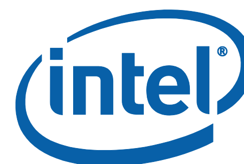


Intel® Processor Graphics DirectX* Developer's Guide

How to maximize graphics performance on Intel® microarchitecture codename Sandy Bridge

Copyright © 2008-2010 Intel Corporation

All Rights Reserved

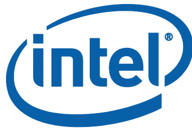
Document Number: 321371-002

Revision: 2.9.6

Contributors: Jeff Freeman, Chris McVay, Chuck DeSylva, Luis Gimenez, Katen Shah, Jeff Frizzell, Ben Sluis, Anthony Bernecky, Raghu Muthyalampalli, Ganeshkumar Doraisamy, Steven Smith, Axel Mamode

World Wide Web: http://www.intel.com

Document Number: 321671-005US



Disclaimer and Legal Information

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

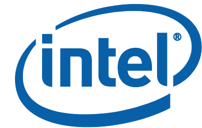
Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm

Software Source Code Disclaimer

Any software source code reprinted in this document is furnished under a software license and may only be used or copied in accordance with the terms of that license:

Intel Sample Source Code License Agreement

This license governs use of the accompanying software. By installing or copying all or any part of the software components in this package, you ("you" or "Licensee") agree to the terms of this agreement. Do not install or copy the software until you have carefully read and agreed to the following terms and conditions. If you do not agree



to the terms of this agreement, promptly return the software to Intel Corporation ("Intel").

1. Definitions:

   A. "Materials" are defined as the software (including the Redistributables and Sample Source as defined herein), documentation, and other materials, including any updates and upgrade thereto, that are provided to you under this Agreement.

   B. "Redistributables" are the files listed in the "redist.txt" file that is included in the Materials or are otherwise clearly identified as redistributable files by Intel.

   C. "Sample Source" is the source code file(s) that: (i) demonstrate(s) certain functions for particular purposes; (ii) are identified as sample source code; and (iii) are provided hereunder in source code form.

   D. "Intel's Licensed Patent Claims" means those claims of Intel's patents that (a) are infringed by the Sample Source or Redistributables, alone and not in combination, in their unmodified form, as furnished by Intel to Licensee and (b) Intel has the right to license.

2. License Grant: Subject to all of the terms and conditions of this Agreement:

   A. Intel grants to you a non-exclusive, non-assignable, copyright license to use the Material for your internal development purposes only.

   B. Intel grants to you a non-exclusive, non-assignable copyright license to reproduce the Sample Source, prepare derivative works of the Sample Source and distribute the Sample Source or any derivative works thereof that you create, as part of the product or application you develop using the Materials.

   C. Intel grants to you a non-exclusive, non-assignable copyright license to distribute the Redistributables, or any portions thereof, as part of the product or application you develop using the Materials.

   D. Intel grants Licensee a non-transferable, non-exclusive, worldwide, non-sublicenseable license under Intel's Licensed Patent Claims to make, use, sell, and import the Sample Source and the Redistributables.

3. Conditions and Limitations:

   A. This license does not grant you any rights to use Intel's name, logo or trademarks.

   B. Title to the Materials and all copies thereof remain with Intel. The Materials are copyrighted and are protected by United States copyright laws. You will not remove any copyright notice from the Materials. You agree to prevent any unauthorized copying of the Materials. Except as expressly provided

herein, Intel does not grant any express or implied right to you under Intel patents, copyrights, trademarks, or trade secret information.

C. You may NOT: (i) use or copy the Materials except as provided in this Agreement; (ii) rent or lease the Materials to any third party; (iii) assign this Agreement or transfer the Materials without the express written consent of Intel; (iv) modify, adapt, or translate the Materials in whole or in part except as provided in this Agreement; (v) reverse engineer, decompile, or disassemble the Materials not provided to you in source code form; or (vii) distribute, sublicense or transfer the source code form of any components of the Materials and derivatives thereof to any third party except as provided in this Agreement.

4. No Warranty:

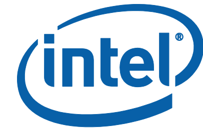
THE MATERIALS ARE PROVIDED "AS IS". INTEL DISCLAIMS ALL EXPRESS OR IMPLIED WARRANTIES WITH RESPECT TO THEM, INCLUDING ANY IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR ANY PARTICULAR PURPOSE.

5. Limitation of Liability: NEITHER INTEL NOR ITS SUPPLIERS SHALL BE LIABLE FOR ANY DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR OTHER LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE, EVEN IF INTEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BECAUSE SOME JURISDICTIONS PROHIBIT THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

6. USER SUBMISSIONS: You agree that any material, information or other communication, including all data, images, sounds, text, and other things embodied therein, you transmit or post to an Intel website or provide to Intel under this Agreement will be considered non-confidential ("Communications"). Intel will have no confidentiality obligations with respect to the Communications. You agree that Intel and its designees will be free to copy, modify, create derivative works, publicly display, disclose, distribute, license and sublicense through multiple tiers of distribution and licensees, incorporate and otherwise use the Communications, including derivative works thereto, for any and all commercial or non-commercial purposes

7. TERMINATION OF THIS LICENSE: This Agreement becomes effective on the date you accept this Agreement and will continue until terminated as provided for in this Agreement. Intel may terminate this license at any time if you are in breach of any of its terms and conditions. Upon termination, you will immediately return to Intel or destroy the Materials and all copies thereof.

8. U.S. GOVERNMENT RESTRICTED RIGHTS: The Materials are provided with "RESTRICTED RIGHTS". Use, duplication or disclosure by the Government is subject to restrictions set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor. Use of the Materials by the Government constitutes acknowledgment of Intel's rights in them.



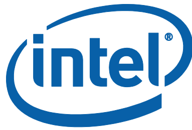
9. APPLICABLE LAWS: Any claim arising under or relating to this Agreement shall be governed by the internal substantive laws of the State of Delaware, without regard to principles of conflict of laws. You may not export the Materials in violation of applicable export laws.

* Other names and brands may be claimed as the property of others.

Copyright (C) 2008 – 2010, Intel Corporation. All rights reserved.

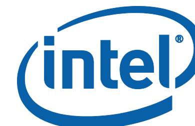
Revision History

Document Number	Revision Number	Description	Revision Date
321671-001US	1.0	Re-drafted for Intel® 4-Series Chipsets.	Sept 2008
321671-001US	1.1	Re-drafted for Intel® 4-Series Chipsets.	Sept 2008
321671-002US	2.6.6	Intel HD Graphics DirectX* Developer's Guide (Sandy Bridge).	March 2009
321671-003US	2.6.7	Intel® HD Graphics DirectX* Developer's Guide (Sandy Bridge).	April 2009
321671-004US	2.7.1	Intel® HD Graphics DirectX* Developer's Guide (Sandy Bridge) featuring Intel® HD Graphics	Feb 2010
321671-005US	2.8.0	Intel® HD Graphics DirectX* Developer's Guide for the Intel® microarchitecture codename Sandy Bridge.	August 2010
321671-006US	2.9.5	Incremental changes and additions related to Intel microarchitecture codename Sandy Bridge	December 2010
	2.9.6	Incremental changes prior to GDC release	February 2011



Contents

Disclaimer and Legal Information 2
Software Source Code Disclaimer 2
Revision History 5
1 About this Document 8
1.1 Intended Audience 8
1.2 Conventions, Symbols, and Terms 8
Table 1 Coding Style and Symbols Used in this Document 8
Table 2 Terms Used in this Document 9
1.3 Further Help Beyond this Guide 9
2 Intel® HD Graphics to Intel® Processor Graphics 10
2.1 Intel® HD Graphics is Becoming Core 10
2.2 What's New in Intel® Processor Graphics 12
2.3 Intel® Processor Graphics Specifications 14
3 Quick Tips: Graphics Performance Tuning 15
3.1.1 Optimizing for the Vertex Cache 15
3.1.2 Other Tips On Vertex/Primitive Processing 16
3.2 Shader Capabilities 17
3.2.1 DirectX* 11 on 10 18
3.2.2 Tips on Shader Capabilities 18
3.3 Texture Sample and Pixel Operations 20
3.3.1 Tips on Texture Sampling / Pixel Operations 21
3.4 Microsoft DirectX* 10 Optional Features 22
3.5 Managing Constants on Microsoft DirectX* 22
3.5.1 Tips on Managing Constants on Microsoft DirectX* 9 23
3.5.2 Tips on Managing Constants on Microsoft DirectX* 10 23
3.6 Advanced DirectX* 9 Capabilities 24
3.6.1 FourCC and other surface and texture formats 24
3.6.2 Notes on supported FourCC texture formats 26
3.6.3 MSAA Under DirectX* 9 26
3.7 Graphics Memory Allocation 26
3.7.1 Checking for Available Memory 27
3.7.2 Tips On Resource Management 27
3.8 Identifying Intel® Processor Graphics and Specifying Graphics Presets 28
3.9 Surviving a Graphics Hardware Switch on the Fly 28
3.9.1 Microsoft DirectX* 9 Algorithm 28
3.9.2 Algorithm for DirectX* 10 29
3.10 Some suggestions, tips & tricks from the field 29
3.10.1 Device Capabilities 29
3.10.2 DirectX* 9 Extensions 30
3.10.3 Revisit Assumptions on Performance 30
3.10.4 Avoid Writing to Unlocked Buffers 31



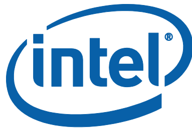
3.10.5 Avoid Tight Polling on Queries 31

4 Appendix: Sample Code for Identifying Intel® Processor Graphics and Specifying Graphics Presets 32

5 Support 37

6 References 38

§



1 About this Document

This document provides development hints and tips to ensure that your customers will have a great experience playing your games and running other interactive 3D graphics applications on Intel® Processor Graphics. This document details software development practices using the latest generation of Intel processor graphics: Intel® Processor Graphics as well as two previous generations of the Intel® Graphics Media Accelerator with a focus on performance analysis using Microsoft DirectX*. Intel tools useful in optimizing graphics applications are introduced in a section detailing performance analysis with the Intel® Graphics Performance Analyzers (Intel® GPA).

