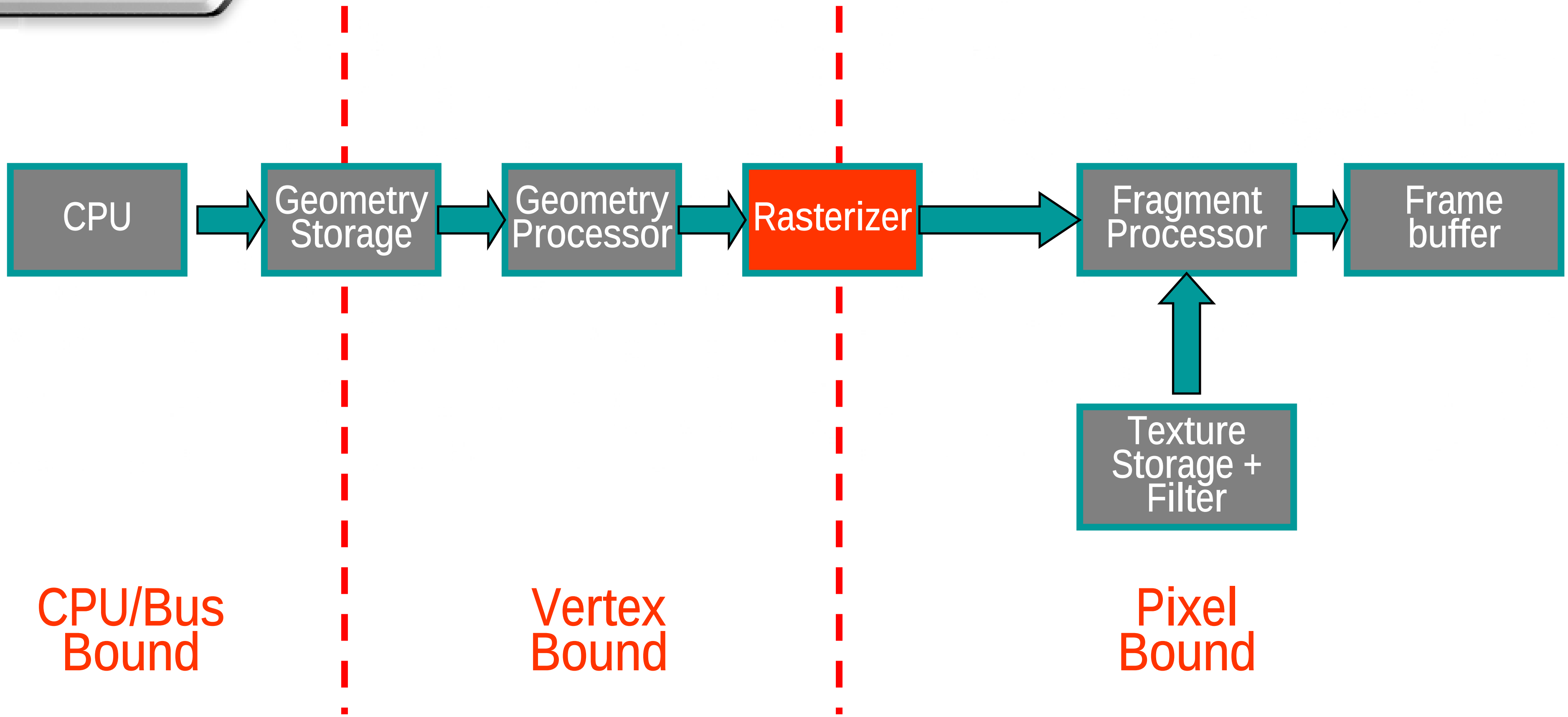
Geometry Transform Bottlenecks

- Too many vertices
  - Use LOD models
  - Use bump maps to fake geometric details
- Too much vertex computation
  - Pre-compute lighting or reduce number of lights
    - If you have to use lights, try to use directional diffuse
  - Avoid texture matrix
    - If using BYTE or SHORT texcoords, make sure texture matrix cost isn't offsetting the gain
  - Unnecessary use of GL normalize
    - Pre-compute normalized normals
  - Use fog sparingly

Geometry Transform Bottlenecks

- Calculations per-vertex that could be per-object
- Vertex cache efficiency (where exists)
  - Know your platform
    - Is there a vertex cache?
    - Are degenerate triangles preferred?
  - Re-order vertices to be sequential in use
  - Use degenerate triangles if appropriate

