

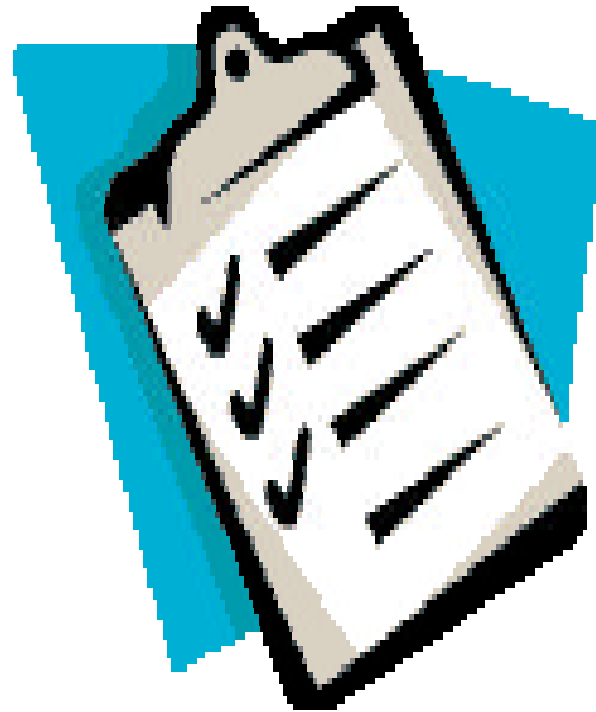


**CS 563 Advanced Topics in
Computer Graphics
*Pipeline Optimization***

by Mike Schmidt

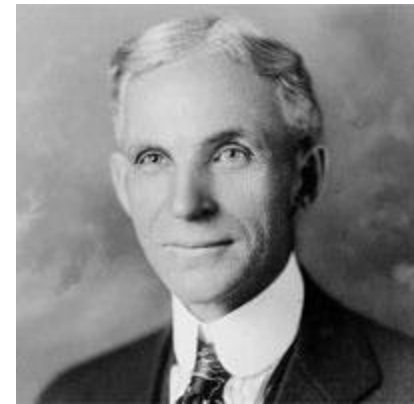
Outline

- The Assembly Line
- Pipelines Defined
- Pipeline Problems
- The Graphics Pipeline
- Finding Bottlenecks
- Bottleneck Stages
- Optimization Techniques
- Performance Tools



The Assembly Line

- The first Ford car plant used runners to transport parts from machines to assemblers.
- Then, a conveyor belt was used to transport parts to the assemblers.
- Bottlenecks were discovered in some areas such as the engines and transmissions.
- Another assembly line was set up for the bottleneck areas.



Assembly Improvement

- Finally, a continuous chain was installed to deliver entire chaises to the assemblers.
- In this way, assembly time was cut from 17 hours to 1.5 hours.





What is a Pipeline?