Intel® Graphics is split into product generations. The latest one was introduced in 2011 with Intel® microarchitecture codename Sandy Bridge. This family of processors is now on the same silicon as the CPU and it is now called *Intel® Processor Graphics*. In addition to vastly improved performance, Intel Processor Graphics also adds significant new features and functionalities over the previous generation of Intel Graphics called Intel® HD Graphics. Intel Processor Graphics currently represents the most common graphics solution chosen by new PC purchasers. This document is meant to help you include this broad market as a target for your applications and optimize the experience for widest people. By following the tips and tricks in this document, you have the opportunity to make your application shine with the graphics volume market leader.

1.1 Intended Audience

This document is targeted at experienced graphics developers who are familiar with OpenGL*/Microsoft DirectX*, C/C++, multithreading and shader programming, Microsoft Windows* operating systems, and 3D graphics.

1.2 Conventions, Symbols, and Terms

Table 1 Coding Style used in the Document

Source code:

```
for( int i = 0; i < 10; ++i )
{
   cout << i << endl;
}
```

The following terms are used in this document.

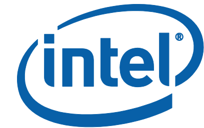


Table 2 Terms Used in this Document

1. **HDG –**Intel® HD Graphics, the generation of graphics from Intel which was characterized by the graphics subsystem being on a separate die from the CPU
2. **Processor Graphics** - The latest generation of graphics from Intel® included in the same processor die of the Intel® microarchitecture codename Sandy Bridge family of processors
3. **UMA** – Unified Memory Architecture - an architecture where the graphics subsystem does not have exclusive dedicated memory and uses the host system's memory (SDRAM)
4. **DVMT** – Dynamic Video Memory Technology – a memory allocation scheme in UMA systems which allocates an exclusive, dynamically resizable chunk of main memory to the graphics (driver)
5. **VF** – Vertex Fetch
6. **VS** – Vertex Shader
7. **PS** – Pixel Shader
8. **GS** – Geometry Shader
9. **EU** – Execution Unit, a vector machine component
10. **I$** - Instruction cache
11. **SO** – Stream Output
12. **HWVP** – Hardware vertex processing - see the following Intel document for more information on this: http://www.**intel**.com/assets/pdf/whitepaper/318888.pdf

1.3 Further Help Beyond this Guide

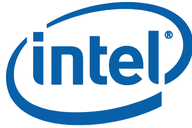
There are several other places you can look for additional information on Intel Processor Graphics, including the following sites:

Intel® HD Graphics (previous generation): http://software.intel.com/en-us/articles/intel-graphics-developers-guides/

Intel® 4 Series Chipsets (the Intel® 4500, X4500, and X4500HD GMAs) Developer's Guide: http://software.intel.com/en-us/articles/intel-graphics-media-accelerator-developers-guide/

Intel® 3 Series Express Chipsets including the Intel® 3000 GMA and Intel® X3000 GMA Developer's Guide: http://software.intel.com/en-us/articles/intel-gma-3000-and-x3000-developers-guide/.

We hope your questions are covered in these resources, including this Guide. We are constantly updating these resources and welcome your comments and suggestions. If you have questions not answered in these resources, or have suggestions on improving the Guide, please get in touch with us at: GameDevInput@intel.com. If you are actively working with Intel already, you can also reach us through your engineering or account management contacts.



2 Intel® HD Graphics to Intel® Processor Graphics

2.1 Intel® HD Graphics is Becoming Core

Several versions of the Intel® Core™ i3, Core™ i5, and Core™ i7 processors have launched in 2011 using Intel® microarchitecture codename Sandy Bridge. These represent the first instantiation of complete platform integration, with Intel® Processor Graphics co-residing on the CPU die.

Two key versions of graphics will be available, Intel® Processor Graphics 2000 and Intel® Processor Graphics 3000, with Intel® Processor Graphics 2000 targeting lower voltage (lower power) applications and Intel® Processor Graphics 3000 a more mainstream set of applications.

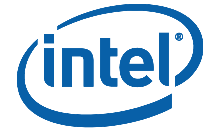


Table 3 Evolution of Intel® Processor Graphics

2009	2010	2011	
GMA Series 4 Chipset	Intel codename Ironlake - 1st Gen CPU Integration	Intel® microarchitecture codename Sandy Bridge - 1st Gen CPU/Graphics single die	
DirectX* 10 SM4, OpenGL* 2.0	DirectX* 10, SM4, OpenGL* 2.1	DirectX* 10.1 SM4.1 , OpenGL* 3.0, DirectX* 11 API on DirectX* 10 hardware	
Mobile / Desktop*: 21 / 32 GFLOPs	Mobile / Desktop: 37 / 43 GFLOPs	Mobile / Desktop: ~105-125 GFLOPs for Processor Graphics 3000 & ~55-65 GFLOPS for Processor Graphics 2000	
Mobile/Desktop 400 - 533 MHz/ 800 MHz	Mobile/Desktop 500-766 MHz/ 533 - 900 MHz	Mobile/Desktop Base: 350-650 MHz/650-850 MHz Turbo: 900-1250 MHz/1100-1350 MHz	
	1.7x 3D performance increase	Intel® Processor Graphics 2000: 6 Execution Units ≥1x with lower voltage requirements.	Intel® Processor Graphics 3000: 12 Execution Units 1.5-2.5x performance increase
* Single precision peak values with MAD instructions.			

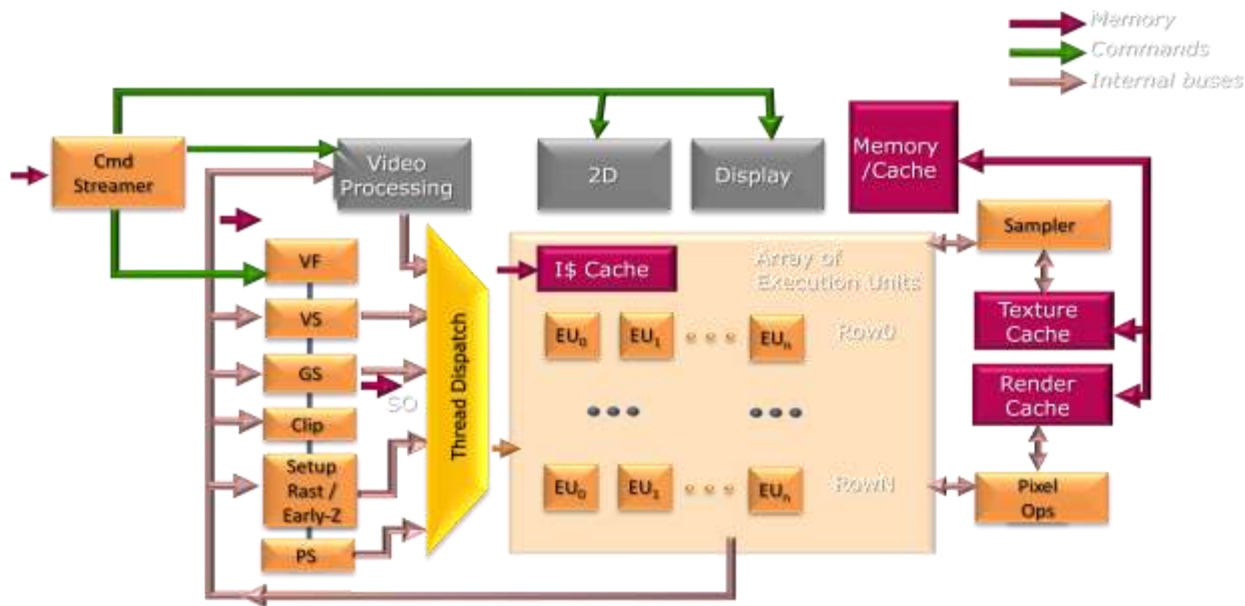


Figure 1 Intel® Processor Graphics Architecture Diagram

Intel® Processor Graphics has been architected to support Microsoft DirectX* 10.1 and take advantage of a generalized unified shader model including support for Shader Model 4.1 and lower. The platform also has support for DirectX* 11 on DirectX* 10 hardware. The graphics core executes vertex, geometry, and pixel shaders on the programmable array of Execution Units (EUs). The EUs have programmable SIMD (Single Instruction, Multiple Data) widths of 4 and 8 element vectors (two 4 element vectors paired) for geometry processing and 8 and 16 single data element vectors for pixel processing). Each EU is capable of executing multiple threads to cover latency. The new generation of Intel® Processor Graphics now integrates transcendental shader instructions into the EU units, rather than a shared math box found in prior generations, resulting in improved processing of instructions such as POW, COS, SIN. Clipping and setup have moved to Fixed Function units, further increasing performance by reducing contention within the EU's. The end result is the fastest Intel® Processor Graphics to date.

