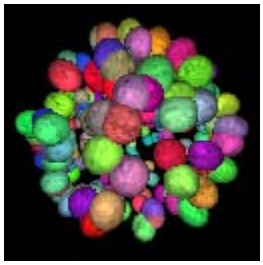
Technical Report

GLSL Pseudo-Instancing



Abstract

GLSL Pseudo-Instancing

This whitepaper and corresponding SDK sample demonstrate a technique to speed up the rendering of instanced geometry with GLSL. The technique relies upon the efficient in-lining of persistent vertex attributes in OpenGL. World transforms (and potentially other instance specific data) are passed down for each instance using texture coordinates instead of uniform variables.

Jeremy Zelsnack
sdkfeedback@nvidia.com

NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, CA 95050

November 17, 2004

Table of Contents

Introduction

Rendering large numbers of geometrically instanced data can be useful for world clutter such as rocks, bushes, trees, crates, and debris. In some genres of games, it can also be useful for weapons and armor.

Unfortunately, rendering large number of instances results in a large amount of driver work. This is particularly true in GLSL where the driver must map abstract uniform variables into real physical hardware registers. From the hardware's perspective, the large number of constant updates is not ideal either. Constant updates can incur hardware flushes in the vertex processing engines.

OpenGL has a notion of persistent vertex attributes that can be passed down by immediate mode calls like glTexCoord(). These API calls are very efficient on the driver side; they don't require validation or potentially complex remapping. They are also very efficient on the hardware side; they do not result in hardware flushes in the vertex processing engines.

The efficiency of persistent vertex attributes can be exploited by passing per-instance data such as transforms, color and other data via OpenGL immediate mode calls. The benefit of this technique can be quite large (especially when rendering instances with a small number of vertices on slow CPUs).

Technique

If you were rendering a large number of instances using GLSL, you could use the modelview matrix stack and use gl_ModelViewProjectionMatrix in your vertex shader. The performance of this might not be that great because of the additional CPU load involved in computing the transform matrices and downloading them to the GPU (most games are CPU bound, so it's usually a good idea to offload the CPU when reasonable).

An alternate approach would be to send down a view matrix once for all of the instances. A world matrix can be sent down for each instance using glUniform4fvARB(), glUniformMatrix4fvARB() or similar calls. This technique requires more computation because each vertex has to be transformed by three matrices (assuming view-space lighting) instead of just one. However, this offloads additional transformation computations from the over-burdened CPU to the GPU. This is one of the techniques employed in the sample.

The pseudo-instancing approach takes a similar, but slightly different approach. Like the above technique, a view transform is passed down for all of the instances. Unlike the above technique, the per-instance world matrices are passed down using glMultiTexCoord() instead of calls like glUniform4fvARB(). This exploits the software and hardware advantages of the persistent vertex attributes. The code sample below demonstrates the technique. For more detailed information, please see the glsl_pseudo_instancing SDK sample.

```
// Render the instances of the mesh
for(i=0; i<instances.size(); i++)
  {
    // Send down the matrix and other parameters
    if(gUseInstancing)
      {
        // pseudo-instancing passes per-instance data down
        // as texture coordinates
        glMultiTexCoord4fv(GL_TEXTURE1, instances[i].mWorld[0]);
        glMultiTexCoord4fv(GL_TEXTURE2, instances[i].mWorld[1]);
        glMultiTexCoord4fv(GL_TEXTURE3, instances[i].mWorld[2]);
      }
    else
      {
        // Traditional GLSL uniform variable downloading
        glUniform4fvARB(WorldMatrixLocation, 3,
                        (float*)(instances[i].mWorld));
      }

    // Set the color for the instance
    glColor4fv(gInstances[i].mColor);

    // Render the instance
    mesh->render();
  }
```

