

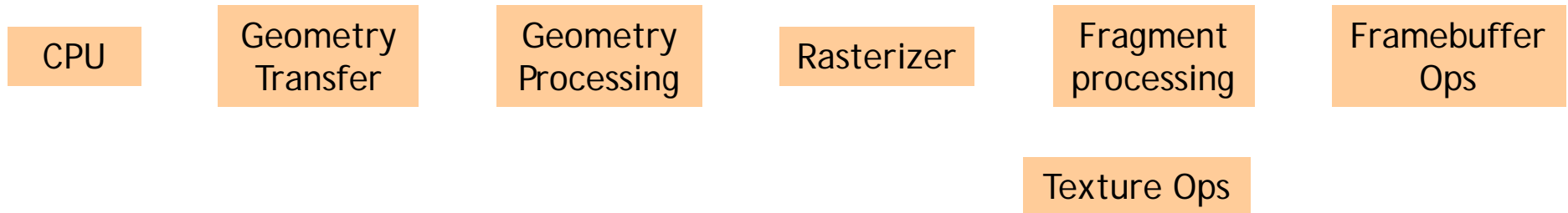
ME 290-R:
General Purpose Computation
(CAD/CAM/CAE) on the GPU
(a.k.a. Topics in Manufacturing)

Sara McMains

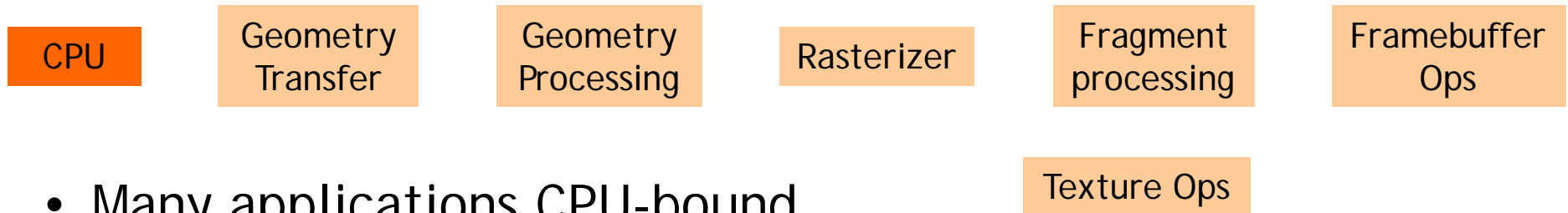
Spring 2009

Performance: Bottlenecks

Sources of bottlenecks

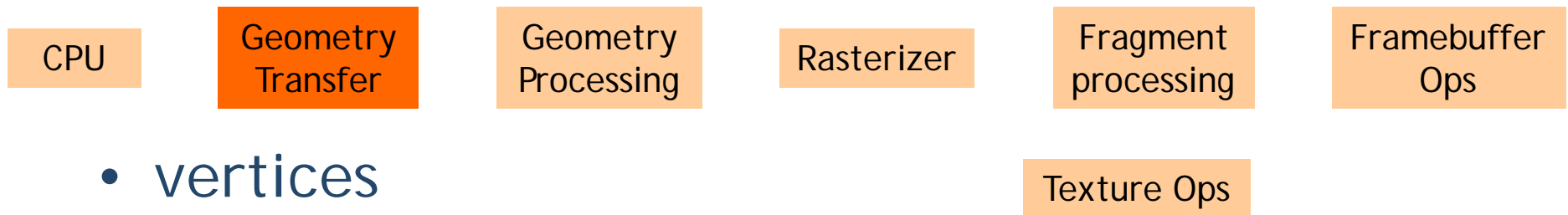


Sources of bottlenecks: CPU



- Many applications CPU-bound
 - preparing data
 - file I/O
- Inefficient rendering of geometry
 - triangle strips GOOD
 - many small batches BAD
- Too many state changes
 - cause pipeline to flush