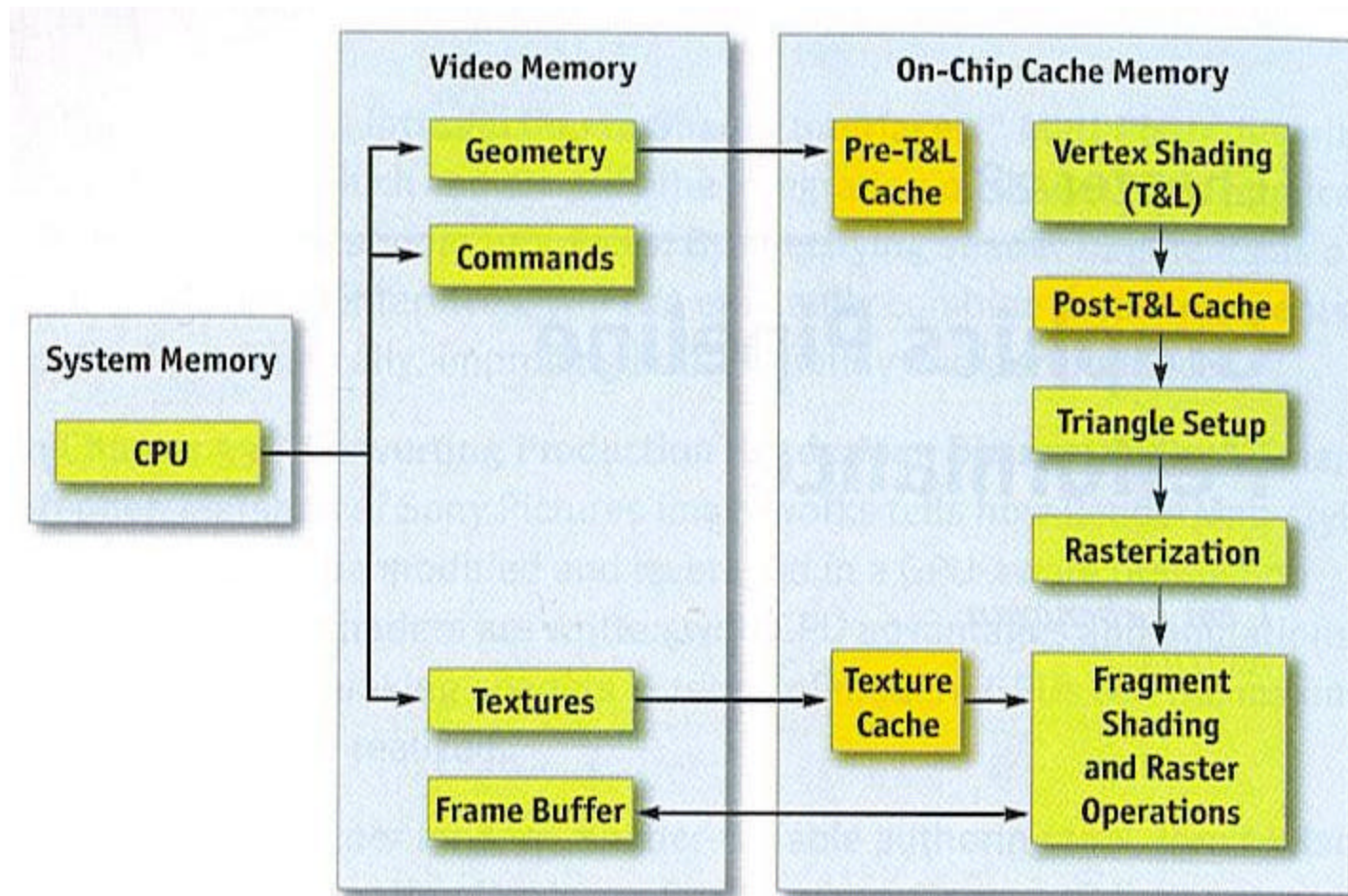
- In a pipeline, multiple instructions are overlapped in execution.
- The ideas and principles in Ford's assembly line are the same.
- Instead of executing one instruction at a time, different stages of n instructions are executed simultaneously.
- If stages are perfect, then the performance speedup is equal to the number of stages.