Rasterization Bottlenecks

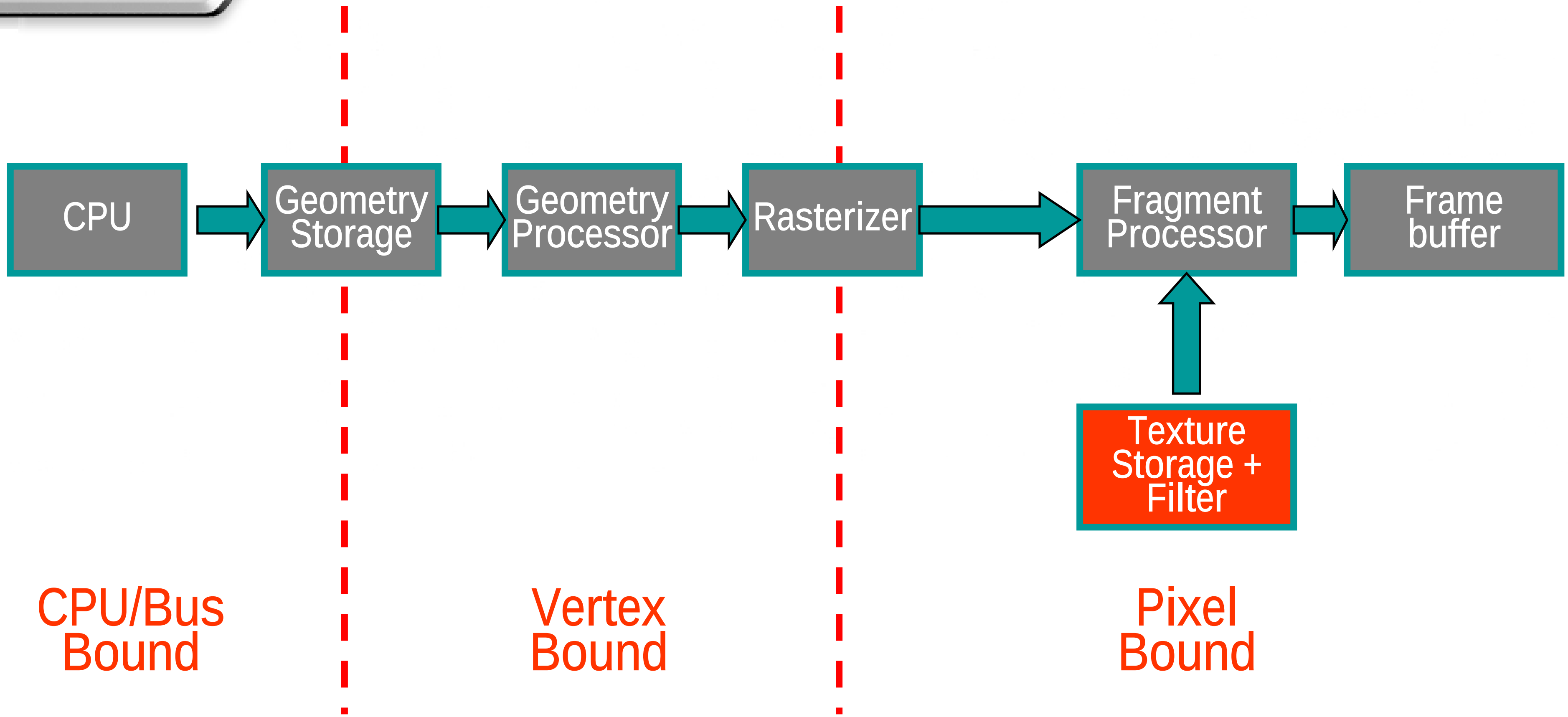


Rasterization Bottlenecks

- Interpolating unnecessary vertex attribute

- Poor depth culling efficiency
  - Always clear depth
  - Coarsely sort objects back to front
  - Constrain near and far planes to geometry visible
  - Avoid polygon offset
  - Again, know your platform!
    - Different advice for different GPUs...