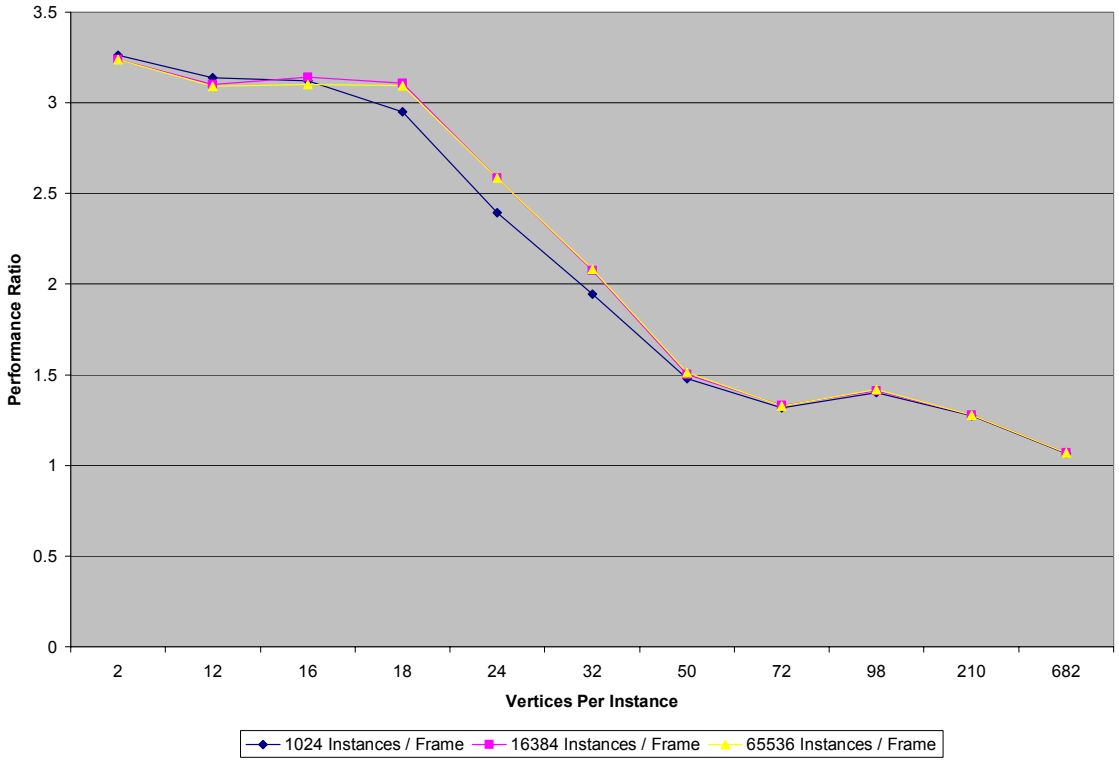
The pseudo-instancing technique is similar to the instancing technique in Direct3D (supported on Shader Model 3.0 GPUs). The major difference is that the Direct3D instancing API reduces the number of DrawIndexedPrimitive() calls from many to one. This DrawIndexedPrimitive() call reduction has a large performance benefit in Direct3D. In OpenGL, the application still calls glDrawElements() (or the like) for every instance. This isn't too much of a performance hit because glDrawElements() is very efficient in OpenGL.

Pseudo-instancing applies well to geometry with a small number of per-instance attributes. The technique does not scale well to complex mesh rendering techniques like skinning; there are not enough vertex attributes to store all of the bone transforms for each instance. The technique is not meant as a replacement for traditional particle system rendering or to replace static meshes where geometry can be reasonably baked into world space.