Pipelining Pitfalls

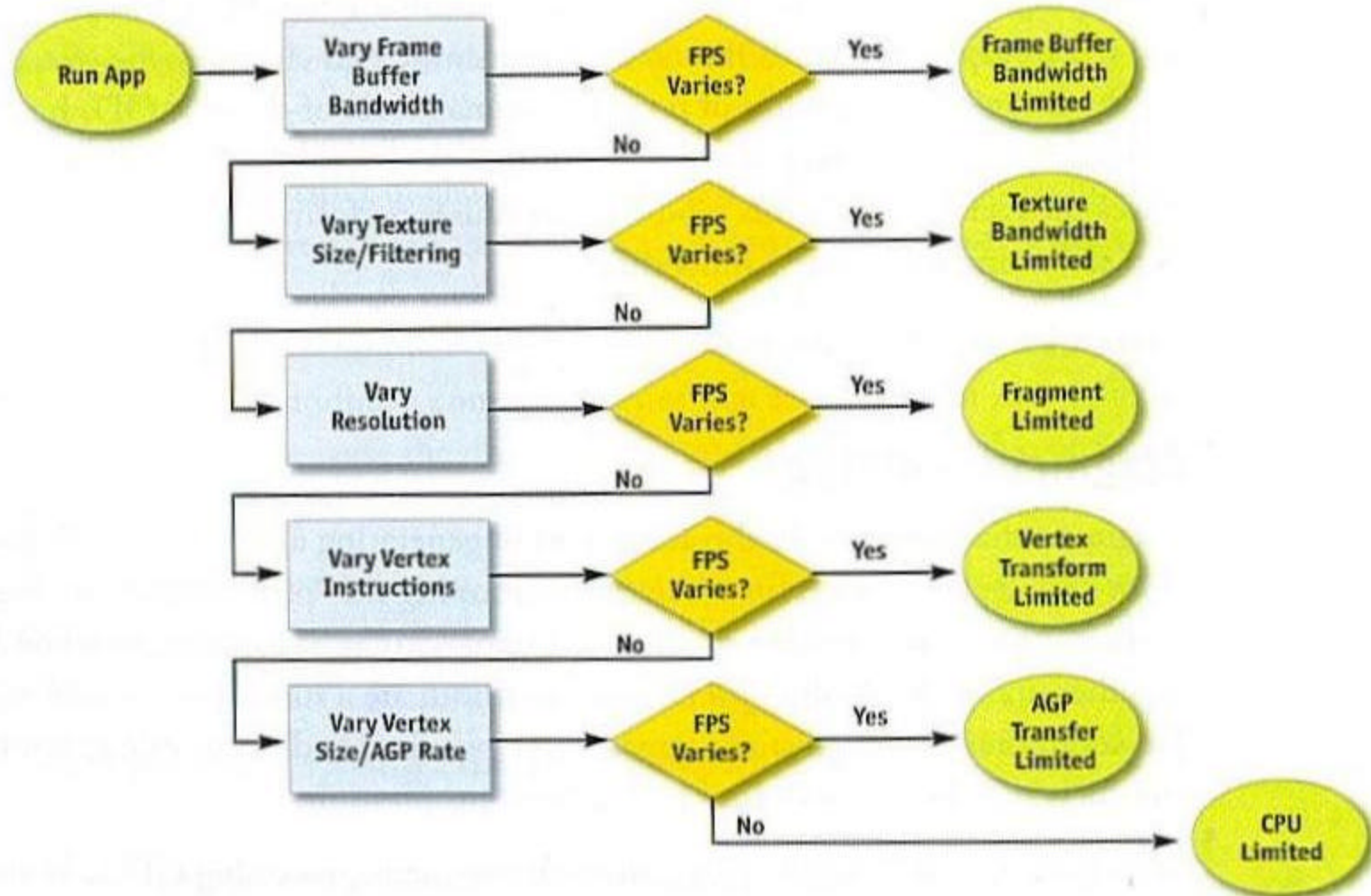
- Unfortunately, pipelines are not perfect.
- Hazards introduce delay into the pipeline.
- The three classes of hazards are structural, data, and control.
- Structural hazards occur when an instruction stage conflicts with a previous instruction stage.
- Data hazards occur when the next instruction needs a result from the previous instruction.
- Control hazards occur when a branch is executed.

The Graphics Pipeline



From GPU Gems

Testing Methodology



From GPU Gems



Finding Bottlenecks

- The bottleneck is the slowest stage in the pipeline.
- The first step of optimization is finding the bottleneck.
- In a graphics program, the bottleneck can be the application, geometry, rasterizer, or communication stages.
- A performance improvement in the bottleneck will result in a speedup.
- Performance improvement in the other areas will not result in a speedup.

Measuring Performance

- Advertised peak rates are usually impossible to achieve.
- Instead of measuring pixels/second or vertices/second, measure frame rate.
- Some drivers optimize performance at the cost of quality.
- Use a real scene and not a special-case, fabricated scenario.





Testing the Application Stage

- On Windows machines, the Task Manager gives processor utilization.
- If the utilization is 100%, this stage may be the bottleneck (unless there is a busy-wait).
- A code profiler can give execution time statistics.
- Replacing GL calls with null calls will remove graphics stages from the pipeline.
- Replace API calls with lighter-weight API calls (e.g. glColor3fv instead of glVertex3f)



Testing Vertex Bandwidth

- The first stage of the pipeline is retrieving the vertex information from memory.
- The vertex information is either in frame buffer memory or local memory.
- Changing the vertex format size will determine whether this stage is the bottleneck.



