Intensely Realistic Graphics Using the 2nd Generation Intel® Core™ Processor Family

Wei Sun, Sr. Research Scientist
Joshua J. Ratcliff, Research Engineer
Contributors: Ninoshka Singh, Warren Hunt

Session RESS005
Agenda

• Overview
• High-precision geometry processing
• Topographic data encoding & retrieval
• Terrain lighting & shading
• Conclusions
Agenda

• Overview
• High-precision geometry processing
• Topographic data encoding & retrieval
• Terrain lighting & shading
• Conclusions
2nd Generation Intel® Core™ Processor

- Formerly codenamed: *Sandy Bridge*
- Multi-core CPU with processor graphics
  - **CPU**
    - Multi-cores
    - Multi-level hardware cache
    - 64-bit capability
    - 4 wide double-precision Intel® Advanced Vector Extensions (Intel® AVX)
  - **Processor graphics**
    - Have access to large cache/system memory
    - Limited in texture sampling (though much improved)
What Can We Do With 2\textsuperscript{nd} Generation Intel\textsuperscript{®} Core\textsuperscript{™} Processor?

- Internal Competition—\textit{Sandy Bridge Challenge}
- Winning application “High-Fidelity Planet Viewer”
  - Real-time full-scale planetary rendering system on a single 2\textsuperscript{nd} Generation Intel\textsuperscript{®} Core\textsuperscript{™} Processor
  - Demo video

Processors used so far, all with real-time performance:
- 2\textsuperscript{nd} Generation Intel\textsuperscript{®} Core\textsuperscript{™} i5-2500K Desktop Processor, 95W
- 2\textsuperscript{nd} Generation Intel\textsuperscript{®} Core\textsuperscript{™} i5-2300 Desktop Processor, 95W
- 2\textsuperscript{nd} Generation Intel\textsuperscript{®} Core\textsuperscript{™} i7-2630QM Mobile Processor, 45W
Why Planetary Rendering?  
A Challenging Graphics Application

• Existing graphics technology mainly focus on rendering small objects, e.g. movies & video games
• Many extra challenges in planetary rendering
  – Huge variation in scale, many different scales co-exist
  – High numerical precision
    ▪ Single-precision floating point arithmetic → visual artifacts
  – Real-time retrieval of huge amount of topographic data
    ▪ Terabytes or more
    ▪ Cannot fit into system memory or even local hard disk
• Realistic lighting and shading
  – Existing approaches rely on high end discrete graphics cards & pre-baking

Computation and memory bandwidth intensive!
How Did We Do It?

- High-precision geometry on the fly on CPU
- Advanced lighting and shading on processor graphics
- Fully parallelizable algorithms
- Data structures efficient for disk I/O, network transmission and memory caching

Planetary Rendering System

Lower power consumption.
More efficient and balanced workload that fully utilizes both CPU and processor graphics.
Higher precision. Visually more realistic.
Agenda

• Overview
• High-precision geometry processing
• Topographic data encoding & retrieval
• Terrain lighting & shading
• Conclusions
High-Precision Geometry Processing

- Approximate a planet with a sphere
- Discrete LODs (levels of detail) + displacement mapping at vertices to compute position, surface normals, texture coordinates
  - No storage required, carefully index vertices, edges and faces to generate mesh topology on demand
- Problems:
  - Need large number of LODs
  - Does not handle scale variation within a scene
High-Precision Geometry Processing

Patch tessellation

- 8 discrete LODs
- Dynamically sub-sample to generate finer triangle mesh (max edge resolution $512 = 9$ additional discrete LODs)
- Displacement mapping at sub-sampled points to compute positions, surface normals, texture coordinates
High-Precision Geometry Processing

- **create hierarchical mesh models**
- **dynamic patch tessellation**
- **vertex & index array**

**discrete LODs**

**adaptive LODs**

**Patch tessellation metric**

- Determine sub-sampling resolution of edges

\[ T(e) = k \frac{(F_x + F_y) \varepsilon_{flat}}{P_{c1} + P_{c2} (1 + \frac{P_{c1}}{|P_{c1}|} \cdot \frac{P_{c2}}{|P_{c2}|})} \]