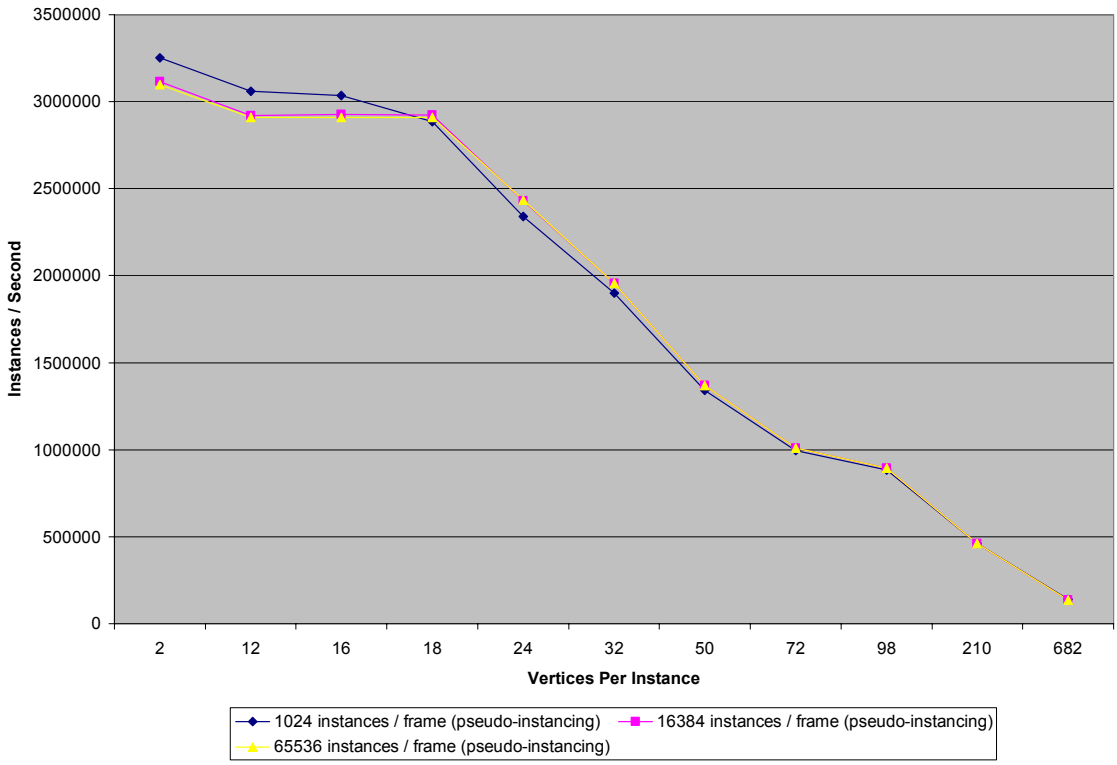
GeForce 6800 GT

The following performance data was collected on a GeForce 6800 GT with 256MB on a PCI-Express 3.4 GHz Hyper-Threaded P4 system using ForceWare driver 66.81. This demonstrates the pseudo-instancing performance on a fast CPU with a fast GPU.

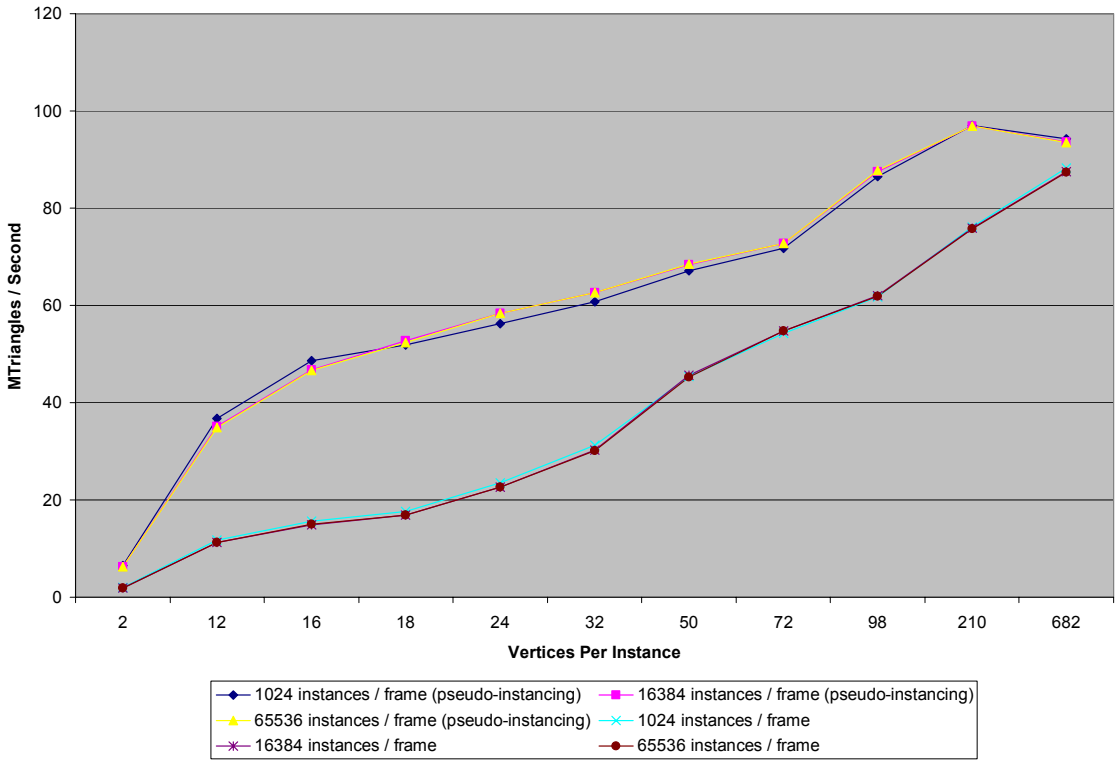
The ratio of instances per second using pseudo-instancing over instances per second without pseudo-instancing is shown below. On this particular setup, the performance difference is significant for small meshes.