Testing the Geometry Stage

- The geometry stage is the most difficult to test.
- Lighting, texture coordinate generation and fog could be disabled.
- To test this stage, add or remove light sources and monitor the performance.
- Another approach involves determining if a simple vertex program results in a performance improvement.

Testing the ROP Stage



- The raster operations stage is responsible for reading and writing depth, stencil, and color.
- One testing method is to vary the bit depth of color or depth buffers (e.g. go from 32 bits to 16 bits).
- The speed of the memory clock can also be modified to test this stage.



Testing the Fragment Stage

- The fragment stage includes the time spent running fragment programs.
- This stage is separate from the ROP stage.
- Complex pixel shaders may make this stage the bottleneck.
- Decreasing the resolution of the viewport is one method.
- Another method is to use simpler fragment programs.



Testing Texture Bandwidth

- Fetching textures from memory can also be a bottleneck.
- One method involves changing the mipmap function to use less detailed mipmap levels.
- This reduces the effective texture size.
- Another method is modifying the memory clock.

Optimizing Bottlenecks

- Once the bottleneck stage has been identified, it must be optimized.
- Either use a better algorithm or reduce image quality.
- Look for the bottlenecks within the bottleneck.





Application Optimization

- Multiply by $1/\text{length}$ instead of dividing by length.
- Align structs on word boundaries.
- Use inline, const, and pass by reference.
- Predecrement instead of postdecrement.
- Take advantage of spatial locality of memory.
- Write your own memory manager.
- Avoid redundant computations.
- Use the most efficient algorithms for sorting, etc.



Vertex Bandwidth Optimization

- Use the smallest sufficient format.
- Compute some vertex attributes.
- Use shorter indices.
- Use spatial locality of memory references.
- Vertex arrays are the fastest to access.
- Triangle Strips are faster than unconnected triangles.



Geometry Optimization

- Eliminate superfluous light sources.
- Don't light polygons unnecessarily.
- Consider using environment maps.
- Preprocess normals (don't normalize).
- Simplify vertex programs.
- Don't swap vertex programs needlessly.
- Don't try to optimize a compiled vertex program (unless you REALLY know what you are doing).
- Exit out of a vertex program if possible.



Fragment Optimization

- Perform a depth rendering pass first.
- Use textures for lookup tables.
- Don't do anything in a pixel shader that can be done in a vertex shader.
- Render in front-to-back order.
- Use the smallest precision possible.



Texture Optimization

- Reduce the size of textures.
- Compress textures.
- Use efficient formats.
- Favor `glTexSubImage` to `glTexImage`.
- Use multitexturing instead of multiple passes.
- Use mipmaps.
- Use cheaper filtering.



Frame-Buffer Optimizations

- Reduce alpha blending.
- Disable depth testing after first pass when depth is known.
- Don't call `glClear` if the entire viewport will be overwritten.
- Render front to back.
- Use smaller precision formats.

