Instancing Tests

Bryan Dudash
bdudash@nvidia.com

14 February 2007

Document Change History

Version	Date	Responsible	Reason for Change
1.0	3/14/08	Bryan Dudash	Initial release

Abstract

Instancing is an important tool for direct developers to help squeeze extra performance out of their complex 3D scene. It makes use of special hardware and low level driver code to reduce the overhead in drawing multiple copies of the same vertex buffer. In DirectX 10, the concept of instancing is built into the standard draw call.

InstancingTests provides a simple toolkit to performance analysis of instancing for various meshes and on various graphics chips.

Motivation

Our goal with this sample is to provide a easy to use tool for benchmarking various instancing techniques against straight draw calls. It is not complicated, and can be customized by the user to run different meshes and/or shader code.

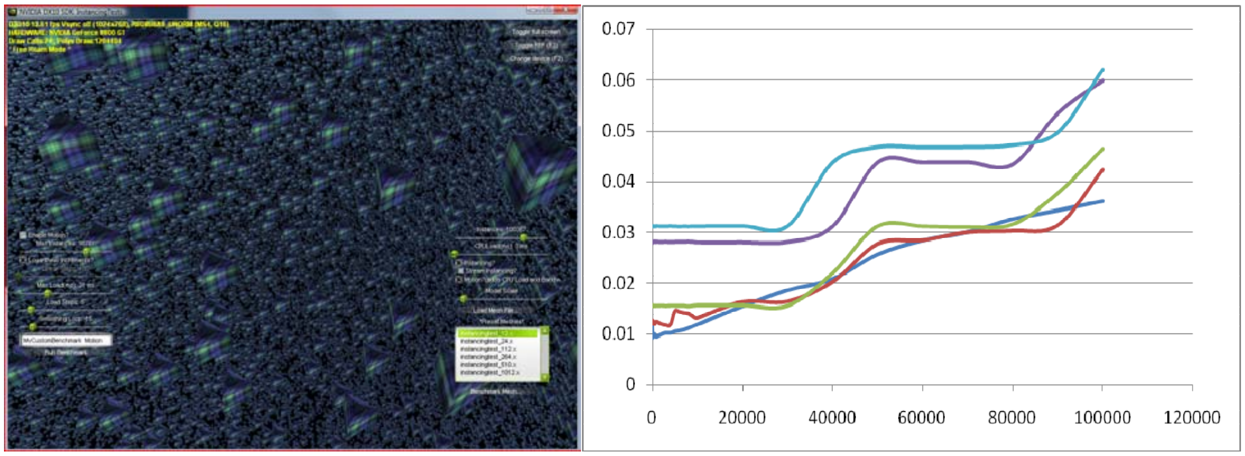


Figure 1:

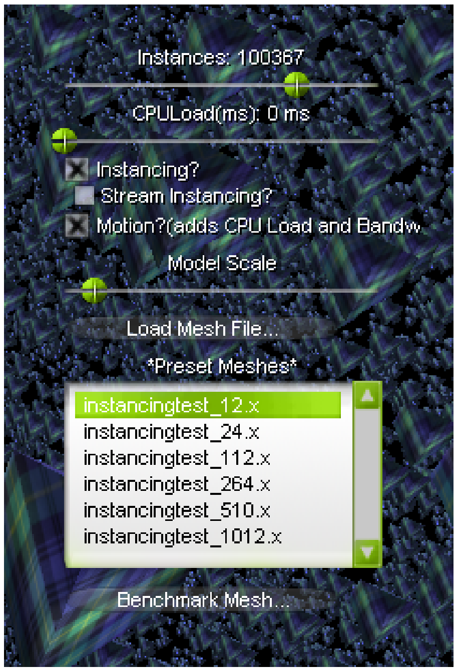
Left: Screen shot of InstancingTests in action.