The GeForce 6800 GT can hit over 3 million instances per second for tiny meshes.



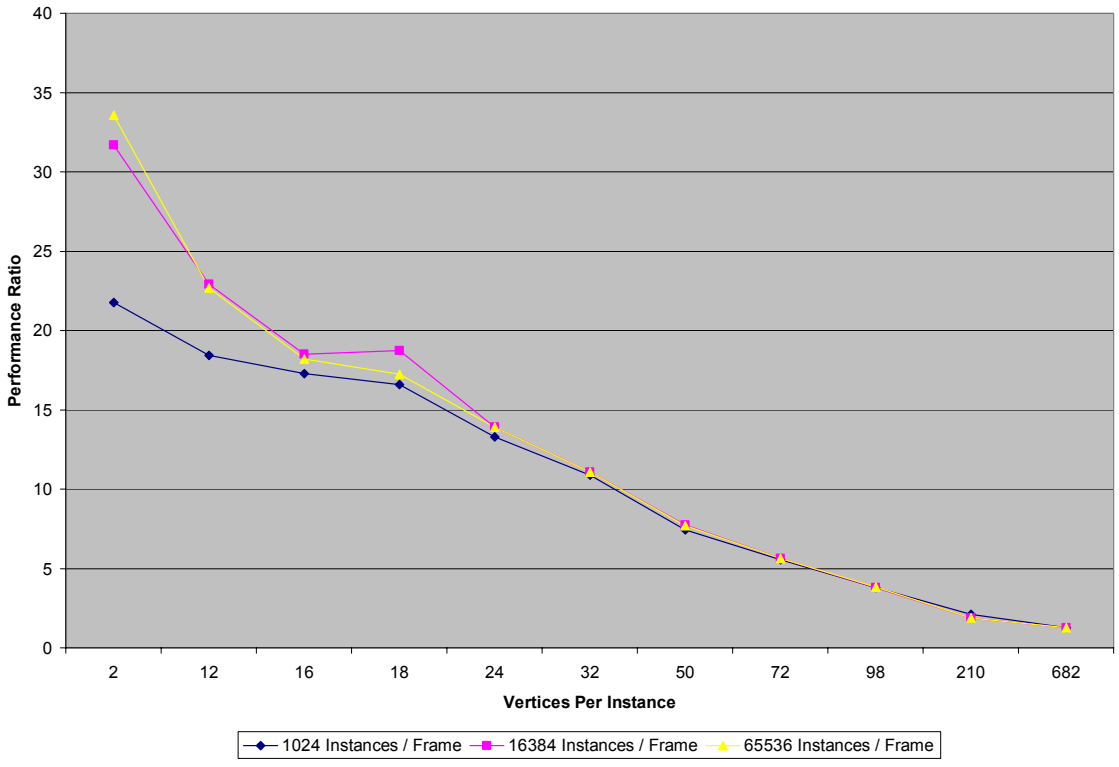
Pseudo-instancing allows the GeForce 6800 GT to render more triangles per second until the mesh size approaches 1,000 vertices.