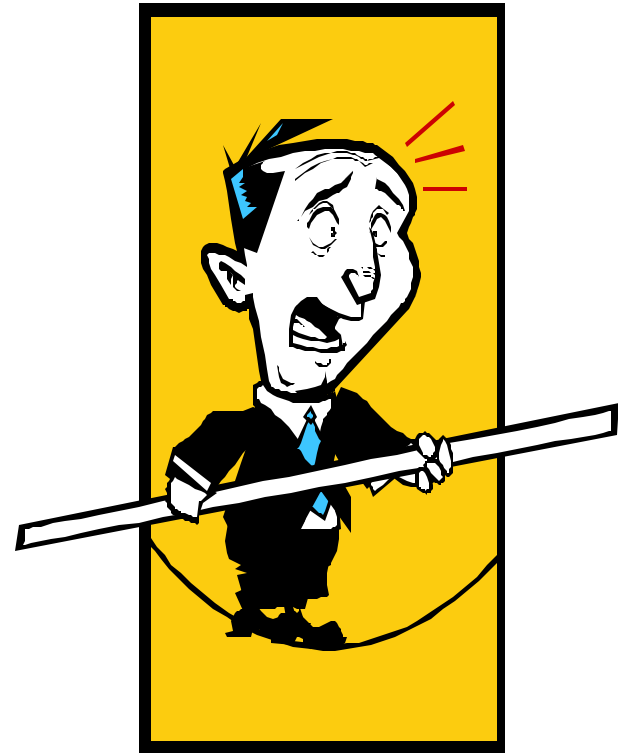


General Optimizations

- Know your architecture.
- Reduce the number of primitives.
- Turn off unused features.
- Preprocess whenever possible.
- Minimize API calls.
- Make sure textures are in texture memory.
- Don't read from the frame buffer.
- Use display lists.

