Shader Model 3.0 - No Limits

Updated: April 5, 2004
By D. Sim Dietrich Jr., Nvidia

Microsoft® DirectX® 9.0 introduced several new standards for advanced vertex and pixel shader technology, version 2.0 and version 3.0. Shader Model 2.0 hardware has been available for over a year, and both hardware and software support is growing rapidly. Shader Model 2.0 includes technologies useful for advanced lighting and animation techniques, but has limited shader program length, and complexity, which limits the fidelity of the effects that can be achieved.

As developers push against the limits inherent in Pixel Shader 2.0 and Vertex Shader 2.0, they have started to adopt the newer, more advanced Shader Model 3.0. This shader model has advances in several areas, in both pixel and vertex shader processing.

The following is a feature summary outlining the key differentiators between Pixel Shader 2.0 and 3.0.

Pixel Shader Feature	Shader 2.0	Shader 3.0	Description
Shader length	96	65535+	Allows more complex shading, lighting, and procedural materials
Dynamic branching	No	Yes	Saves performance by skipping complex shading on irrelevant pixels
Shader anti-aliasing	Not supported	Built-in derivative instructions	Developers can calculate the screen space derivatives of any function, allowing them to adjust shading frequencies or over-sampling to eliminate artifacts
Back-face register	No	Yes	Allows two-sided lighting in a single pass
Interpolated color format	8-bit integer minimum	32-bit floating point minimum	Higher range and precision color allows high-dynamic range lighting at the vertex level
Multiple render targets	Optional	4 required	Allows advanced lighting algorithms to save filtering and vertex work – thus more lights for minimal cost
Fog and specular	8-bit fixed function minimum	Custom fp16-fp32 shader program	Shader Model 3.0 gives developers full and precise control over specular and fog computations, previously fixed-function
Texture coordinate count	8	10	More per-pixel inputs allows more realistic rendering, especially for skin

Here is a similar listing of key features developers enjoy when moving from Vertex Shader Model 2.0 to 3.0.

Vertex shader feature	Shader 2.0	Shader 3.0	Description
Shader length	256 Instructions	65535 instructions	More instructions allow more detailed character lighting and animation
Dynamic branching	No	Yes	Saves performance by skipping animation and calculations on irrelevant vertices
Vertex texture	No	Any number of lookups from up to 4 textures	Allows displacement mapping, particle effects
Instancing support	No	Required	Allows many varied objects to be drawn with only a single command

One major feature of both Shader 3.0 models (vertex and pixel) is Dynamic Branching. Put simply, this allows a shader author to create true loops and conditionals in their shader programs. For instance, one could write a shader that looped through a certain number of vertex lights, determine which ones might influence a particular vertex, and then pass down the index of each relevant light to the pixel shader. The pixel shader could then use this 'light index' to determine which light parameters to apply. The pixel shader would then loop over the active lights, then use dynamic branching to exit the shader early once all lights are processed.

Most light types only apply to the front side of an object—the side facing the light. Therefore, you can use both vertex and pixel branching to skip processing for lights that the shader detects as facing away from the light. This can save significant processing time, and speed up the shader. Similar speedups can be used to skip processing of character bone animation as well as many similar algorithms.

As game engines become more and more complex, they often create many different versions of each shader in order to fit them all in to the Pixel Shader 2.0 program length limitations. This can add to code maintenance, as well as take up valuable system memory at runtime. Shader Model 3.0 eliminates this issue, through its comprehensive looping and branching, allowing the engine to write a single vertex and single pixel shader containing appropriate static and dynamic branching in order to select the correct execution path at runtime, thus greatly simplifying the shader combinatorial explosion issue.

Another key feature of Shader Model 3.0 is the support for the Microsoft DirectX® Instancing API. Currently, games face limits on the number of unique objects they can display in the scene, not because of graphics horsepower, but often the CPU-side overhead of either storing or submitting many slightly different variations of the same object. For instance, a forest is made up of trees that are often similar to each other, but each would be in a different position, have differing height, branch length, leaf color, and so on. In order to add the desired variation, developers have to choose between storing many

separate copies of the tree, each slightly different, or making expensive render state changes in order to rotate, scale, color and place each tree.

Instancing allows the programmer to store a single tree, and then several other vertex data streams to specify the per-instance color, height, branch size and so on. For instance, a single 1000-vertex tree model would contain the vertex positions and normals, and a 200-element vertex streams would contain positions, colors, heights, and branch length values. Instancing allows the programmer to submit a single draw call, which renders each of the 200 trees, using the same data for the basic tree shape, but then vary it through the per-instance streams.

In summary, DirectX 9.0 Shader Model 3.0 is a significant step forward in terms of ease of use, performance, and shader complexity. Dynamic branching brings speed-ups to many algorithms which contain early-out opportunities, while also simplifying shader code paths in graphics engines and tools. Lastly, instancing allows extreme complexity for very low CPU and memory overhead.