2.2 What's New in Intel® Processor Graphics

2.2.1.1 Graphics Features

The latest version of Intel® Processor Graphics includes several performance changes since the previous generation:

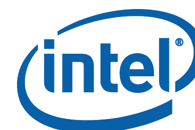
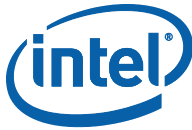


Table 4 Intel® Processor Graphics Feature Specifications

3D Pipeline	Key Improvements in Intel® Processor Graphics
Geometry Processing	• Improved throughput up to 2x better than previous generations • Sharing of the last level cache with the CPU • Increased number of threads for vertex shading • Faster clip, cull, and setup • Improved throughput of geometry shader and stream out • OpenGL* driver now uses hardware geometry processing
Rasterization and Z	• Improved hierarchical Z performance • Improved clear performance • Added OpenGL* MSAA 4X support • Added 2X and 4X MSAA support under DirectX* 9 and DirectX* 10
Computes	• >3X increase in transcendental computations • Overall arithmetic performance improvement in shaders due to math box integration within EU's
Texture and Pixel Processing	• Added support for Gather4, Target Independent Blend Modes, Per Sample Shader Frequency, TextureCubeArray, VS/GS 32 Registers, per the DirectX* 10.1 specification • Improved fill rate for short shaders due to fixed function setup management of barycentric coefficients

2.2.1.2 Intel® Turbo Boost Technology

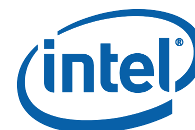
Intel® Processor Graphics utilizes a dynamic frequency on mobile and desktop graphics to automatically increase the clock frequencies of the CPU cores and the graphics cores to boost performance when the workload demands it and also to scale back the frequencies when demand decreases. Intel® microarchitecture codename Sandy Bridge supports higher performance boosts after extended CPU idle periods. In addition to the Intel® Turbo Boost Technology on the CPU, a similar technology has been extended to graphics on both the mobile and desktop platforms. This allows the graphics subsystem to run at higher frequencies when the CPU is not using its nominal *thermal design power* (TDP). In combination, these technologies dynamically manage the CPU and graphics core performances based on workload demands, to allow for better performance when needed and minimize power usage when possible.



2.3 Intel® Processor Graphics Specifications

Table 5 Intel® Processor Graphics Feature Specifications

<table>
<tr><th>Intel Graphics Core</th><th colspan="2">Intel® Processor Graphics</th></tr>
<tr><td>Intel Chipset
(see Error! Reference source not found.)</td><td colspan="2">Intel® microarchitecture codename Sandy Bridge</td></tr>
<tr><td>Graphics Architecture</td><td>Intel® Processor Graphics 2000</td><td>Intel® Processor Graphics 3000</td></tr>
<tr><td>Segment</td><td>Desktop/Mobile</td><td>Desktop/Mobile</td></tr>
<tr><td>Maximum Video Memory</td><td colspan="2">Depends on system memory and operating system.
Windows Vista*/Windows* 7: refer to the table below:
<table><tr><td>System Memory</td><td>1GB</td><td>2GB</td><td>>2GB</td></tr><tr><td>Max Available Video Memory</td><td>256 MB</td><td>783 MB</td><td>1692 MB</td></tr></table></td></tr>
<tr><td>DirectX* Support</td><td colspan="2">10.1, DirectX* 11 on DirectX* 10 hardware,
Compute Shader 4.x,
DirectX* 11 API multi-threaded rendering on DirectX* 10 hardware (asynchronous object creation supported, software support for asynchronous command list in the runtime)</td></tr>
<tr><td>OpenGL* Support</td><td colspan="2">3.0</td></tr>
<tr><td>Shader Model Support</td><td colspan="2">4.1</td></tr>
</table>



3 Quick Tips: Graphics Performance Tuning

3.1.1 Optimizing for the vertex cache

Use `IDirect3DDevice9::DrawIndexedPrimitive` (DirectX* 9) or `ID3D10Device::DrawIndexed` (DirectX* 10) to maximize reuse of the vertex cache.

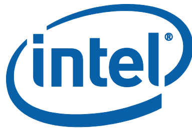
Pre- and post-transform vertex cache sizes can vary significantly, even across different generations of Intel Graphics platforms. To maximize performance of vertex processing, it can help to optimize the ordering of vertices and triangle indices in your vertex and index buffers.

There are two methods you can use to optimize your data for the vertex cache:

The preferred method is to use the `D3DXOptimizeFaces` and `D3DXOptimizeVertices` APIs. These apply a good generalized optimization that will work well for all cache sizes (and across all hardware). Since this optimization works well across all hardware, it can be applied at authoring time (e.g. when exporting mesh data from your content creation pipeline), cutting down level load times.

The code snippet below demonstrates how to use these APIs on mesh data:

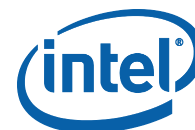
```
void OptimizeMesh( WORD * Indices, // [In/Out] - Index buffer data
                   DWORD NumFaces, // Number of faces in the mesh
                   Vertex * Vertices, // [In/Out] - Vertex buffer data
                   DWORD NumVertices ) // Number of vertices in the mesh
{
    HRESULT hr;

    // Create a "re-map" buffer for the new face ordering, and
    // calculate the new order.
    DWORD *Remap = new DWORD[ NumFaces ];
    hr = D3DXOptimizeFaces( IndicesIn, NumFaces, NumVertices,
                            FALSE, Remap );

    // Make a copy of the old indices, which we'll pull from for the new
    // re-ordered list of indices.
    WORD *OldIndices = new WORD[ NumFaces * 3 ];
    memcpy( OldIndices, IndicesIn, sizeof(WORD) * NumFaces * 3 );
    WORD * NewFaces = Indices;

    // Apply the new mapping.
    for( DWORD i = 0; i < NumFaces; ++i )
    {
        WORD *OldFaceBase = OldIndices + ( Remap[ i ] * 3 );
        NewFaces[0] = OldFaceBase[0];
```




```
        NewFaces[1] = OldFaceBase[1];
        NewFaces[2] = OldFaceBase[2];

        NewFaces += 3;
    }

    delete[] Remap;
    delete[] OldIndices;

    // Create a "re-map" buffer for the new vertex ordering, and
    // calculate the new order.
    DWORD ActualVertexCount = 0;
    Remap = new DWORD[ NumVertices ];
    hr = D3DXOptimizeVertices( Indices, NumFaces, NumVertices,
                 FALSE, Remap );

    // Count how many vertices we actually have. Remap indices of
    // 0xffffffff indicate a vertex that was not referenced by any faces.

    DWORD dwVertexCount = 0;
    for( DWORD i = 0; i < NumVertices; ++i )
    {
        if( aRemap[ i ] == 0xffffffff )
            break;

        ++ActualVertexCount;
    }

        // Copy the vertex data into a temp buffer.
    Vertex *OldVertices = new Vertex[ NumVertices ];
    memcpy( OldVertices, Vertices, sizeof(Vertex) * NumVertices );

    // Perform the remapping
    const DWORD *CurrentRemap = Remap;
    const SVertex *OldVertex = OldVertices;
    for( DWORD i = 0; i < ActualVertexCount; ++i )
    {
        aVertices[ *( CurrentRemap++ ) ] = *( OldVertex++ );
    }

    // We'll also need to re-index our vertices to point to the new
    // vertex locations.
    for( DWORD i = 0; i < NumFaces * 3; i++ )
    {
        Indices[ i ] = ( WORD )aRemap[ Indices[ i ]];
    }

    delete[] Remap;
    delete[] OldVertices;
}
```

3.1.2 Other Tips On Vertex/Primitive Processing

1. Ensure adequate batching of primitives to amortize runtime and driver overhead.
    a. Use instancing to enable better vertex throughput especially for small batch sizes. This also minimizes state changes and Draw calls.