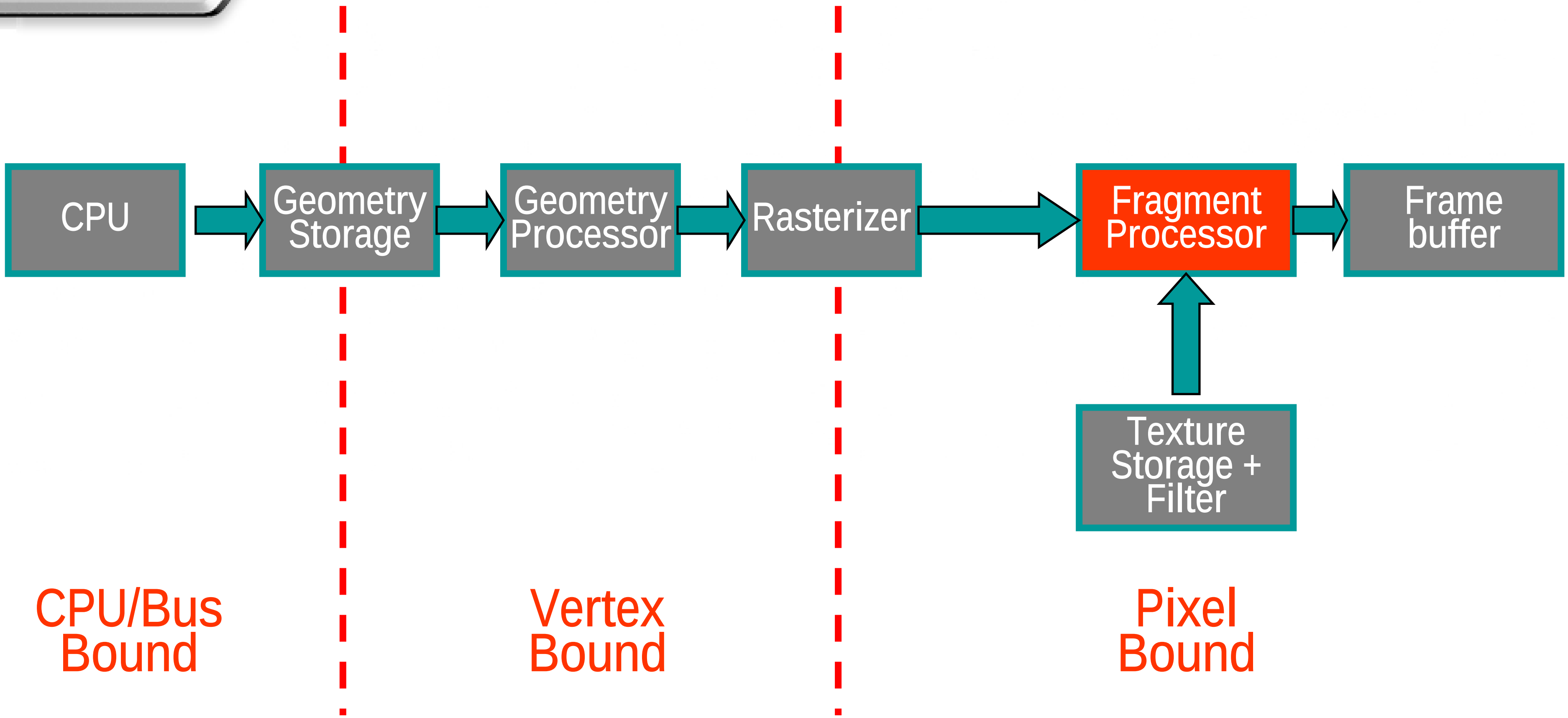
Texture Bottlenecks



Texture Bottlenecks

- Texture resolutions should only be as big as needed
- Compress textures:
  - Use ATI_TC texture compression
  - Palette textures are expanded in the driver.
- Poor texture cache utilization
  - Always use mip mapping
  - But not necessarily trilinear!
  - Best bang for the buck: LINEAR_MIPMAP_NEAREST