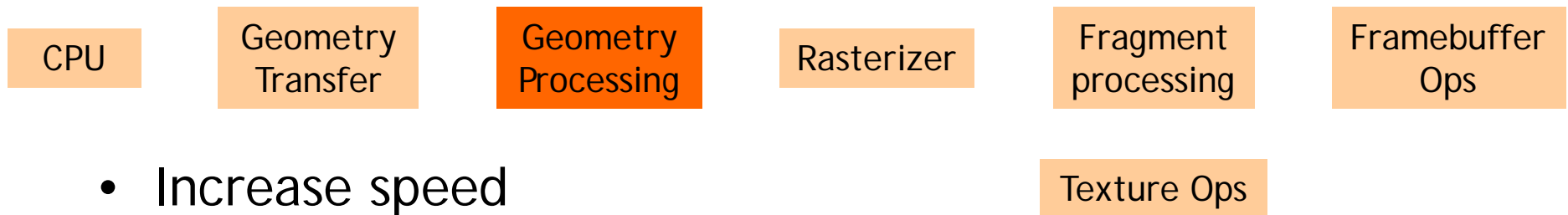
Sources of bottlenecks: Transfer



- vertices

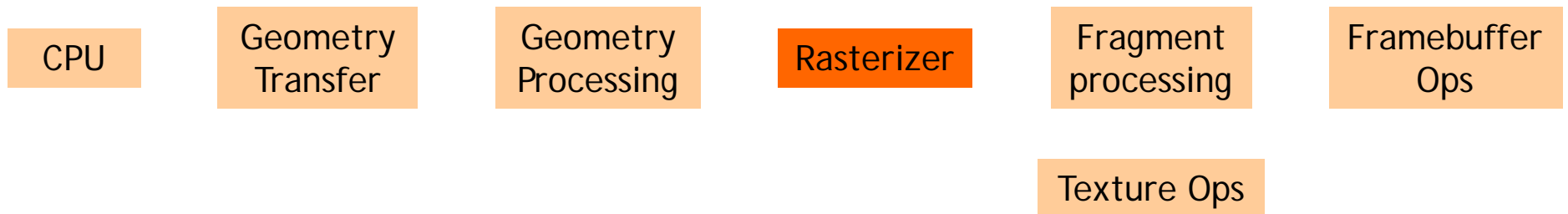
- multiple of 32 bytes best
- lower precision, compress, pad
- static geometry
 - display lists
 - write-only vertex buffer (D3D)
 - ARB_vertex_buffer_object (OGL)
- dynamic geometry
 - dynamic vertex buffer (D3D)
 - ARB_vertex_buffer_object (OGL)
 - access vertices sequentially (cache friendly)

Sources of bottlenecks: Transform



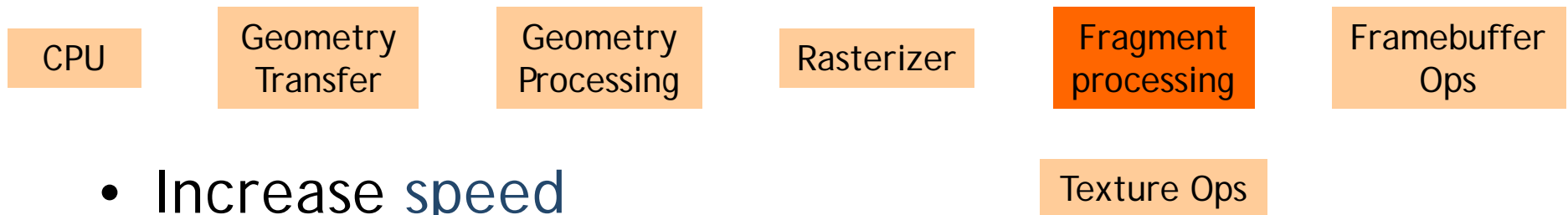
- Increase speed
 - Use triangle strips/fans
 - reduces vtx count
 - Use built-in operators rather than explicitly computing in VP
 - Use swizzles and masks
 - cheaper than MOVs
- Move bottleneck
 - shift computation from VP to the CPU
 - increase resolution for free!