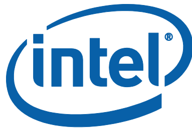
   b. Aim for 2000 or fewer draw-calls per frame (or less than 50,000 draws per second). Above this number the CPU overhead in the driver can become prohibitive.
   c. Minimize render state changes between batches to reduce the number of pipeline flushes.
2. Use static vertex buffers where possible.
3. Use visibility tests to reject objects that fall outside the view frustum to reduce the impact of objects that are not visible.
   a. Set `D3DRS_CLIPPING` to FALSE for objects that do not need clipping.
4. Take advantage of Early-Z rejection.
   a. Render with a Z-only pass (meaning no color buffer writes or pixel shader execution) followed by a normal render pass. This uses the higher performance of Early-Z to reject occluded fragments which reduces compute times and raster operations.
   b. Balance a Z-only pass against the inherent cost of such an additional pass. A Z-only pass increases the number of draw calls (which can impact the CPU usage) as well as the amount of work done up to the Rasterizer. Measure the performance difference between the two approaches to assess the actual benefit.
   c. Avoid writing Z values (depth) in the pixel shader. Writing the depth value will skip the Early-Z hardware optimization algorithm since it potentially changes the visibility of the fragment.

5. Use the Occlusion Query feature of Microsoft DirectX* to reduce overdraws for complex scenes. Render the bounding box or a simplified model – if it returns zero, then the object does not need to be rendered at all.
   a. Allow sufficient time between Occlusion Query tests and verifying the results to avoid serializing the platform. See the Microsoft DirectX* 10 "Draw Predicated" sample included in the Microsoft DirectX* SDK for more information on this.

      See Section 3.10.5 *Avoid Tight Polling on Queries* for more tips on using queries properly.
6. Consider drawing the opaque overlays in the scene such as heads up displays (HUD) first and writing them to the Z buffer. This reduces the screen rendering area leading to considerable performance improvement.

3.2 Shader Capabilities

Intel® Processor Graphics includes support for Microsoft DirectX* Shader Model 4.1 for 10.1 devices and Shader Model 3.0 for DirectX* 9 devices. Intel® microarchitecture codename Sandy Bridge provides significantly improved computational capability and better handling of large and complicated shaders over prior architectures.



3.2.1 DirectX* 11 on 10

In addition to DirectX* 10.1 hardware support, Intel® Processor Graphics includes partial DirectX11 API support. This allows developers to use the DirectX* 11 API on DirectX* 10 (or 10.1) level hardware. There are two DirectX11-specific features exposed through this paradigm, specifically:

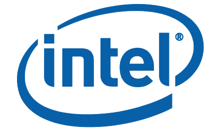
1. Multithreading: The D3D11 API provides means to record command lists on multiple threads (multithreaded rendering) as well as object creation on multiple threads. The Processor Graphics driver currently supports driver-level concurrent object creation.
2. Compute Shader 4 (CS4): A limited subset of DirectX11-level compute shader (CS5) a number of restrictions, specifically:
   - Maximum threads is limited to 768.
   - Z dimension of numthreads/dispatch limited to 1.
   - Only one unordered access view bound per shader (and only RWStructuredBuffers and RWByteAddressBuffers are bindable).
   - Threads are limited to writing to only their own region of groupshared memory (although they can thread from any location).
   - SV_GroupIndex must be used when accessing groupshared memory for writing.
   - Groupshared memory limited to 16KB per group.
   - Atomics and double-precision are not available.

Other DirectX* 11 features (Tessellation, CS5, extended texture formats such as BC6/7) are not supported.

Utilize the DirectX* `ID3D11Device::CheckFormatSupport()` API for individual format support of the D3D11_FORMAT_SUPPORT enumeration.

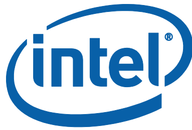
3.2.2 Tips on Shader Capabilities

1. Reduce texture sampling intensive operations when possible. The following common shader effects typically affect performance and should be tested for performance and optimization. Pay special attention to full screen post processing effects including per-pixel and multiple pass techniques when evaluating graphics related performance bottlenecks.
   a. Glow/Bloom
   b. Motion blur
   c. Depth of field
   d. HDR/tone mapping
   e. Heat distortion
   f. Atmospheric effects
   g. Dynamic Ambient Occlusion
   h. High quality soft shadows
   i. Parallax occlusion mapping with a wide radius
2. Balance texture samples and shader complexity.



   a. Recommend greater than 4:1 ratio of ALU:Sample for better latency coverage. A larger ratio may be better for floating point textures, higher order filtering, and 3D textures.

3. Space your texture sampling calls away from where it is used in pixel shaders when possible and practical to maximize EU utilization.
4. Generically speaking where a choice is available between using programmable shading and fixed function pipeline, programmable shading gives better performance.
   a. In particular*, use sh*ader-based fog instead of fixed function fog on DirectX* 9. Fixed Function fog has been deprecated on SM 3.0.
   b. One notable exception to this rule is the use of the *texkill* instruction. Consider using alpha test to achieve the same result if possible, as it performs better on this platform.
5. Use flow control wisely - it can be expensive. In some cases, it might be advantageous to split a single shader into multiple ones, to avoid flow control statements.
   a. The pixel shader operates on up to 16 pixels in parallel. This means the benefits of dynamic flow control will depend on the likelihood of the number of pixels taking the same branch. If any 2 of those pixels take different paths, all pixels will execute both branches of the control flow.
   b. Dynamic flow control can provide significant benefits by skipping a large number of computations. Ensure that this is used where a large number of instructions can be skipped.
6. Shader Model 3.0 supports dynamic flow instructions such as *if bool, break,* and *break_comp.* It also supports predication registers. Often the dynamic flow control instructions can be eliminated by using predication registers in their place. When possible, use predication instructions over dynamic flow control for shorter branching instruction sequences. Optimize your shader performance by adequate use of your processor graphics - mask alpha if you are not using it.
7. Minimize the usage of geometry shaders, especially in cases where they result in geometry amplification (more geometry output from the GS than came in). If in doubt, examine your application's performance with and without geometry shaders to determine if they are a significant performance issue.
8. In general, minimize use of StreamOut and DrawAuto() for optimal performance, since they can incur a significant bandwidth penalty with their usage of system memory.



3.3 Texture Sample and Pixel Operations

Table 6 Intel® Processor Graphics Texture Sampling and Pixel Specifications

CPU Product	**See** Error! Reference source not found.
Graphics Architecture	**Desktop and Mobile (Processor Graphics 2000 & 3000)**
Format Support	16/32-bit fixed point 16/32-bit floating point operations
Max # of Samplers	Up to 16
Vertex Textures	Yes
Max 2D/3D/Cube Textures	8Kx8K/2Kx2Kx2K/8K
Filtering Type Support	BLF, TLF and Dynamic AF with up to 16 degrees of anisotropic filtering + DirectX* 10.1
Texture Compression	DirectX* 9: DXT1/3/5, ATI1, ATI2; DirectX* 10: BCx
Multi-Sample Render	MSAA 4X
Multi-Target Render	8
Alpha-Blend FP formats	FP16 and FP32 formats are supported. For a complete list, do a caps check on DirectX* 9 and on DirectX* 10, utilize the DirectX* CheckFormatSupport() call as format support may be added in future driver versions.

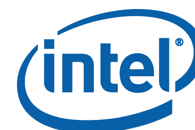


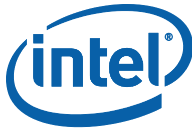
Table 7 Intel® Processor Graphics Sampler Filtering Specifications

Product	**See** Error! Reference source not found.
32-bit Texels (per clock cycle) **e.g. RGBA UNORM8, RG FP16, R FP32**	
Point/Bilinear	1X
Trilinear	1X
Anisotropic (n samples)	0.5X/n
64-bit Texels (per clock cycle) **e.g. RG FP32, RGBA FP16, RGBA UINT16**	
Point/Bilinear	1X
Trilinear	0.5X
Anisotropic (n samples)	0.25X/n
128-bit Texels (per clock cycle) **e.g. RGBA FP32, RGBA UINT32**	
Point	0.25X
Bilinear	0.25X
Trilinear	0.125X

All sampler filtering types are supported, including dynamic anisotropic filtering.

3.3.1 Tips on Texture Sampling / Pixel Operations

1. Use compressed textures and mip-maps and minimize the use of large textures. Even though the architecture supports up to 8K×8K textures, for optimal performance it is best to use smaller textures.
2. Minimize the use of Anisotropic Filtering, and both Trilinear and Anisotropic filtering for floating point textures. With floating point texture formats, the performance of bilinear and trilinear filtering are not equivalent.



Examine the scene to determine where you can make performance/quality trade-offs with texture filtering. Prefer bilinear filtering where there is little visual difference.

3. Generically speaking, the more compact the texture format being used, the better the performance. DXT compressed formats are best. Minimize the use of 32-bit floating point textures, since they carry a heavy bandwidth penalty and fill the texture caches faster.
4. Use as few render targets as possible, ideally keeping it to less than four. More render targets requires more bandwidth. If in doubt, analyze your performance using a tool such as Intel® GPA to determine if you are fill bound.
5. Minimize the number of Clear calls. Clear surfaces, Color and Z/Stencil buffer at the same time when required.
6. Avoid stencil shadows as they are fill intensive.