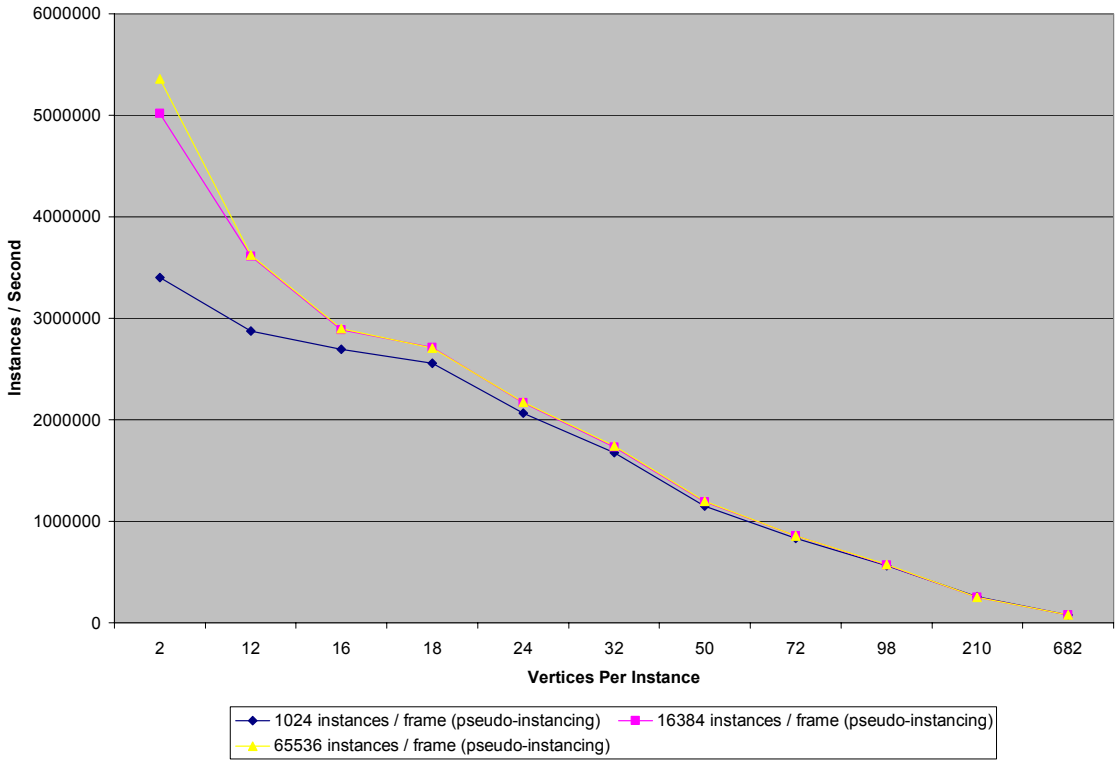
GeForce FX 5900 Ultra

The following performance data was collected on a GeForce FX 5900 Ultra with 256MB on an AGP 8x Athlon XP 2500 system using ForceWare driver 66.81. The configuration represents a moderately fast CPU with a moderately fast GPU