Right: Graph generated in Microsoft Excel from CSV data dumped from InstancingTests. This particular graph shows the behavior of various CPU load levels on ConstantsBased Instancing of a 12 poly mesh as the # of instances varies. The Y axis is frametime. This graphic shows some interesting spikes as instances increase when under CPU load. Straight blue line is under no load.

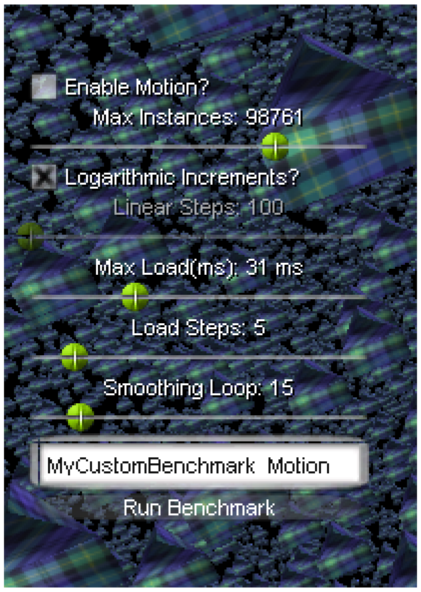
Usage Guide

This whitepaper will function more like a user guide. The concepts behind instancing are better covered in other texts.

The interface of InstancingTests contains two separate and distinct UI areas. On the right is the "Interactive" UI. This UI is active when the user is dynamically moving the camera and changing various options about the scene. The UIs are shown below.



Option	Description	Notes
Instances	The number of meshes to render	Upper limit based on multiples of how many instances can fit in a constant buffer.
CPU Load	Add artificial CPU load in milliseconds	Busy loop until the time is passed. Does not sleep.
Instancing?	Use instancing or not	Off means single draw calls.
Stream Instancing?	This enables use of "traditional" vertex stream based instancing	Off means constants based instancing
Motion?	Simple random oscillation of all meshes	Forces instance data buffer to update every frame
Load Mesh File…	Specify your own .x mesh to load	Opens a file dialog
Preset Meshes	A handy set of sphere meshes.	The number after the "_" indicates the poly count.
Benchmark Mesh…	Open the benchmark UI	Uses the currently loaded mesh.



Enable Motion?
Max Instances: 98761
Logarithmic Increments?
Linear Steps: 100
Max Load(ms): 31 ms
Load Steps: 5
Smoothing Loop: 15
MyCustomBenchmark Motion
Run Benchmark

Option	**Description**	**Notes**
Enable Motion	Set motion on or off for the benchmark	
Max Instances	Max for this benchmark run	Max capped by global max
Logarithmic Increments	10 steps from each factor of 10.	1,2…9,10,20…90,100,200… etc.
Linear Steps	Break range from 1 to max into a # of steps	Only used when not logarithmic
Max Load(ms)	Load is also tested across a range	Always linear
Load Steps:	Break load range into # steps	Capped by max load
Smoothing Loop	# of frames to average over	Loop count
Benchmark (EditBox)	Custom Benchmark Tag	Inserted into the output filename
Run Benchmark	Execute with above parameters	When finished results saved to a file in CWD.

Note

The output filename with the results is generated to be of the following form

InstancingTests_<BenchmarkTag>_i<Max Instances>_l<Max Load>.csv

DirectX 10's Draw overhead

This tool is designed to allow the user to get an idea of instancing performance gains and to do performance analysis. However, it is likely that there are some common results that are worth discussing. Microsoft has removed a lot of runtime verification of draw calls, and as a result the per-call overhead of Draw*() under DirectX 10 is considerably lower than under previous DirectX versions. Nevertheless, there are still cases where using instancing will provide a significant performance due to reduction of CPU load. In addition, given the fact that GPU power is increasing faster than CPU power, the benefits of reducing CPU load will grow. The bottom line is that because each Draw call has some overhead, reducing the number of Draw calls will always increase efficiency.

NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, CA 95050
www.nvidia.com