3.4 Microsoft DirectX* 10 Optional Features

D3D10 specifies optional features that can be checked for support in the code through API functions like *CheckFormatSupport(...)*, *CheckMultipleQualityLevels(...)*, *CheckFeatureSupport(...)*

The current platform supports more of those optional features than the previous ones and even more features will likely be supported in future. So it is better to explicitly test for such features using those APIs rather than relying on vendor and device ID's for the platform.

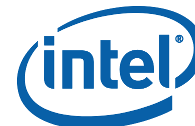
For example, DirectX* 10 specifies a large number of resource types and data formats that are optional. Utilize the DirectX* 10 CheckFormatSupport(...) call to determine which ones are supported. Also, utilize the DirectX* 10 CheckFormatSupport(...) call for UNORM and SNORM blending support

The following optional features are supported at the time of review of this Guide:

1. MSAA 2X and 4X on DirectX* 10.
2. 32-bit floating point blending

3.5 Managing Constants on Microsoft DirectX*

Constants are external variables passed as parameters to the shaders; their values remain “constant” during each invocation of the shader program. Despite their name, constants are one of the most frequently changing values in a Microsoft DirectX*



application. A shader program can initialize a constant value statically to a value in the shader file or at runtime through the application.

Many of the recommendations described here are standard in the industry. They are very much applicable to Intel processor graphics and the recommendations attempt to detail them in a cohesive manner.

In addition to these points it is worth noting that care should be taken when porting from Microsoft DirectX* 9 to Microsoft DirectX* 10 to maintain performance. For more information on this topic, see the Intel publication "DirectX* Constants Optimizations For Intel® Processor graphics" [2] available on the Intel Software Network.

3.5.1 Tips on Managing Constants on Microsoft DirectX* 9

1. The driver optimizes access to the most frequently used constants. Use less than 32 (FLOAT4) constants per shader to achieve the best performance gain from this feature. Limit the use of dynamic indexed constants (C[ax], C[r]) as these cannot be optimized by the driver and will result in high latency in shaders (dynamic indexed constants are normally found in vertex shaders).
2. Prefer local constants over global constants - the former are better for performance.
3. Immediate constants provide better performance than dynamic indexed constants. In dynamic indexed constants the driver cannot determine a previous index value and needs to create a full size constant buffer space in memory, instead of using the hardware constant buffer.

3.5.2 Tips on Managing Constants on Microsoft DirectX* 10

1. As previously detailed for DirectX* 9 above, the driver optimizes access to the most frequently used constants. The same advice applies for DirectX* 10: use less than 32 constants per shader to achieve the best performance gain from this feature, and limit the use of dynamic indexed constants (C[ax], C[r]) as these cannot be optimized by the driver.
2. For better performance prefer multiple, smaller constant buffers. The constant buffers need to be loaded to the graphics subsystem ahead of the shader execution, and the entire buffer needs to reloaded every time its contents change. Larger the size of the buffer, longer it takes to load the buffer to the graphics subsystem, causing significant performance impacts. If multiple buffers are combined into a single larger buffer, every time any of the contents changes, the entire large buffer will have to be loaded again, degrading the performance. And, because of the performance impact that reloading a constant buffer can have, where possible, if multiple shaders share the same buffer that could help the performance. Whether that performance gain is actually realized depends on whether the shared buffer is still resident when the second shader sharing the buffer is executed. But sharing the buffers between shaders can help but does not hurt the performance. For optimal constant buffer management, smaller packed constant buffers grouped by frequency of update and access pattern are

ideal for higher performance. As an example: organize Per Frame/ Per Pass/ Per Instance constant buffers first which tend to be smaller in size and have a low update rate followed by Per Draw/Per Material constant buffers which may also be small but have a higher update rate. Put large constant buffers like skinning constants last.

3. If there are constants that are unused by most of the shaders move those to the bottom of the constant definition list so that you can bind a smaller buffer to those shaders.
4. Break up constant buffers based on features that are optional in games (e.g. shadows, post-processing effects, etc.). Very likely, due to performance constraints for integrated platforms, some of these features are either going to be disabled or run with a lower setting. So it would be beneficial to break up the constants into separate buffers and then selectively disable the updates to these constant buffers based on the settings selected by the user.
5. When using indexed constant buffers, it is recommended to keep the buffer size tailored to actual needs. For example, if the shader iterates over five elements only, declare a 5-element constant buffer for this shader rather than a general purpose 50-element constant buffer shared among shaders. This allows the driver to optimize placement so that it incurs a low latency path.

3.6 Advanced DirectX* 9 Capabilities

Several advanced features beyond those required by the DirectX* 9 specifications are supported by the Intel Graphics Platforms. This section provides the details on some of them.

The feature list is current for Intel® microarchitecture codename Sandy Bridge, when this Guide is being prepared and is likely to be extended over time. Further, support for the features vary between different Intel platforms. We strongly advise the developers to first confirm in their code that the feature of interest is supported and provide alternative execution paths in the code where features are not supported.

These are capabilities not directly exposed by DirectX* 9 interfaces. Developers will have to use indirect methods to check for their availability. Code segments showing how to check for such features are given in this section.

3.6.1 FourCC and other surface and texture formats

Intel Graphics platforms support multiple surface-formats beyond those required by the DirectX* 9 specifications. These are listed in Table 8.

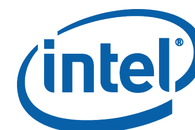


Table 8 Additional DirectX* 9 texture and surface formats supported on Intel platforms

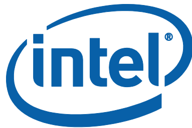
Format	Resource Type	Usage	Description
INTZ	Texture	Depth/Stencil	For reading depth buffer as a texture
DF16	Texture	Depth/Stencil	For reading depth buffer as a texture
RESZ	Surface	Render Target	For converting a multisampled depth buffer into a depth stencil texture.
ATI1N	Texture	Texture	Single Channel Texture Compression. Functionally equivalent (but not identical) to DirectX* 10 BC4 format.
ATI2N	Texture	Texture	Two-channel Texture Compression. Functionally equivalent (but not identical) to DirectX* 10 BC5 format.
NULL	Texture	Render Target	Render Target with no memory allocated. Useful as a dummy render target to render exclusively to a depth buffer.
ATOC	Surface	Render Target	Alpha to coverage multisampling.

The following code segment shows a sample outline and works for most FourCC

```
... ... ...
// format of the display mode into which the adapter will be placed
D3DFORMAT AdapterFmt = D3DFMT_X8R8G8B8;
// check for FourCC formats like INTZ
if (pD3D->CheckDeviceFormat( D3DADAPTER_DEFAULT,
                             D3DDEVTYPE_HAL,
                             AdapterFmt,
                             D3DUSAGE_DEPTHSTENCIL,
                                D3DRTYPE_TEXTURE,
                             (D3DFORMAT)(MAKEFOURCC('I','N','T','Z')
                            ) == D3D_OK )
{
    return  true;
}
... ... ...
```

formats.

See the code accompanying this Guide for an example of how to test for more formats.



3.6.2 Notes on supported FourCC texture formats

1. INTZ is a depth texture format meant for accessing 24-bit depth buffer and 8-bit stencil buffer. In addition to depth operations this allows for using it for stencil operations as well.

This surface cannot be used as a texture when it is being used for depth buffering, *unless the depth writes are disabled.*

2. DF24 and DF16 formats are meant for use in Percentage-Closer Filtering (PCF) used in shadow mapping applications that use PCF. DF16 has better performance.

3. Intel® microarchitecture codename Sandy Bridge supports MSAA under DirectX* 9. RESZ provides the ability to use multisampling in depth buffer and then copy the result as a single value into a depth-stencil buffer. This process is often referred to as "resolving" a multi-sample depth-stencil buffer into a single-sample depth-stencil buffer.

4. Some ISV's use FourCC formats at times known as ATI1N which allows for the compression of single channel textures. The latest Intel platforms support this for the benefit of those ISV's.

5. 3DC format is also known in the industry as ATI2N format. This format is also supported in the latest Intel platforms.

6. A *NULL Render Target* is a dummy render target format which acts like a valid render target with a crucial difference that the driver will not allocate any memory for it. DX9 requires that a color render target be used for *all* rendering operations. Since a color render target is often not used with depth-only render targets, using a NULL render target in such cases can avoid memory allocation for a color render target will not be used.

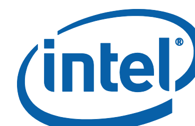
7. ATOC, short for "Alpha To Coverage", is meant for interpreting the alpha channel value for approximating the multichannel pixel coverage. The texture itself has no practical use; but you use it as an argument in the *SetRenderState(...)* function, to convert the incoming alpha value output by the pixel shader into multichannel pixel coverage value.

3.6.3 MSAA Under DirectX* 9

2x and 4X MSAA are supported in DX9 on the latest Intel platforms. In general MSAA requires the graphics engine to do more work and hence impacts the performance to varying degrees. The developers should weigh in the tradeoffs ahead of using MSAA in their titles.