Sources of bottlenecks: Rasterize



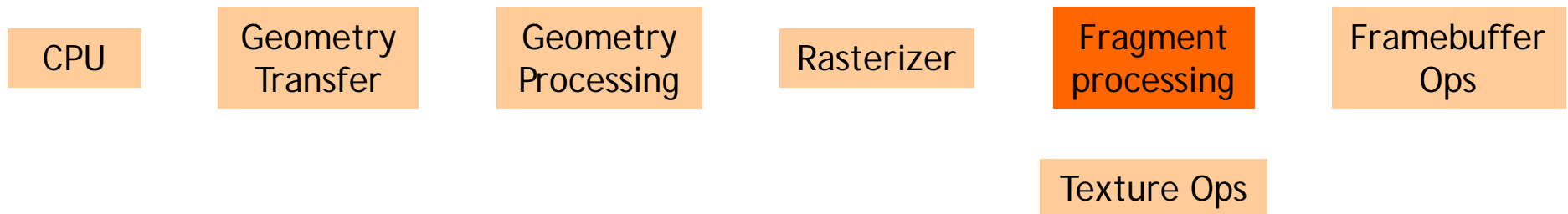
- rarely the bottleneck
- unless lots of z-culled triangles
 - no-ops downstream
- speed affected by
 - large triangles
 - lots of triangles

Sources of bottlenecks: Fragments



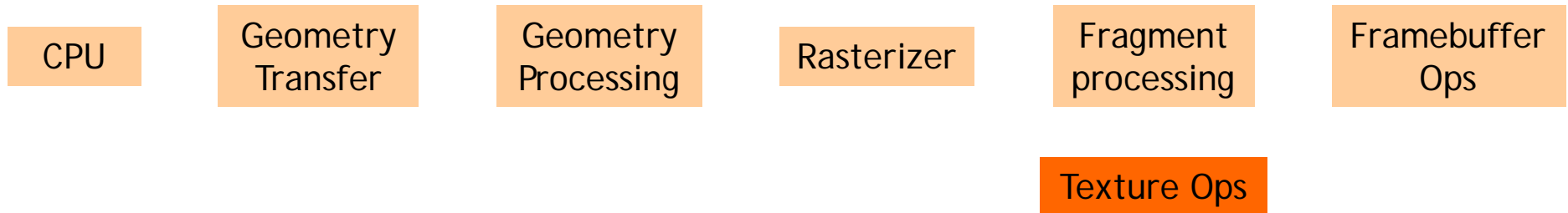
- Increase speed
 - built-in functions and swizzle operators
 - branching expensive
 - mix math and texture access
 - 8 math ops in parallel per texture access
- Move bottleneck
 - shift computation to VP
 - GPGPU often fill-rate limited, not vertex-pipeline limited
 - shift linear interpolation to rasterizer

Sources of bottlenecks: Fragments



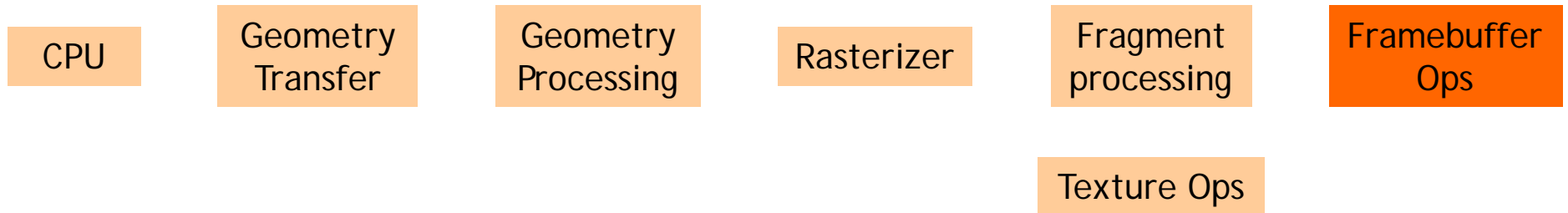
- Decrease fragments
 - draw in rough front-to-back order
 - z-only 1st pass may help
 - masking color ops fast
 - masking only some color channels slow

Sources of bottlenecks: Texture



- Expensive
 - cache-unfriendly access
 - dependent texture reads
 - high precision
 - large non-power-of-two textures
 - ...