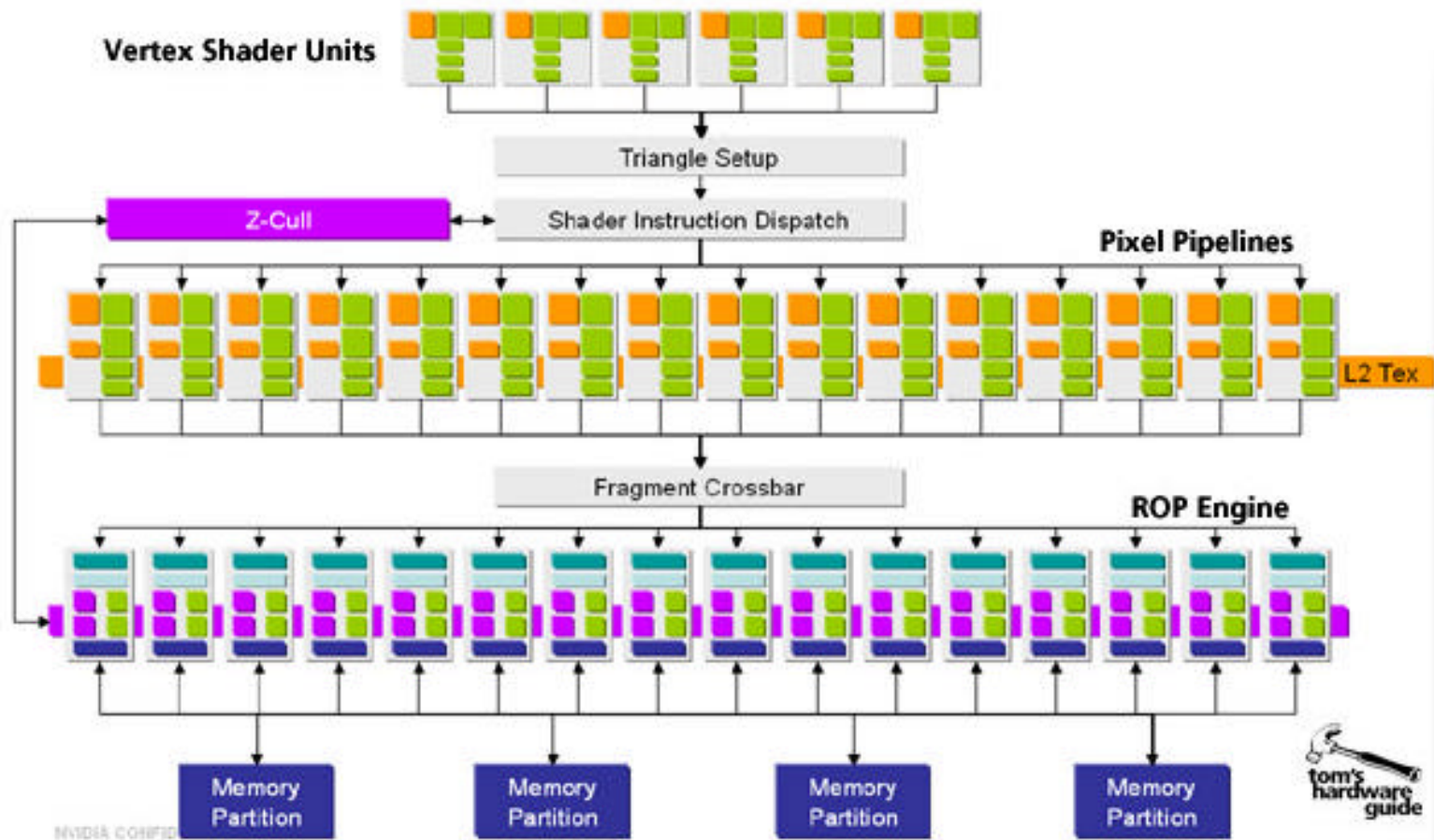
Balancing Act

- Add advanced features to stages that are not bottlenecks.
- Add advanced physics to the application stage.
- Use more lights in the geometry stage.
- Use more expensive texture filtering.
- Add more triangles.



Modern Hardware Example

GeForce 6800 series 3D Pipeline



From Tom's Hardware



Optimization Tools

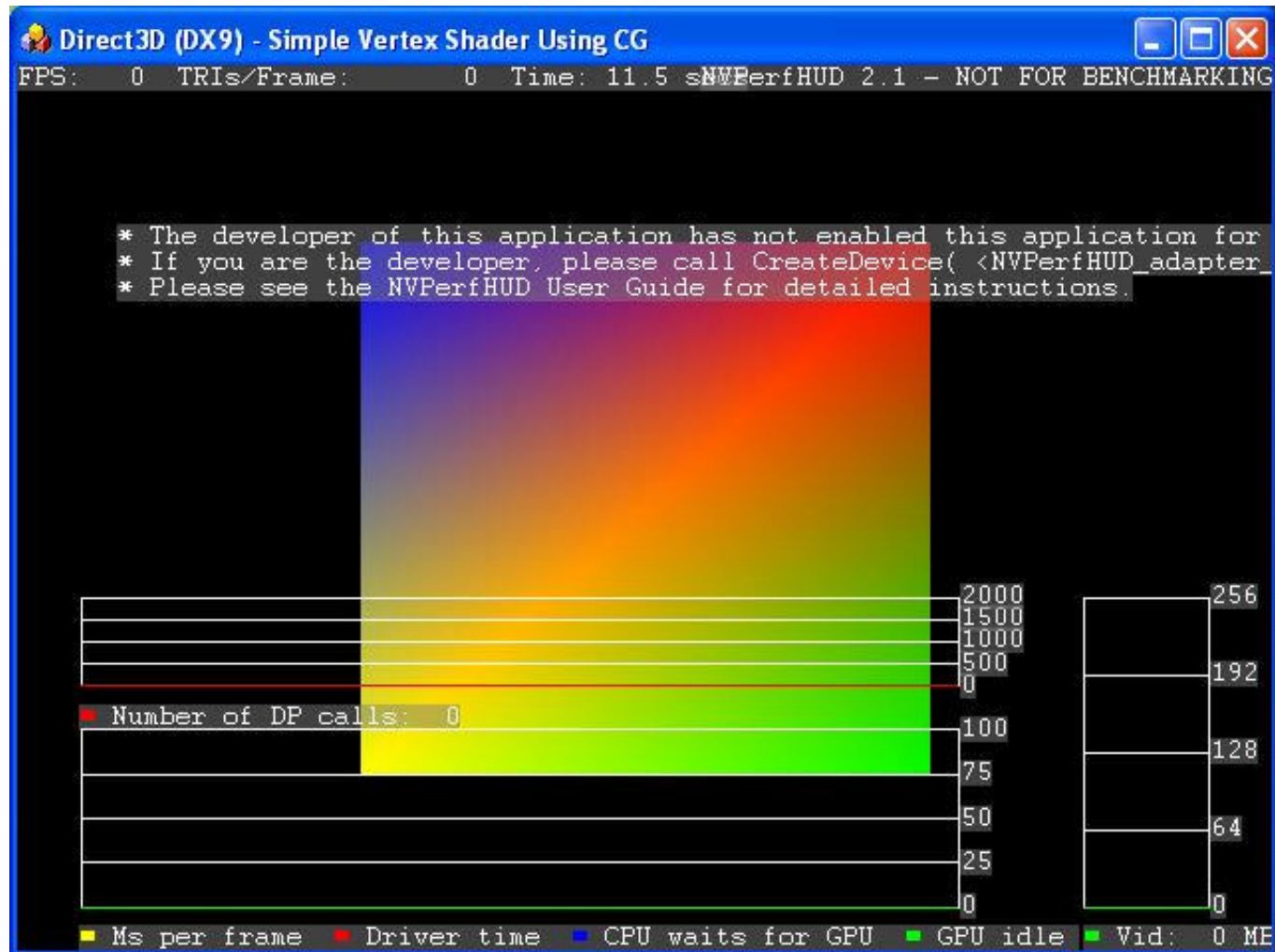
- NVPerfHUD
- VTune
- FX Composer
- PowerStrip (useful for modifying the GPU's clock)