3.7 Graphics Memory Allocation

Processor graphics will continue to use the Unified Memory Architecture (UMA) and Dynamic Video Memory Technology (DVMT) as noted in the chart below. As with past processor graphics solutions, UMA specifies that memory resources can be used for video memory when needed. DVMT is an enhancement of the UMA concept, wherein



system memory is allocated for balanced graphics and system performance. DVMT dynamically responds to system requirements and application demands, by allocating the proper amount of display, texturing, and buffer. The operating system views the Intel graphics driver as an application, which uses system memory to request allocation of additional memory for 3D applications, and returns the memory to the operating system when no longer required.

Table 9 Intel® Processor Graphics Memory Specifications

CPU Product	**See** Error! Reference source not found.	
Segment	**Processor Graphics 2000**	**Processor Graphics 3000**
Memory BW (GB/s)	17.1 - 21.3	17.1-25.6
UMA Capability	2x DDR3 up to 1333	2x DDR3 up to 1600
Max DVMT (XP) 1 or 2GB System Memory	Limited to 1GB max for all system memory configurations	
Max DVMT (Windows Vista*/Windows* 7) x86/x64: System Memory	The memory is managed by the operating system and the driver.	
Memory Interface	64 bits	

3.7.1 Checking for Available Memory

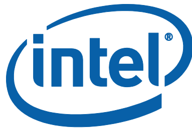
Graphics applications often check for the amount of available free video memory early in execution. Developers typically want to know the amount of memory that the graphics device can access at full performance.

As a result of the dynamic allocation of graphics memory performed by the Intel® Processor Graphics devices (based on application requests), memory checks that only request the amount of 'local' or 'dedicated' graphics memory available do not supply a number that is appropriate for the Intel® Processor Graphics devices.

The Microsoft DirectX* SDK (June 2010) includes the VideoMemory sample code which demonstrates 5 commonly used methods to detect the total amount of video memory. Of these tests, GetVideoMemoryViaWMI is recommended. All other methods either return the local/dedicated graphics memory and consequently will report incorrect results on Intel® Processor Graphics, or will report the sum of the dedicated memory and the shared memory, something that is typically not suitable for discrete graphics devices. For more information, see the sample code: http://msdn.microsoft.com/en-us/library/ee419018(v=VS.85).aspx

3.7.2 Tips On Resource Management

1. Allocate surfaces in priority order. The render surfaces that will be used most frequently should be allocated first. On Microsoft DirectX* 10, memory is taken care of for you by the OS. On Microsoft DirectX* 9:



   a. Use D3DPOOL_DEFAULT for lockable memory (dynamic vertex/index buffers).
   b. Use D3DPOOL_MANAGED for non-lockable memory (textures, back buffers, etc).
2. On D3D10, use of the Copy…() methods are preferred over calling the Update…() operations. Partial or sub-resource copies should be used sparingly. For example when updating all or most of the LODs of a resource use CopyResource() rather than multiple CopySubResource().

3.8 Identifying Intel® Processor Graphics and Specifying Graphics Presets

Games often specify a range of graphics capabilities and presets to identify with a Low, Medium, and High fidelity level. Please refer to the Appendix for sample code that demonstrates how to identify Intel® Processor Graphics versions and set fidelity levels for older generations (Low) to the most recent (Medium) based on the requested D3D*_FEATURE_LEVEL.

Note that on Windows* 7, multiple graphics adapters are supported so care should be taken in determining which adapter will be used for rendering.

3.9 Surviving a Graphics Hardware Switch on the Fly

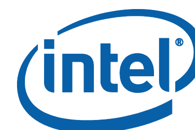
Intel in combination with third party graphics vendors jointly developed a switchable graphics solution that allows end users to switch on-the-fly, between two heterogeneous graphics hardware systems without a reboot. This functionality can incorporate the energy efficiency of Intel processor graphics with the 3D performance of discrete graphics hardware in a single notebook solution. This technology is applicable to about 30 million discrete graphics hardware notebooks purchased annually. Currently most applications running on PC platforms with heterogeneous graphics hardware do not survive when switched between the two at run-time as they do not re-query underlying graphics capability when the active adapter changes.

Keys to handling graphics hardware switches:

- New applications should be aware of multiple graphics hardware configurations and handle the D3DERR_DEVICELOST and WM_DISPLAYCHANGE messages properly.
- Legacy applications where possible should develop and distribute patches for old games to handle the D3DERR_DEVICELOST and WM_DISPLAYCHANGE messages.

3.9.1 Microsoft DirectX* 9 Algorithm

Microsoft DirectX* 9 applications should follow the below procedure to query GFX adapter's capabilities (re-create DX object/device) on D3DERR_DEVICELOST:



1. Manually save the current context including state and draw information in the application.
2. Query if the graphics adapter has changed, using the Windows* API's EnumDisplaySettings() or EnumDisplayDevices().
3. If the adapter has changed, then:
    a. Recreate a Microsoft DirectX* device.
    b. Restore the context.
    c. Continue rendering from last scene rendered before the `D3DERR_DEVICELOST` event occurred.

3.9.2 Algorithm for DirectX* 10

By design, DirectX* 10 does not have the concept of device lost until the next Present, and the developer is guaranteed the API will keep working until then. The changes in Microsoft DirectX* 10 applications are:

1. Check for `WM_DISPLAYCHANGE` windows message in the message handler.
2. Query if the graphics adapter has changed using the Windows* API's EnumDisplaySettings() or EnumDisplayDevices().
3. If yes, then save off the current context including state and draw information in the application and then:
    a. Recreate the Microsoft DirectX* device.
    b. Restore the context.
    c. Continue rendering from the last scene rendered before the `WM_DISPLAYCHANGE` message occurred.

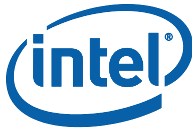
3.10 Some suggestions, tips & tricks from the field

The items in this section are based on the observations of Intel engineers with code from developers with different levels of experiences. These are collected here as a checklist for reference for developers. Some of the items are generic to all graphics platforms.

3.10.1 Device Capabilities

Intel® microarchitecture codename Sandy Bridge supports DirectX* functionality up to and including full D3D10.1 support.

If you encounter a Direct3D feature that does not work on the latest drivers on this device, please contact your Intel Account Manager. Intel will investigate these issues for future drivers and should be able to suggest workarounds.



3.10.2 DirectX* 9 Extensions

Several hardware vendors support their own extensions to the DirectX* 9 specifications through specific texture formats and render paths that are not part of Microsoft's official DirectX* 9 specifications.

To ensure maximum compatibility with these extensions, Intel now supports many of those extensions. A list of those, current as of the release of this Guide, is given in section 3.6 Advanced DirectX* 9 Capabilities (page 24).

If there are additional extensions that you believe are useful, (or have OpenGL* extensions that you require) please let your Intel Account Manager know.

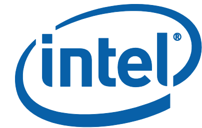
3.10.3 Revisit Assumptions on Performance

Intel® Processor Graphics is continually increasing functionality and performance. As well as the addition of full D3D10.1 support and increased capabilities previously mentioned, the performance profile has been improved significantly for this platform. As such, it is advised to remove previous restrictions and scale your title to match this increased performance and functionality.

Should you see unexpected issues, please follow these steps:

1. Verify you are running the latest drivers. This platform is evolving, so there will be frequent driver updates. Check for updates at http://www.intel.com and if you are an Intel software partner, at http://platformsw.intel.com.

   Intel graphics drivers follow a naming convention that use four field numbers - for example, 15.21.8.2279. The number in the last field - 2279 in the example - increases sequentially with each driver release. So a driver with a higher number there is more recent. You always want to have the most recent driver installed on your system.
2. If you suspect that it is a functionality bug, try to recreate the bug with the reference rasterizer.
3. Look for easily fixed hotspots using the Intel® Graphics Performance Analyzers (Intel® GPA). Talk to your Intel Account Manager if you do not already have access to this tool.
4. If the above steps do not resolve the issue, or you need additional help determining the root cause, please contact us - see the section Further Help Beyond this Guide in this document. Intel engineers are available to help enable your title to run effectively on our platforms. This enabling can potentially include (but is not limited to):
   a. Training on using Intel GPA to get the best results for your title.
   b. In-depth performance analysis of your code running on our platform, with specific feedback on optimization opportunities.
   c. Championing the resolution of your issues within Intel, such as helping resolve or workaround driver issues, addressing tool issues, etc.



3.10.4 Avoid Writing to Unlocked Buffers

We have found multiple cases of corruption which were caused by the writes to unlocked buffers. Typically the title first locked the buffer, then unlocked the buffer and then attempted to write to the buffer that is already unlocked. This *sometimes* works because the system might have not moved the buffer and has not dispatched the rendering commands to the graphics hardware. This is inconsistent with Microsoft DirectX* specifications and it is not safe for applications to expect it to work consistently.