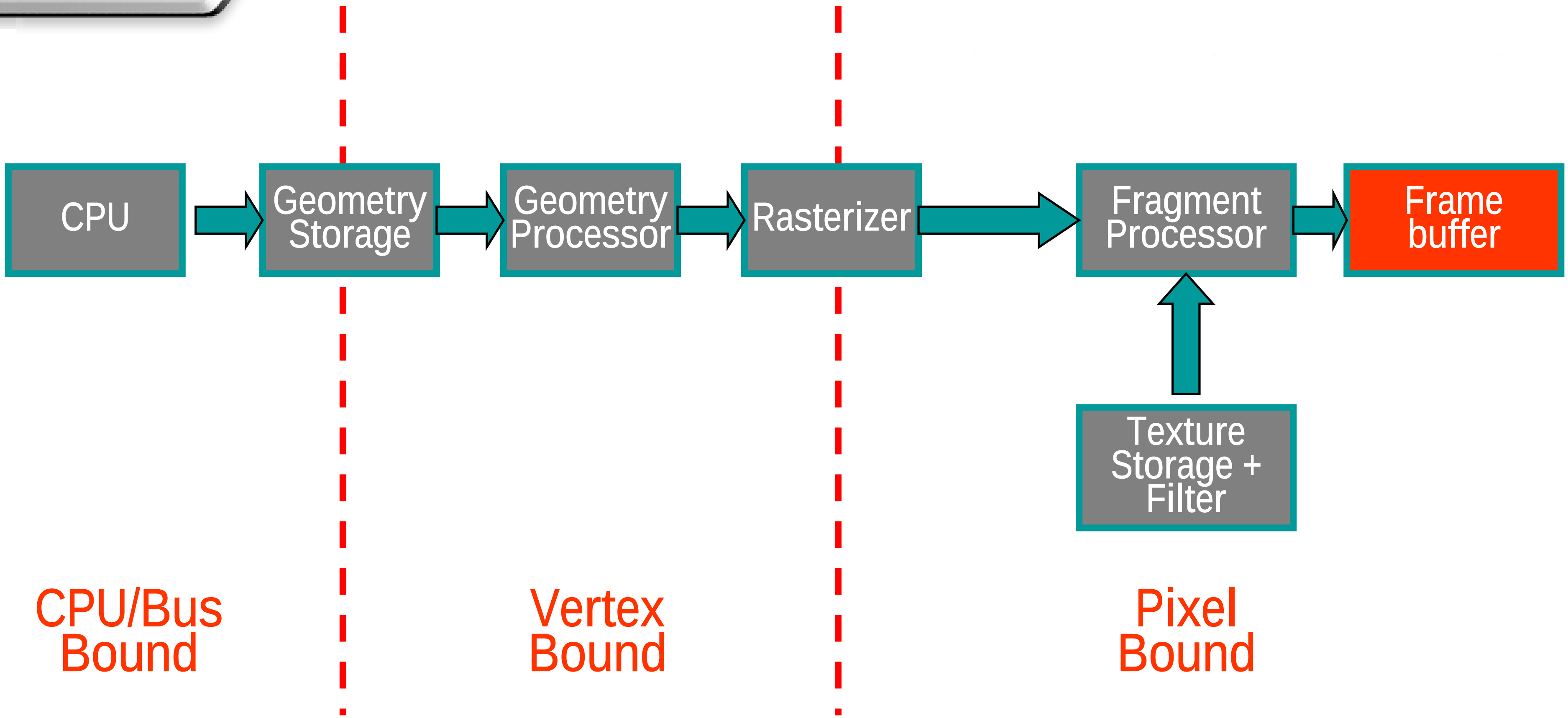
Fragment Bottlenecks



Fragment Bottlenecks

- Follow advice for maximizing depth culling efficiency
- Avoid unnecessary texture combine operations
    - Example: combine pass that does (Basemap * Constant)
    - Do this as a preprocess
- Unlikely bottleneck on fixed-function devices
    - But highly common with shaders...

Framebuffer Bottlenecks



Framebuffer Bottlenecks

- Turn off Z writes for transparent objects and multipass
  - glDepthMask
- Only do alpha blending if you have to
  - Use multi-texture to replace multi-passing if possible
  - Don’t assume glBlendFunc(GL_ONE, GL_ZERO) is free
- Use appropriate bit depths
  - For render-to-texture, use 16-bit color/z if possible

General Hardware Performance Tips

- Fixed-function lighting
  - Per-vertex
  - Per-pixel
- Render-to-texture
- Geometry storage

Fixed-Function Per-Vertex Lighting Tips

- Directional lights are faster than point lights
  - Determined by w-component of light position
- Specular computation adds additional overhead
  - Disable with specular material of (0, 0, 0, 0)
- Careful with lighting and skinning
  - Effectively reduces matrix palette size by half
  - Driver needs to transform normal using inverse modelview matrix
- Pre-compute lighting if not dynamic
- ....use per-pixel lighting instead!

Fixed-Function Per-Pixel Lighting Tips

- Lightmaps for static lighting
- DOT3 bump-mapping for dynamic per-pixel diffuse lighting
  - Use directional light & world-space normals for static geometry
  - Use cubemap normalizer for tangent-space surfaces
  - Reduces dynamic vertex data by allowing effective lighting with less geometry
  - Goal: avoid dynamic vertex data!
- Projective textures for spotlight effect
- Specular cube map to fake per-pixel specular
- Gloss map to attenuate per-pixel specular

Render-to-Texture Tips

- Don't use glCopyTexImage2D!
- Use OES_framebuffer_object for best performance
- Use glGenerateMipMapOES for explicit fast mipmap generation
- Share depth buffer among same-sized offscreen surfaces
- Use 2D offscreen texture for dynamic planar reflections
  - Texture matrix to compute projective texture coordinates
- Use cubemap for general dynamic reflections
  - Consider using proxy geometry for speed

Geometry Storage Tips

- Use GL_SHORT of GL_BYTE for position
    - Use modelview matrix to scale to desired range
- ATI tip: use ATI_extended_texture_coordinate_data_formats
    - Byte 4.4, Short 4.12, Short 8.8
    - Advantage: does not require texture matrix scaling!
    - Use GL_SHORT or GL_BYTE on other hardware
- Use GL_SHORT for normals
- For skinning: sort primitives by bone indices
- Use strips and degenerate triangles
- ATI tip: use ATI mesh lists for optimal performance!

Conclusions

- Locate performance bottlenecks
- Use best-practices to optimize each stage
- Use hardware features to offload from CPU -> GPU

QUESTION PERIOD