Sources of bottlenecks: Framebuffer

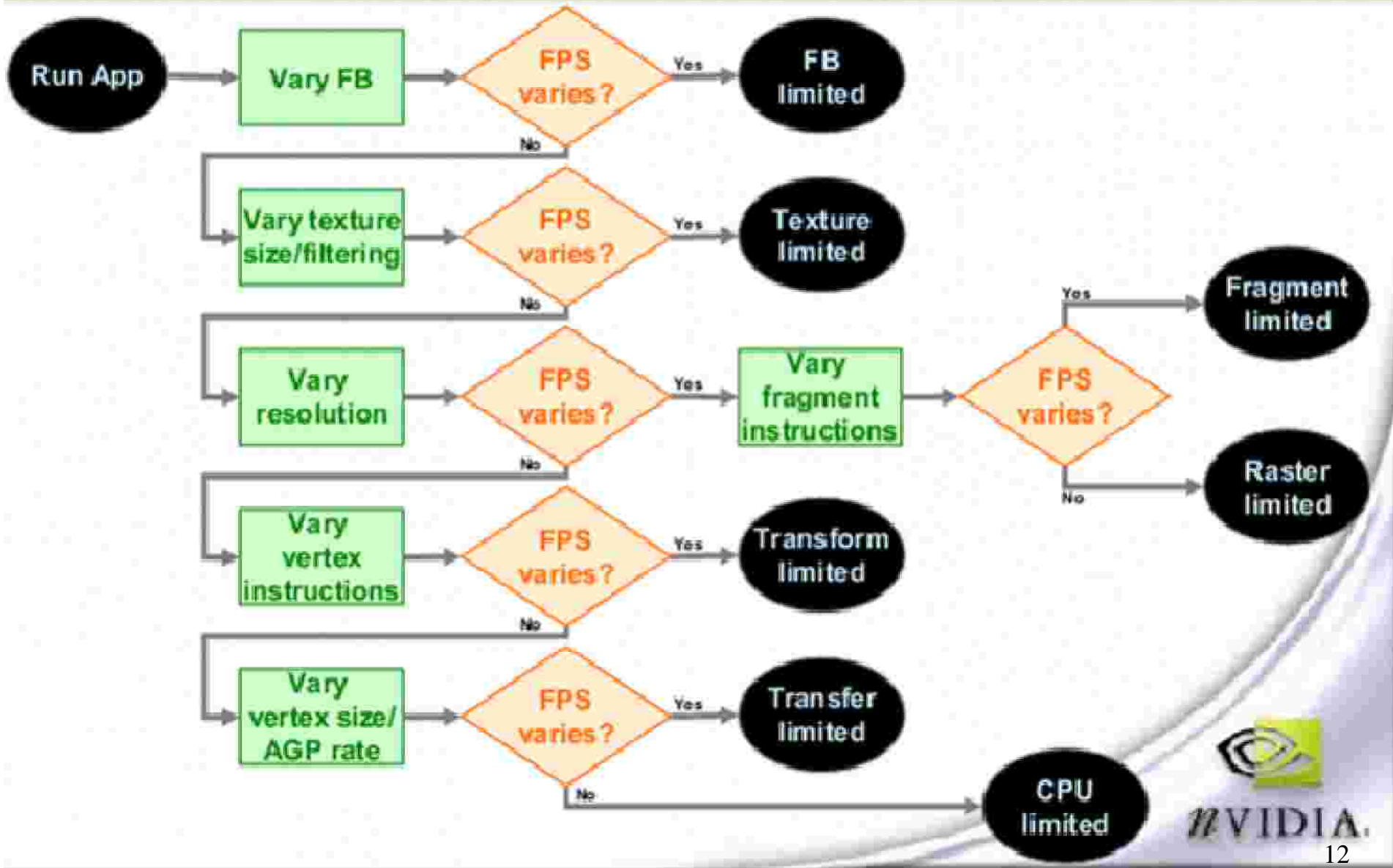


- 16 bits cheaper than 32
- floating point expensive
- blending ops can become bottleneck

Find the bottleneck !