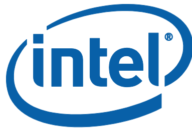
Avoid writing to buffers that have not been locked. Some drivers on other hardware platforms are more forgiving of this approach, and may handle it gracefully. The Intel driver makes optimizations based on the specified behavior of the API, so the behavior of this platform will be undefined in this case.

Always test your application with the DirectX* debug runtime enabled. The debug runtime will catch scenarios like this and help you avoid problems should driver behavior change in the future.

3.10.5 Avoid Tight Polling on Queries

Tight polling on event/occlusion queries degrades performance by interfering with the graphics subsystem's turbo mode operation.

Allow for queries to work asynchronously and avoid waiting on the query response immediately after sending the query. Issue queries as early as possible in the frame, and issue their dependent draws as late as possible. If the query result is not available at the time the draw is to be submitted just issue the draw. Any additional delay at this point will cause a pipeline stall.



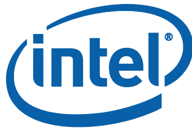
4 Appendix: Sample Code for Identifying Intel® Processor Graphics and Specifying Graphics Presets

The following sample code and configuration file demonstrates how to identify Intel® Processor Graphics versions and set fidelity levels for older generations (Low) to the most recent (Medium) based on the requested D3D*_FEATURE_LEVEL.

Example Source Code:

```
// Copyright 2010 Intel Corporation
// All Rights Reserved
//
// Permission is granted to use, copy, distribute and prepare derivative works of this
// software for any purpose and without fee, provided, that the above copyright notice
// and this statement appear in all copies.  Intel makes no representations about the
// suitability of this software for any purpose.  THIS SOFTWARE IS PROVIDED ""AS IS.""
// INTEL SPECIFICALLY DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, AND ALL LIABILITY,
// INCLUDING CONSEQUENTIAL AND OTHER INDIRECT DAMAGES, FOR THE USE OF THIS SOFTWARE,
// INCLUDING LIABILITY FOR INFRINGEMENT OF ANY PROPRIETARY RIGHTS, AND INCLUDING THE
// WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.  Intel does not
// assume any responsibility for any errors which may appear in this software nor any
// responsibility to update it.


//
// DeviceId.cpp : Implements the Graphics Device detection and graphics settings
//                configuration functions.
//

#include <stdio.h>
#include <tchar.h>

#include <D3DCommon.h>
#include <DXGI.h>
#include <D3D9.h>

static const int FIRST_GFX_ADAPTER = 0;

// Define settings to reflect Fidelity abstraction levels you need
typedef enum {
               NotCompatible,   // Found Graphics is not compatible with the app
               Low,
               Medium,
               High,
               Undefined        // No predefined setting found in cfg file.
                                // Use a default level for unknown video cards.
             }
             PresetLevel;

/*****************************************************************************************
 * getGraphicsDeviceID
 *
 *     Function to get the primary graphics device's Vendor ID and Device ID, either
 *     through the new DXGI interface or through the older D3D9 interfaces.
 *
 *****************************************************************************************/
```

```
bool getGraphicsDeviceID( unsigned int& VendorId, unsigned int& DeviceId )
{
    bool retVal = false;
    bool bHasWDDMDriver = false;

    HMODULE hD3D9 = LoadLibrary( L"d3d9.dll" );
    if ( hD3D9 == NULL )
        return false;

    /*
     * Try to create IDirect3D9Ex interface (also known as a DX9L interface).
     * This interface can only be created if the driver is a WDDM driver.
     */

    // Define a function pointer to the Direct3DCreate9Ex function.
    typedef HRESULT (WINAPI *LPDIRECT3DCREATE9EX)( UINT, void **);

    // Obtain the address of the Direct3DCreate9Ex function.
    LPDIRECT3DCREATE9EX pD3D9Create9Ex = NULL;
    pD3D9Create9Ex = (LPDIRECT3DCREATE9EX) GetProcAddress( hD3D9, "Direct3DCreate9Ex" );

    bHasWDDMDriver = (pD3D9Create9Ex != NULL);

    if( bHasWDDMDriver )
    {
        // Has WDDM Driver (Vista, and later)
        HMODULE hDXGI = NULL;

        hDXGI = LoadLibrary( L"dxgi.dll" );

        // DXGI libs should really be present when WDDM driver present.
        if ( hDXGI )
        {
            // Define a function pointer to the CreateDXGIFactory1 function.
            typedef HRESULT (WINAPI *LPCREATEDXGIFACTORY)(REFIID riid, void **ppFactory);

            // Obtain the address of the CreateDXGIFactory1 function.
            LPCREATEDXGIFACTORY pCreateDXGIFactory = NULL;
            pCreateDXGIFactory = (LPCREATEDXGIFACTORY) GetProcAddress( hDXGI,
                                                                  "CreateDXGIFactory" );

            if ( pCreateDXGIFactory )
            {
                // Got the function hook from the DLL
                // Create an IDXGIFactory object.
                IDXGIFactory * pFactory;
                if ( SUCCEEDED( (*pCreateDXGIFactory)(__uuidof(IDXGIFactory),
                                                      (void**)(&pFactory) ) ) )
                {
                    // Enumerate adapters.
                    // Code here only gets the info for the first adapter.
                    // If secondary or multiple Gfx adapters will be used,
                    // the code needs to be modified to accomodate that.
                    IDXGIAdapter *pAdapter;
                    if ( SUCCEEDED( pFactory->EnumAdapters( FIRST_GFX_ADAPTER,
                                                            &pAdapter ) ) )
                    {
                        DXGI_ADAPTER_DESC adapterDesc;
                        pAdapter->GetDesc( &adapterDesc );

                        // Extract Vendor and Device ID information from adapter descriptor
                        VendorId = adapterDesc.VendorId;
                        DeviceId = adapterDesc.DeviceId;

                        pAdapter->Release();

                        retVal = true;
                    }
                }
            }

            FreeLibrary( hDXGI );
        }
    }
    else
    {
```

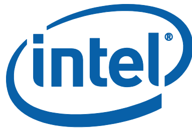



```
        /*
         * Does NOT have WDDM Driver. We must be on XP.
         * Let's try using the Direct3DCreate9 function (instead of DXGI)
         */

        // Define a function pointer to the Direct3DCreate9 function.
        typedef IDirect3D9* (WINAPI *LPDIRECT3DCREATE9)( UINT );

        // Obtain the address of the Direct3DCreate9 function.
        LPDIRECT3DCREATE9 pD3D9Create9 = NULL;
        pD3D9Create9 = (LPDIRECT3DCREATE9) GetProcAddress( hD3D9, "Direct3DCreate9" );
        if( pD3D9Create9 )
        {
            // Initialize the D3D9 interface
            LPDIRECT3D9 pD3D = NULL;
            if ( (pD3D = (*pD3D9Create9)(D3D_SDK_VERSION)) != NULL )
            {
                D3DADAPTER_IDENTIFIER9    adapterDesc;
                // Enumerate adapters. Code here only gets the info for the first adapter.
                if ( pD3D->GetAdapterIdentifier( FIRST_GFX_ADAPTER, 0,
                                                 &adapterDesc ) == D3D_OK )
                {
                    VendorId = adapterDesc.VendorId;
                    DeviceId = adapterDesc.DeviceId;

                    retVal = true;
                }

                pD3D->Release();
            }
        }
    }

    FreeLibrary( hD3D9 );
    return retVal;
}



/******************************************************************************************
 * getDefaultFidelityPresets
 *
 *     Function to find / set the default fidelity preset level, based on the type
 *     of graphics adapter present.
 *
 *     The guidelines for graphics preset levels for Intel devices is a generic one
 *     based on our observations with various contemporary games. You would have to
 *     change it if your game already plays well on the older hardware even at high
 *     settings.
 *
 ******************************************************************************************/

PresetLevel getDefaultFidelityPresets( void )
{
    unsigned int VendorId, DeviceId;

    PresetLevel presets = Undefined;

    if ( !getGraphicsDeviceID ( VendorId, DeviceId ) )
    {
        return NotCompatible;
    }

    FILE *fp = NULL;
    switch( VendorId )
    {
        case 0x8086:
            fopen_s ( &fp, "IntelGfx.cfg", "r" );
            break;

           // Add cases to handle other graphics vendors…

        default:
            break;
    }
```