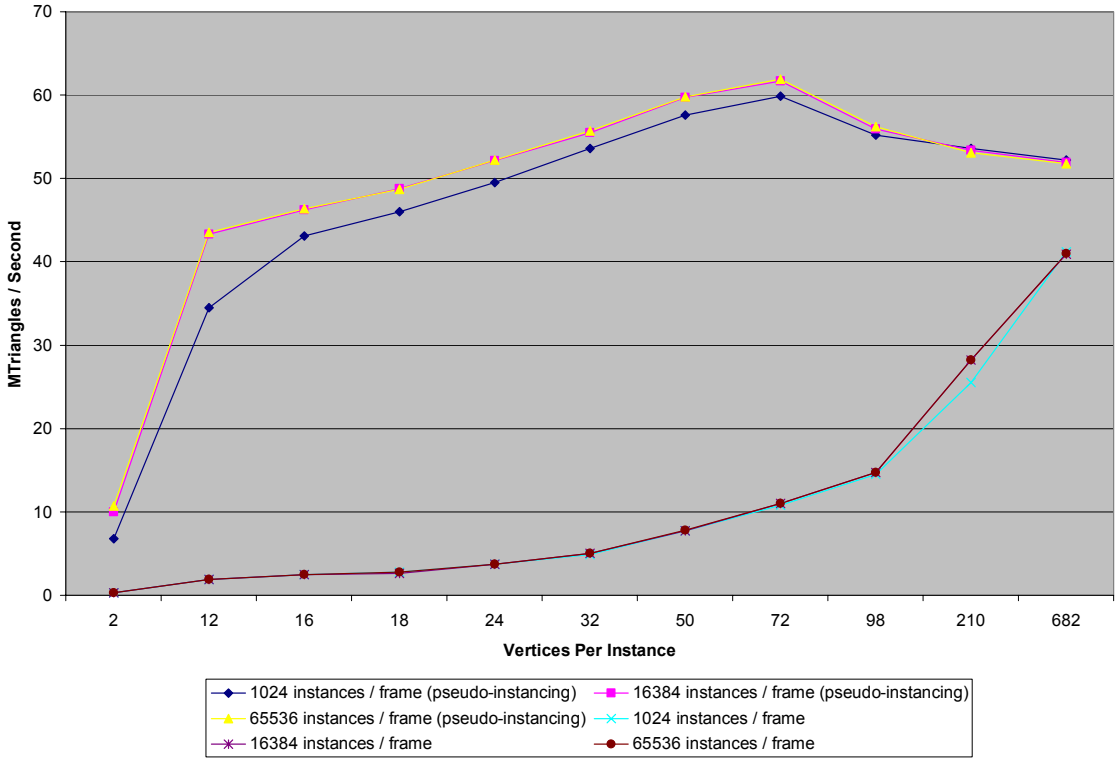
The ratio of instances per second using pseudo-instancing over instances per second without pseudo-instancing is shown below. On this setup, the performance difference is quite dramatic. On tiny instance meshes, the pseudo-instancing performance advantage approaches 35x! For more realistically sized meshes, the performance improvement is still impressive.



The GeForceFX 5900 Ultra can hit over 5 million instances per second when using tiny instance meshes. This is on a system without an incredibly fast CPU.



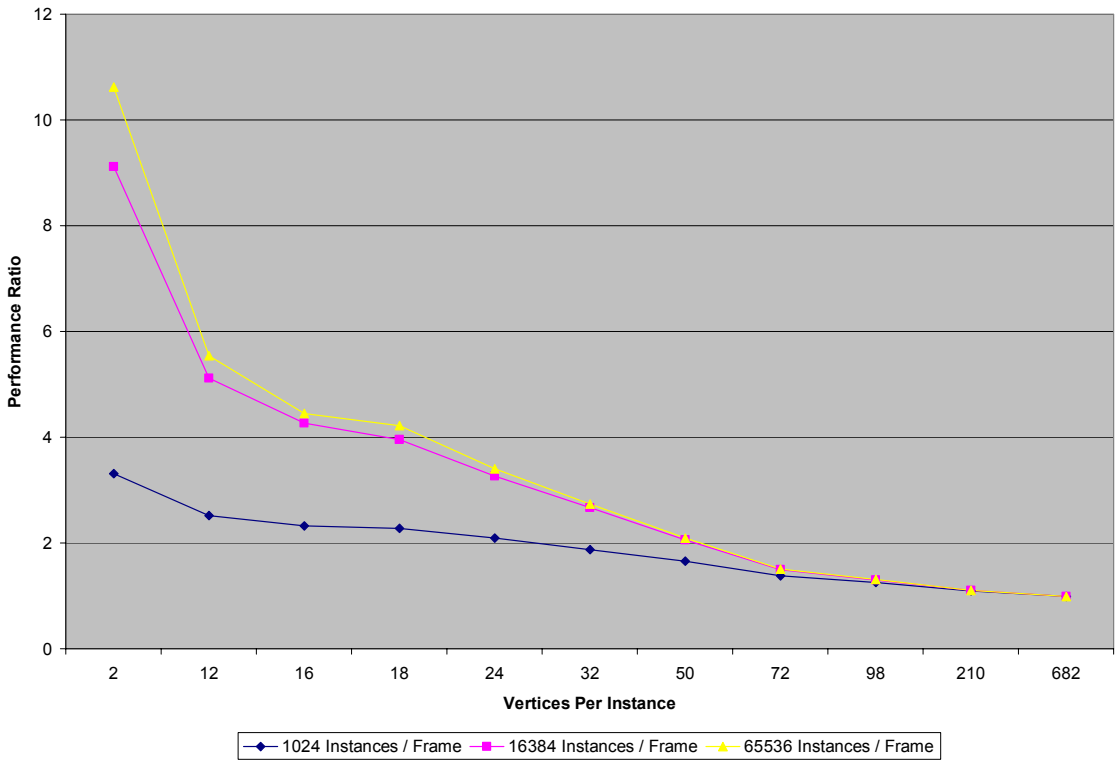
The graph below shows a dramatic difference in the triangle rendering rates between using pseudo-instancing and the traditional rendering method. The downturn in performance for pseudo-instanced meshes with more than 50 vertices is probably due to less than perfect vertex cache friendliness in the generated meshes.