- viewport dimensions
- surface unevenness
- viewing distance
- viewing angle
High-Precision Geometry Processing

Invisible patch removal

- Hierarchical back face culling
  - $O(n)$ complexity
  - Consider elevation of vertices
- View frustum culling
  - Consider bounding volume of patches
Geometry Processing Optimization

- Hotspot: displacement mapping in generating adaptive LODs
  - Tessellation resolution varies hugely: 1x1~512x512
- Task parallelism → multi-threading
  - Sort patches according to tessellation resolution
    - Highest resolution first (closest to camera)
    - Render front-to-back, minimize overdraw
  - 1 unit task = displacement mapping 1024 (32x32) vertices
    - If patch resolution > 32x32, subdivide patch to >1 tasks
    - If patch resolution = 32x32, dispatch entire patch as 1 task
    - If patch resolution < 32x32, group >1 patches into 1 task
  - Process corner & edge points separately, prevent edge cracking artifacts
Geometry Processing Optimization

• Data parallelism → SIMD
  – 4 wide double-precision Intel® Advanced Vector Extensions (Intel® AVX)
  – Process 4 vertices simultaneously
  – Implement vectorized trigonometric functions using Intel® AVX, e.g. sin(), cos()

• Use 64-bit build configuration
  – Take full advantage of 64-bit hardware capabilities
  – 50% speed up
Geometry Processing Results

More accurate than Google* Earth
High-Precision Geometry Processing

create hierarchical mesh models

hierarchical back face culling

dynamic patch tessellation

vertex & index array

Data Server

Network

Geometry Processing (CPU)

Lighting & Shading (Processor Graphics)

Planetary Rendering System

A general-purpose geometry processing pipeline. Applicable to any graphics rendering system.
Agenda

- Overview
- High-precision geometry processing
- Topographic data encoding & retrieval
- Terrain lighting & shading
- Conclusions
Quadtrees Data Encoding

- Require random data access in any region of interest
- Require LODs (levels of detail) to adapt to varying resolution
- Use quadtree data structure instead of raster lines
2D Haar Transform

\[
\begin{align*}
H_{00} & : \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} & H_{01} & : \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix} & H_{10} & : \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix} & H_{11} & : \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}
\end{align*}
\]

Forward: \( X_{ij} = \begin{bmatrix} x_{00} & x_{01} \\ x_{10} & x_{11} \end{bmatrix} \otimes H_{ij} \)

Inverse: \( x_{ij} = \frac{1}{4} \cdot \begin{bmatrix} X_{00} & X_{01} \\ X_{10} & X_{11} \end{bmatrix} \otimes H_{ij} \)

- Essentially averaging & differencing operations
- Concentrate low frequency components
- In-place operations
- Suitable for bit packing compression
Data Compression/Decompression

Bit packing/unpacking

- Lossless compression
- Light weight, mostly bit shift

16 bits x 16 = 32 Bytes

166 bits < 21 Bytes
2D Maximum Transform

- **Forward transform**
  1. Find the maximum
  2. Rearrange position
  3. Compute difference
  4. Add position bits
- **Inverse transform**
  1. Extract position bits
  2. Add difference
  3. Recover original position
- Essentially maximizing & differencing operations
- Concentrate low frequency components
- Suitable for bit packing

\[
\begin{bmatrix}
X_{00} & X_{01} \\
X_{10} & X_{11}
\end{bmatrix}
\]

\[
\begin{bmatrix}
x_{00} & x_{01} \\
x_{10} & x_{11}
\end{bmatrix}
\]

(1) \( X_{00} = \max\{ x_{00}, x_{01}, x_{10}, x_{11} \} \)

(2) \[
\begin{bmatrix}
X_{00} & x_{01} \\
x_{10} & x_{11}
\end{bmatrix}
\]