- Increase framebuffer resolution
 - tells you if rasterizer onward is bottleneck
- Change fragment instructions
 - checks fragment processor load
- Reduce texture precision
 - checks texture load
- Change vertex instructions
 - checks vertex processor load
- Change vertex sizes
 - checks transfer
- **Balanced pipeline is goal**

Bottleneck Identification



Tools: GPUBench

- toolkit of programs that probe the GPU
 - Measure download speeds
 - transfer from CPU to texture memory
 - Measure readbacks speeds for different kinds of RenderTextures
 - Evaluate costs of various instructions
 - Multiple parameters that show variation with size of quad, form of rendering, cache effects, etc.
 - Evaluating Texture Access
 - To check cache effects
 - repeatedly access the same texture location
 - To check sequential behaviour
 - access texture elements in a row

Tools: NV ShaderPerf (2.0)

- Takes a shader file in appropriate format
 - produces scheduling information (#clock cycles, #instructions) for running the shader on the GPU
- Can take different architectures as target, yielding different bandwidths

```
float2 pos1 = float2(passno, texcoords0.y);  
float2 pos2 = float2(texcoords0.x, passno);  
  
return (texRECT(sum, texcoords0) +  
texRECT(tex1, pos1) * texRECT(tex2, pos2)).r;
```

NV31: 66 MP/s

NV40: 1.6 GP/s

Tools: NV PerfHUD (6.0)

- Only works with DirectX
- Provides much more detailed shader profiling
- Enable it by choosing a specific target in Cg code


Tools: Graphic Remedy gDebugger


- Breakpoints, stepping
- Watch windows
- Performance analysis
- OpenGL

gDebugger

```
OpenGL calls log - Test App - Context 1 - generated by gDebugger - Microsoft...
File Edit View Favorites Tools Help
Address Q:\Management\Website\images\Context1-OpenGL.CallsLog.html
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// This File contain an OpenGL calls log
// Application: Test App
// Generation date: Sunday, January 27, 2005
// Generation time: 15:03:50
// Context id: 1
//
// Generated by gDebugger - an OpenGL Debugger
// www.gremedy.com
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

vglSwapBuffers(0x1901215A)
glClear(0x4100)
glLoadIdentity()
glRotatef(-90.00, 1.00, 0.00, 0.00)
glRotatef(83.99, 0.00, 0.00, 1.00)
glLightfv(GL_LIGHT0, GL_POSITION, (0.00, 0.71, 0.71, 0.00))
glTranslatef(-101.60, -100.09, -1.35)

glBindTexture(GL_TEXTURE_2D, 2) [Context 1 - Texture 2: ]
glTexEnvfv(GL_TEXTURE_ENV, GL_TEXTURE_ENV_COLOR, (1.00, 1.00, 1.00, 0.00))
glEnable(GL_TEXTURE_2D)
glColor3ub(48, 64, 176)
glDisable(GL_CULL_FACE)
glDisable(GL_DEPTH_TEST)
glCallList(1)
glEnable(GL_DEPTH_TEST)
glEnable(GL_CULL_FACE)

glBindTexture(GL_TEXTURE_2D, 1) [Context 1 - Texture 1: ]
glTexEnvfv(GL_TEXTURE_ENV, GL_TEXTURE_ENV_COLOR, (0.10, 0.10, 0.10, 0.00))
```

Acknowledgements

- John Spitzer
- Koji Ashida
- Suresh Venkatasubramanian
- Tim Purcell