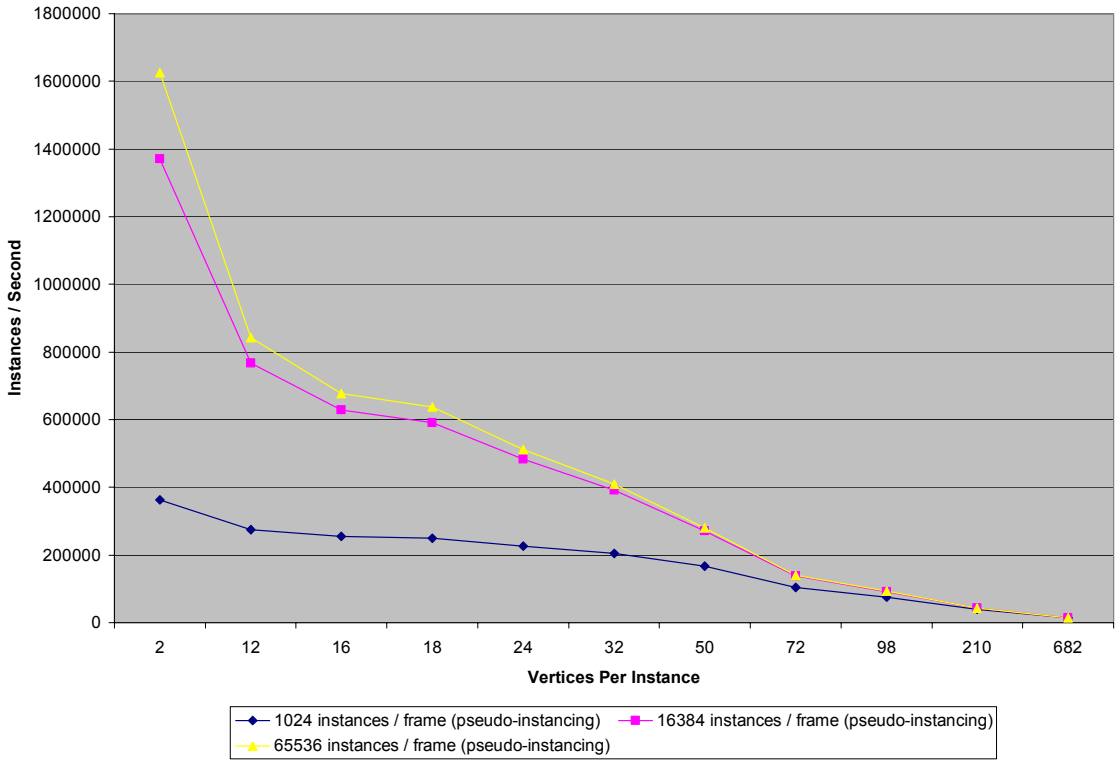
GeForce FX 5200 Ultra

The following performance data was collected on a GeForce FX 5200 Ultra with 128MB on an AGP 8x Athlon XP 2500 system using ForceWare driver 66.81.