(3) \[
\begin{bmatrix}
X_{00} & X_{00} - x'_{01} \\
X_{00} - x'_{10} & X_{00} - x'_{11}
\end{bmatrix}
\]\[
\Rightarrow
\begin{bmatrix}
X_{00} & X_{01} \\
X_{10} & X_{11}
\end{bmatrix}
\]

(4) \[
\begin{bmatrix}
X_{00} & X_{01} \\
X_{10} & X_{11}^*
\end{bmatrix}
\]
2D Minimum Transform

- **Forward transform**
  1. Find the minimum
  2. Rearrange position
  3. Compute difference
  4. Add position bits
- **Inverse transform**
  1. Extract position bits
  2. Add difference
  3. Recover original position
- Essentially minimizing & differencing operations
- Concentrate low frequency components
- Suitable for bit packing

\[
\begin{bmatrix}
  x_{00} & x_{01} \\
  x_{10} & x_{11}
\end{bmatrix} \quad \text{forward transform} \quad \begin{bmatrix}
  X_{00} & X_{01} \\
  X_{10} & X_{11}^*
\end{bmatrix}
\]

(1) \[ X_{00} = \min\{ x_{00}, x_{01}, x_{10}, x_{11} \} \]
\[ X_{00} = x_{00} \quad X_{00} = x_{01} \quad X_{00} = x_{10} \quad X_{00} = x_{11} \]
\[
\begin{bmatrix}
  X_{00} & x_{01} \\
  x_{10} & x_{11}
\end{bmatrix} \quad \Downarrow \quad \begin{bmatrix}
  X_{00} & x_{00} \\
  x_{10} & x_{11}
\end{bmatrix} \quad \Downarrow \quad \begin{bmatrix}
  X_{00} & x_{01} \\
  x_{00} & x_{11}
\end{bmatrix} \quad \Downarrow \quad \begin{bmatrix}
  X_{00} & x_{01} \\
  x_{10} & x_{10}
\end{bmatrix}
\]

(2) \[ \begin{bmatrix}
  X_{00} & x_{01}' \\
  x_{10}' & x_{11}'
\end{bmatrix} \]

(3) \[ \begin{bmatrix}
  X_{00} & x_{01}' - X_{00} \\
  x_{10}' - X_{00} & x_{11}' - X_{00}
\end{bmatrix} \Rightarrow \begin{bmatrix}
  X_{00} & X_{01} \\
  X_{10} & X_{11}
\end{bmatrix}
\]

(4) \[ \begin{bmatrix}
  X_{00} & X_{01} \\
  X_{10} & X_{11}^*
\end{bmatrix} \]
Quadtree Hierarchy

- A single quadtree
  - Base size = 16×16, = 4 LODs
  - Total size ≤ 1 KB, ≈ optimum for network transmission

- Quadtree hierarchy
  - 5 layers of quadtrees = 20 LODs
  - Duplicate child root node at parent leaf node
    - Decode ≤ 1 quadtree during data retrieval
    - No need for storage if data uniform
  - Each quadtree contains
    - bit array: if each child quadtree exists
    - offset values: where each child quadtree is stored
  - Root quadtree in each file contains bit-length tables for bit unpacking
Real-Time Data Retrieval

Problems:

- Load data from local storage
  - Data cannot fit into local hard disk
  - Disk I/O slow
- Synchronous data fetching
  - Geometry processing threads wait on data fetching
  - Serial processing
Real-Time Data Retrieval

- Query data from (cloud) server via network
  - Data stored (distributedly) on server disk, cached in RAM
  - Network transmission faster than disk seek
- Asynchronous data fetching
  - Geometry processing & data fetching on separate threads
  - Parallel processing
Real-Time Data Retrieval

Problems:
- Multi-threaded geometry processing causes new issues
- Shared global cache → race condition
- Cache lock → Lock contention
- Serial processing
Real-Time Data Retrieval

Create hierarchical mesh models
- Discrete LODs

Asynchronous data fetching
- Global cache
- Local cache

Invisible patch removal
- Hierarchical back face culling
- View frustum culling

Dynamic patch tessellation
- Adaptive LODs

Add per-thread private local cache
- Reduce system memory access frequency
- Alleviate race condition & lock contention
- SW cache hierarchy matches CPU HW cache hierarchy
- Improve CPU cache efficiency
Real-Time Data Retrieval