NVPerfHUD

- NVPerfHUD works with DirectX 9.0 applications.
- Statistics are provided on graphs similar to a heart monitor while the application runs.
- The collected statistics can be used to fix bottlenecks.
- Analysis must be enabled through the API.

NVPerfHUD Screenshot

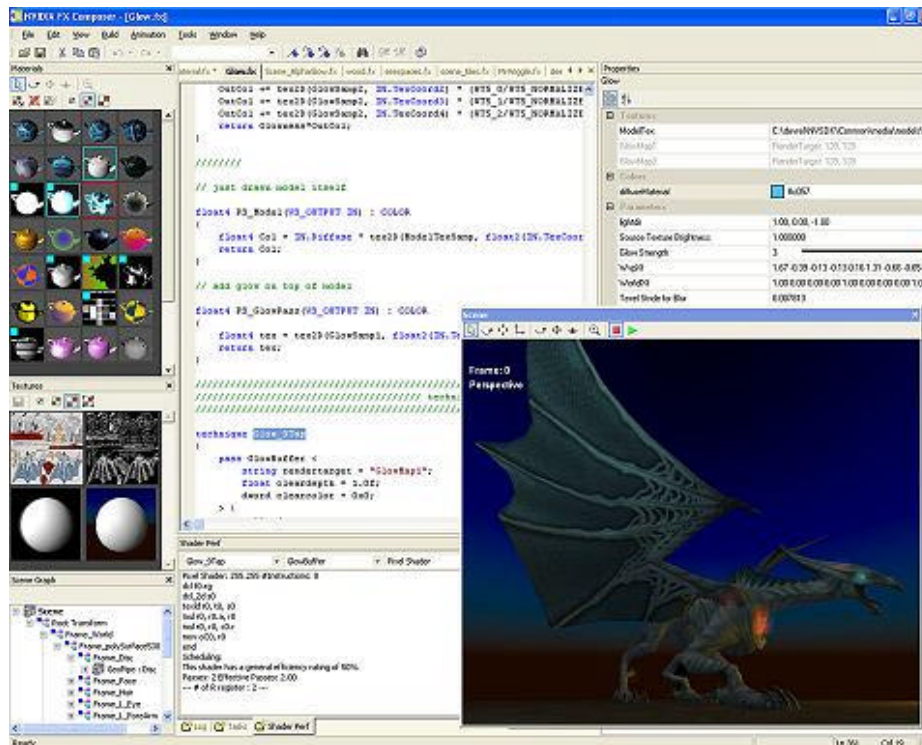


Performance analysis is not enabled.

- 
- VTune is a robust code analysis tool from Intel.
 - Call graphs and execution time are provided.
 - Allows developers to optimize the application stage.

FXComposer

- Allows developers to create shaders.
- Shaders can be debugged.
- Performance analysis and optimization hints help eliminate bottlenecks.



From NVidia

References

- <http://www.pbs.org/wgbh/aso/databank/entries/dt13as.html>
- http://www.aaca.org/history/assmbl_1.htm
- T. Akenine-Möller, E. Haines, "Real-Time Rendering", 2nd ed., A K Peters, 2002
- R. Fernando, "GPU Gems", Addison-Wesley, 2004
- J. Hennessy, D. Patterson, "Computer Architecture", 3rd ed., Morgan Kaufmann, 2003
- <http://www.tomshardware.com/>
- <http://www.intel.com/software/products/vtune/vpa/>
- <http://developer.nvidia.com/page/home.html>