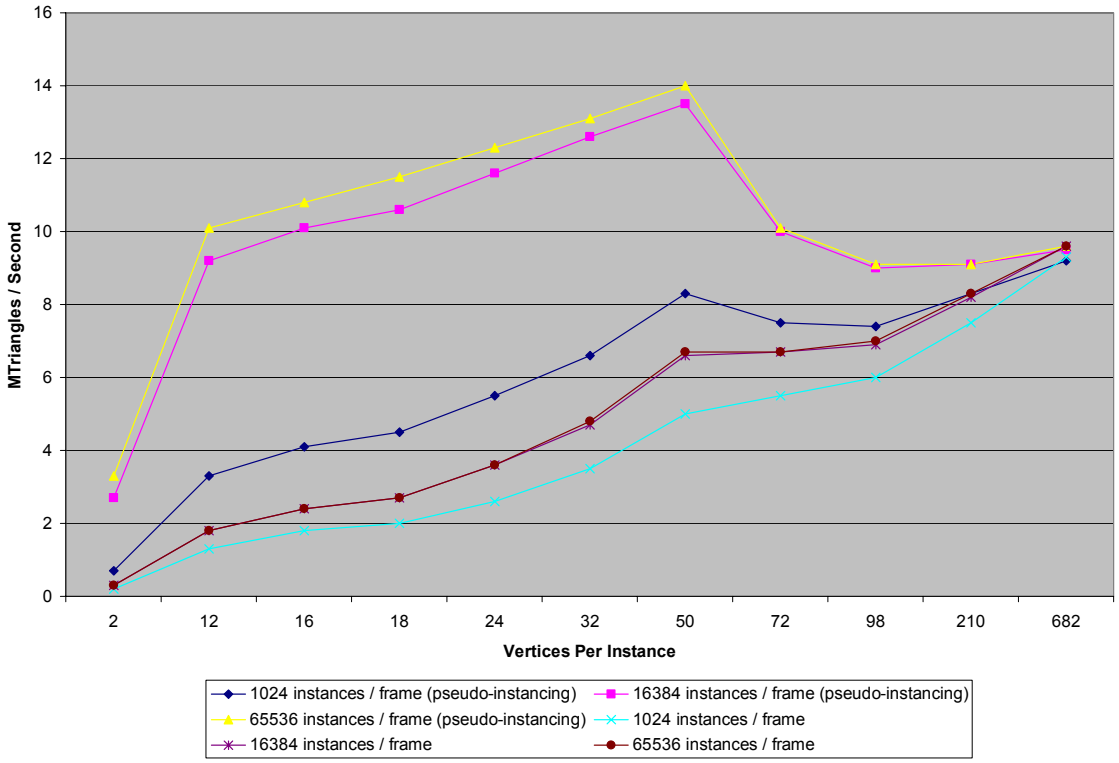
The ratio of instances per second using pseudo-instancing over instances per second without pseudo-instancing is shown below. On this particular setup, the performance difference is substantial for the smaller instance meshes.



The GeForce FX 5200 Ultra can hit over 1.5 million instances per second on tiny instance meshes.



The GeForce FX 5200 enjoys increased triangle rendering rates using pseudo-instancing. The downturn in performance for meshes with more than 50 vertices is probably due to less than perfect vertex cache friendliness in the generated meshes



Conclusion

The pseudo-instancing technique is a simple way to increase performance of rendering large numbers of instanced geometry. The performance advantage is particularly large when the instanced geometry has a smaller number of vertices and on slower CPUs. The technique works on all NVIDIA GPUs that support GLSL vertex shaders (hardware acceleration in GeForce 3 and beyond); there is no requirement for Shader Model 3.0 capable hardware for the technique.

Bibliography

1. Discussion with Mark Kilgard in the hallway.

Notice

Trademarks

Copyright

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