- **Asynchronous data fetching**
  - Global cache
  - Local cache
  - Local cache

- **Create hierarchical mesh models**
  - Discrete LODs

- **Hierarchical back face culling**

- **Invisible patch removal**

- **View frustum culling**

- **Dynamic patch tessellation**
  - Adaptive LODs

- **Vertex & index array**

**Tile data based on adjacency**

- **Quadtree**
  - 16x16, 1 unit task 32x32,
  - Worst case needs 9 quadtrees

- **4 level uncompressed quadtree**
  - 4/3KB, x9=12KB

- **Data needed by a task fit in CPU L1 cache (32KB)**
Real-Time Data Retrieval

Initial fetching of topographic data
- Cache the top layers of quadtree hierarchies
Real-Time Data Retrieval Thread View

SERVER

CLIENT

NETWORK

query

data packet

quadtree available?

assemble packet

store encoded quadtree

load encoded quadtree

server cache

hard disk

query

data packet

bit unpacking

store unpacked quadtree

client cache management threads

client global cache

best available quadtree

inverse transform

store raw data quadtree

per-thread local cache

quadtree available?

Y

store encoded quadtree

inverse transform

store raw data quadtree

quadtree available?

Y

longitude latitude resolution

tri-linear interpolation

geometry processing thread

...
Real-Time Data Retrieval

A general-purpose data management scheme.
Suitable for any application requiring real-time retrieval of large data sets.
Agenda

- Overview
- High-precision geometry processing
- Topographic data encoding & retrieval
- Terrain lighting & shading
- Conclusions
Terrain Lighting & Shading

- OpenGL* Shading Language (GLSL)
- Compute all visual effects at run time from dynamic geometry, no pre-computation or baking
- Use deferred rendering instead of forward rendering, independent of geometry complexity

Planetary Rendering System
Deferred Rendering Pipeline

3D space > $10^6$ triangles

- Create G-buffer
- Surface normal G-buffer

Screen space

- Color G-buffer

Per-pixel lighting:
- Diffuse lighting

Final rasterize

Diffuse lighting

- Assume terrain surface to be Lambertian
- Phong diffuse shading
Diffuse Lighting Result

graphics

diffuse lighting
Deferred Rendering Pipeline

- 3D space >10^6 triangles
  - create G-buffer
  - render light source view

- Screen space
  - color G-buffer
  - surface normal G-buffer
  - position G-buffer
  - shadow depth buffer
  - render camera view
  - shadow mask

- Per-pixel lighting:
  - diffuse lighting + shadow

- Final rasterize

Shadow mapping
- Render light source’s view to generate shadow depth map
- Render camera’s view with depth test against shadow depth map to create shadow mask
Soft Shadow Mapping

- **Shadow mapping**
  - Render Sun’s view (orthographic projection) to generate shadow depth map $z_{shadow}$
  - Render camera’s view with depth test
    \[
    \text{per - pixel} \begin{cases} 
    z(M_{\text{sun}} \cdot P_w) > z_{shadow} \Rightarrow \text{in shadow} \\
    z(M_{\text{sun}} \cdot P_w) \leq z_{shadow} \Rightarrow \text{in light} 
    \end{cases}
    \]

- **Soft shadow**
  - Gaussian smoothing of shadow mask for softer penumbrae
  - Bilinear interpolation for anti-aliasing up-sampling
Shadow Result

diffuse lighting

diffuse lighting + shadow
Screen space ambient occlusion (SSAO)

- Approximate global illumination in screen-space
Screen Space Ambient Occlusion

- Down-sample screen space
- 16+ random neighboring samples per-pixel
- Accumulate occlusion buffer

\[
\text{per-pixel occlusion} = \frac{1}{N} \sum_{i=1}^{N} \frac{(P_i - P_0) \cdot n}{|P_i - P_0|^2}
\]