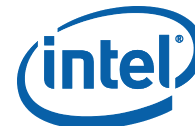
```
    if( fp )
    {
        char line[100];
        char *context = NULL;

        char *szVendorId = NULL;
        char *szDeviceId = NULL;
        char *szPresetLevel = NULL;

        while ( fgets ( line, _countof(line), fp ) )   // read one line at a time till EOF
        {
             // Parse and remove the comment part of any line
            int i; for( i = 0; line[i] && line[i]!=';'; ++i ); line[i] = '\0';

            // Try to extract VendorId, DeviceId and recommended Default Preset Level
            szVendorId    = strtok_s( line, ",\n", &context );
            szDeviceId    = strtok_s( NULL, ",\n", &context );
            szPresetLevel = strtok_s( NULL, ",\n", &context );

            if( ( szVendorId == NULL ) ||
                ( szDeviceId == NULL ) ||
                ( szPresetLevel == NULL ) )
            {
                continue;  // blank or improper line in cfg file - skip to next line
            }

            unsigned int vId, dId;
            sscanf_s( szVendorId, "%x", &vId );
            sscanf_s( szDeviceId, "%x", &dId );

           // If current graphics device is found in the cfg file, use the
           // pre-configured default Graphics Presets setting.
            if( ( vId == VendorId ) && ( dId == DeviceId ) )
            {
                // Found the device
                char s[10];
                sscanf_s( szPresetLevel, "%s", s, _countof(s) );

                if (!_stricmp(s, "Low"))
                    presets = Low;
                else if (!_stricmp(s, "Medium"))
                    presets = Medium;
                else if (!_stricmp(s, "High"))
                    presets = High;
                else
                    presets = NotCompatible;

                break; // Done reading file.
            }
        }

        fclose( fp );  // Close open file handle
    }

    // If the current graphics device was not listed in any of the config
   // files, or if config file not found, use Low settings as default.
    if ( presets == Undefined )
        presets = Low;

    return presets;
}
```

Example Configuration File:

```
;
; Intel Graphics Preset Levels
;
; Format:
; VendorIDHex, DeviceIDHex, CapabilityEnum      ;Commented name of cards
;


0x8086, 0x2582, Low    ;  SM2   ; Intel(R) 82915G/GV/910GL Express Chipset Family
0x8086, 0x2782, Low    ;  SM2   ; Intel(R) 82915G/GV/910GL Express Chipset Family
0x8086, 0x2592, Low    ;  SM2   ; Mobile Intel(R) 82915GM/GMS, 910GML Express Chipset Family
```




```
0x8086, 0x2792, Low    ;  SM2   ; Mobile Intel(R) 82915GM/GMS, 910GML Express Chipset Family
0x8086, 0x2772, Low    ;  SM2   ; Intel(R) 82945G Express Chipset Family
0x8086, 0x2776, Low    ;  SM2   ; Intel(R) 82945G Express Chipset Family
0x8086, 0x27A2, Low    ;  SM2   ; Mobile Intel(R) 945GM Express Chipset Family
0x8086, 0x27A6, Low    ;  SM2   ; Mobile Intel(R) 945GM Express Chipset Family
0x8086, 0x2972, Low    ;  SM2   ; Intel(R) 946GZ Express Chipset Family
0x8086, 0x2973, Low    ;  SM2   ; Intel(R) 946GZ Express Chipset Family
0x8086, 0x2992, Low    ;  SM2   ; Intel(R) Q965/Q963 Express Chipset Family
0x8086, 0x2993, Low    ;  SM2   ; Intel(R) Q965/Q963 Express Chipset Family
0x8086, 0x29B2, Low    ;  SM2   ; Intel(R) Q35 Express Chipset Family
0x8086, 0x29B3, Low    ;  SM2   ; Intel(R) Q35 Express Chipset Family
0x8086, 0x29C2, Low    ;  SM2   ; Intel(R) G33/G31 Express Chipset Family
0x8086, 0x29C3, Low    ;  SM2   ; Intel(R) G33/G31 Express Chipset Family
0x8086, 0x29D2, Low    ;  SM2   ; Intel(R) Q33 Express Chipset Family
0x8086, 0x29D3, Low    ;  SM2   ; Intel(R) Q33 Express Chipset Family

0x8086, 0xA001, Low    ;  SM2   ; Intel(R) NetTop Atom D410 (GMA 3150)
0x8086, 0xA002, Low    ;  SM2   ; Intel(R) NetTop Atom D510 (GMA 3150)
0x8086, 0xA011, Low    ;  SM2   ; Intel(R) NetBook Atom N4x0 (GMA 3150)
0x8086, 0xA012, Low    ;  SM2   ; Intel(R) NetBook Atom N4x0 (GMA 3150)

0x8086, 0x29A2, Low    ;  SM3   ; Intel(R) G965 Express Chipset Family
0x8086, 0x29A3, Low    ;  SM3   ; Intel(R) G965 Express Chipset Family

0x8086, 0x8108, Low    ;  SM3   ; Intel(R) GMA 500 (Poulsbo) on MID platforms
0x8086, 0x8109, Low    ;  SM3   ; Intel(R) GMA 500 (Poulsbo) on MID platforms

0x8086, 0x2982, Low    ;  SM4   ; Intel(R) G35 Express Chipset Family
0x8086, 0x2983, Low    ;  SM4   ; Intel(R) G35 Express Chipset Family
0x8086, 0x2A02, Low    ;  SM4   ; Mobile Intel(R) 965 Express Chipset Family
0x8086, 0x2A03, Low    ;  SM4   ; Mobile Intel(R) 965 Express Chipset Family
0x8086, 0x2A12, Low    ;  SM4   ; Mobile Intel(R) 965 Express Chipset Family
0x8086, 0x2A13, Low    ;  SM4   ; Mobile Intel(R) 965 Express Chipset Family
0x8086, 0x2A42, Low    ;  SM4   ; Mobile Intel(R) 4 Series Express Chipset Family
0x8086, 0x2A43, Low    ;  SM4   ; Mobile Intel(R) 4 Series Express Chipset Family
0x8086, 0x2E02, Low    ;  SM4   ; Intel(R) 4 Series Express Chipset
0x8086, 0x2E03, Low    ;  SM4   ; Intel(R) 4 Series Express Chipset
0x8086, 0x2E22, Low    ;  SM4   ; Intel(R) G45/G43 Express Chipset
0x8086, 0x2E23, Low    ;  SM4   ; Intel(R) G45/G43 Express Chipset
0x8086, 0x2E12, Low    ;  SM4   ; Intel(R) Q45/Q43 Express Chipset
0x8086, 0x2E13, Low    ;  SM4   ; Intel(R) Q45/Q43 Express Chipset
0x8086, 0x2E32, Low    ;  SM4   ; Intel(R) G41 Express Chipset
0x8086, 0x2E33, Low    ;  SM4   ; Intel(R) G41 Express Chipset
0x8086, 0x2E42, Low    ;  SM4   ; Intel(R) B43 Express Chipset
0x8086, 0x2E43, Low    ;  SM4   ; Intel(R) B43 Express Chipset
0x8086, 0x2E92, Low    ;  SM4   ; Intel(R) B43 Express Chipset
0x8086, 0x2E93, Low    ;  SM4   ; Intel(R) B43 Express Chipset
0x8086, 0x0046, Low    ;  SM4   ; Intel(R) Processor Graphics - Core i3/i5/i7 Mobile
Processors
0x8086, 0x0042, Low    ;  SM4   ; Intel(R) Processor Graphics - Core i3/i5 + Pentium G9650
Processors

0x8086, 0x0106, Low    ;  SM4.1 ; Mobile SandyBridge Processor GRAPHICS 2000
0x8086, 0x0102, Low    ;  SM4.1 ; Desktop SandyBridge Processor GRAPHICS 2000
0x8086, 0x010A, Low    ;  SM4.1 ; SandyBridge Server

0x8086, 0x0112, Medium ;  SM4.1 ; Desktop SandyBridge Processor GRAPHICS 3000
0x8086, 0x0122, Medium ;  SM4.1 ; Desktop SandyBridge Processor GRAPHICS 3000
0x8086, 0x0116, Medium ;  SM4.1 ; Mobile SandyBridge Processor GRAPHICS 3000

0x8086, 0x0126, Medium ;  SM4.1 ; Mobile SandyBrdige Processor GRAPHICS 3000
```



5 Support

- Intel's processor graphics chipset development community forum:

  http://software.intel.com/en-us/forums/developing-software-for-visual-computing/

- Game programming resources:

  http://software.intel.com/en-us/visual-computing/

- Intel® Software Network:

  http://software.intel.com/en-us/

- Intel® Software Partner Program:

  http://www.intel.com/software/partner/visualcomputing/

- Intel® Visual Adrenaline graphics and gaming campaign:

  http://www.intel.com/software/visualadrenaline/

- Intel® Graphics Performance Analyzers (Intel® GPA):

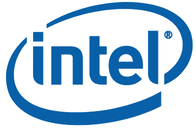
  http://software.intel.com/en-us/articles/intel-gpa/

- Intel® Parallel Studio:

  http://software.intel.com/en-us/articles/intel-parallel-studio-home/

- Intel® VTune™ Amplifier XE (Formerly Intel® VTune™ Performance Analyzer):

  http://software.intel.com/en-us/articles/intel-vtune-amplifier-xe/



6 References

[1] "Copying and Accessing Resource Data (Direct3D 10)". Direct3D Programming Guide. Microsoft DirectX* SDK (November 2008).

[2] "DirectX* Constants Optimizations for Intel processor graphics". Intel Software Network, Intel: http://software.intel.com/en-us/articles/directx-constants-optimizations-for-intel-integrated-graphics/.