- Bilinear interpolation for anti-aliasing up-sampling
Screen Space Ambient Occlusion Result

diffuse lighting + shadow
diffuse lighting + shadow + SSAO
Deferred Rendering Pipeline

3D space > $10^6$ triangles
- create G-buffer

3D space
- color G-buffer
- surface normal G-buffer
- position G-buffer
- shadow depth buffer

screen space
- ambient occlusion
- render camera view
- shadow mask
- occlusion buffer
- per-pixel lighting:
  - diffuse lighting
  - + shadow
  - + ambient occlusion
  - + sunlight color

final rasterize
- atmospheric scattering

Simulate time-of-day and season-of-year sunlight color
Atmospheric Scattering

- Natural physics
  - Atmospheric scattering yields dramatic lighting

- Simplify & simulate sunlight color change based on atmospheric distance
  - Trace a ray from camera to sun
  - Compute atmospheric distance travelled
  - Modulate light source color

\[
\begin{align*}
r &= r_0 \\
g &= g_0 \max\left\{g_{\min}, 1 - \left(k_g \frac{d}{d_0}\right)^4\right\} \\
b &= b_0 \max\left\{b_{\min}, 1 - \left(k_b \frac{d}{d_0}\right)^4\right\}
\end{align*}
\]
Performance Optimization on Graphics Processor

• Deferred rendering
  – Decouple rendering speed from triangle count, independent of scene complexity

• Asynchronous rendering
  – Geometry processing and shading on separate threads, fully utilize CPU and graphics processor, provide fluid user interactions
  – Fine tune workload balance between geometry processing on CPU and rendering on graphics processor

• Tunable balance between quality and performance
  – Screen space algorithm speed depends on pixel count, down sampling increases speed & causes aliasing
  – Shadow smoothing filter size
  – SSAO (screen space ambient occlusion) sampling rate
Deferred Rendering Pipeline

3D space > $10^6$ triangles

- create G-buffer
- render light source view

screen space

- color G-buffer
- surface normal G-buffer
- position G-buffer
- shadow depth buffer

- ambient occlusion
- render camera view

- occlusion buffer
- shadow mask

per-pixel lighting:
- diffuse lighting
- + shadow
- + ambient occlusion
- + sunlight color

- final rasterize

atmospheric scattering

A standard deferred rendering pipeline applicable to any rendering system.
Agenda

• Overview
• High-precision geometry processing
• Topographic data encoding & retrieval
• Terrain lighting & shading
• Conclusions
Summary

- Real-time full-scale planetary rendering system on 2nd Generation Intel® Core™ Processor
  - Highly accurate geometry on CPU
  - Realistic lighting & shading on processor graphics
- Efficient and balanced workload that fully utilizes both CPU and processor graphics capabilities
- Keys to performance optimization:
  - Parallel programming
    - Task parallelism: multi-threading, including asynchronous data fetching & asynchronous rendering
    - Data parallelism: SIMD, both CPU and processor graphics
  - 64-bit configuration
  - Cache efficient data management
  - Deferred rendering
Potential Improvements

- **Geometry processing (CPU)**
  - Data prefetching based on user interactions
  - Occlusion culling to further reduce triangle count
  - Better data compression to reduce storage and network

- **Lighting & shading (processor graphics)**
  - Use more flexible software pipeline (e.g. Microsoft* DirectX*11) for better memory management and data parallelism to improve performance
  - Bilateral upsampling based on depth discontinuity for SSAO and shadow
  - Full atmospheric scattering for more realistic sky with non-uniform color
  - High dynamic range (HDR) lighting effect
Call to Action

- Concepts & techniques introduced here are not limited to planetary rendering
  - General-purpose geometry processing and deferred rendering pipeline. Applicable to any graphics rendering system.
  - General-purpose data management scheme. Suitable for any system requiring real-time retrieval of large data sets.
  - General-purpose optimization techniques on CPU and processor graphics. Useful for performance tuning of any application.

*Try it out yourself.*
Additional sources of information on this topic:

- Live demo in the showcase – IP9 @ Intel Software Pavilion
The PDF for this Session presentation is available from our IDF Content Catalog at the end of the day at:

intel.com/go/idfsessionsBJ

URL is on top of Session Agenda Pages in Pocket Guide
Please Fill out the Session Evaluation Form

Give the completed form to the room monitors as you exit!

Thank You for your input, we use it to improve future Intel Developer Forum events
Q&A
Legal Disclaimer

- 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.
- Intel may make changes to specifications and product descriptions at any time, without notice.
- All products, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice.
- Intel, processors, chipsets, and desktop boards 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.
- Sandy Bridge and other code names featured are used internally within Intel to identify products that are in development and not yet publicly announced for release. Customers, licensees and other third parties are not authorized by Intel to use code names in advertising, promotion or marketing of any product or services and any such use of Intel's internal code names is at the sole risk of the user.
- Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark* and MobileMark*, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.
- Intel, Core, Sponsors of Tomorrow. and the Intel logo are trademarks of Intel Corporation in the United States and other countries.
- *Other names and brands may be claimed as the property of others.
- Copyright ©2011 Intel Corporation.
Risk Factors

The above statements and any others in this document that refer to plans and expectations for the first quarter, the year and the future are forward-looking statements that involve a number of risks and uncertainties. Many factors could affect Intel’s actual results, and variances from Intel’s current expectations regarding such factors could cause actual results to differ materially from those expressed in these forward-looking statements. Intel presently considers the following to be the important factors that could cause actual results to differ materially from the corporation’s expectations. Demand could be different from Intel's expectations due to factors including changes in business and economic conditions; customer acceptance of Intel’s and competitors’ products; changes in customer order patterns including order cancellations; and changes in the level of inventory at customers. Intel operates in intensely competitive industries that are characterized by a high percentage of costs that are fixed or difficult to reduce in the short term and product demand that is highly variable and difficult to forecast. Revenue and the gross margin percentage are affected by the timing of Intel product introductions and the demand for and market acceptance of Intel's products; actions taken by Intel's competitors, including product offerings and introductions, marketing programs and pricing pressures and Intel’s response to such actions; and Intel’s ability to respond quickly to technological developments and to incorporate new features into its products. The gross margin percentage could vary significantly from expectations based on capacity utilization; variations in inventory valuation, including variations related to the timing of qualifying products for sale; changes in revenue levels; product mix and pricing; the timing and execution of the manufacturing ramp and associated costs; start-up costs; excess or obsolete inventory; changes in unit costs; defects or disruptions in the supply of materials or resources; product manufacturing quality/yields; and impairments of long-lived assets, including manufacturing, assembly/test and intangible assets. Expenses, particularly certain marketing and compensation expenses, as well as restructuring and asset impairment charges, vary depending on the level of demand for Intel's products and the level of revenue and profits. The majority of Intel’s non-marketable equity investment portfolio balance is concentrated in companies in the flash memory market segment, and declines in this market segment or changes in management’s plans with respect to Intel’s investments in this market segment could result in significant impairment charges, impacting restructuring charges as well as gains/losses on equity investments and interest and other. Intel's results could be impacted by adverse economic, social, political and physical/infrastructure conditions in countries where Intel, its customers or its suppliers operate, including military conflict and other security risks, natural disasters, infrastructure disruptions, health concerns and fluctuations in currency exchange rates. Intel’s results could be affected by the timing of closing of acquisitions and divestitures. Intel's results could be affected by adverse effects associated with product defects and errata (deviations from published specifications), and by litigation or regulatory matters involving intellectual property, stockholder, consumer, antitrust and other issues, such as the litigation and regulatory matters described in Intel's SEC reports. An unfavorable ruling could include monetary damages or an injunction prohibiting us from manufacturing or selling one or more products, precluding particular business practices, impacting Intel’s ability to design its products, or requiring other remedies such as compulsory licensing of intellectual property. A detailed discussion of these and other factors that could affect Intel’s results is included in Intel’s SEC filings, including the report on Form 10-Q for the quarter ended September 25, 2010.