FERMI GF100 GPU ARCHITECTURE

THE FERMI GF100 IS A GPU ARCHITECTURE THAT PROVIDES SEVERAL NEW CAPABILITIES BEYOND THE NVIDIA GT200 OR TESLA ARCHITECTURE. THE FERMI ARCHITECTURE OFFERS UP TO 512 CUDA CORES AND SPECIAL FEATURES FOR GAMING AND HIGH-PERFORMANCE COMPUTING. THIS ARTICLE DESCRIBES THE GPU'S NEW CAPABILITIES FOR TESSELLATION, PHYSICS PROCESSING, AND COMPUTATIONAL GRAPHICS.

Craig M. Wittenbrink
Emmett Kilgariff
Arjun Prabhu
Nvidia

The Fermi GF100 is a GPU architecture that excels in graphics and high-performance computing. First demonstrated for its CUDA capabilities,[1] Fermi GF100 GPU products became available in March 2010.

The GF100 uses an enhanced unified shader architecture that incorporates tessellation shaders into the same vertex, geometry, and pixel shader architecture. The Fermi architecture's primary benefits are the addition of tessellation and improved compute, physics processing, and computational graphics capabilities.

Using the GF100, GF104, and GT200 GPUs graphics rendering benchmarks, this article provides comparative performance figures and discusses our tessellation architecture's performance advantages and challenges. We include top games and discuss details of our Supersonic Sled demonstration (www.youtube.com/watch?v=6RdIrY6NYrM), which we developed to highlight the Fermi architecture's graphics innovations.

We originally presented details of the Fermi architecture at Hot Chips 2010[2] and High Performance Graphics in the Hot3D Track.[3] Expanding on those and other earlier works,[1,4,5] this article describes the architectural benefits, memory system, and scaling to a family of Nvidia GPUs.

GF100 enhancements

Figure 1 shows a die photo of the GF100, which has 512 CUDA cores and 3 billion transistors. The GF100 is manufactured on the 40-nm Taiwan Semiconductor Manufacturing Company (TSMC) process. It provides new compute, tessellation, physics processing, and computational graphics capabilities over our prior Nvidia GT200 or Tesla architecture.

We improved compute and high-performance computing over the GT200 by increasing the double-precision floating-point capability to eight times that of the GT200 and adding C++ programmability features. CUDA architecture enhancements include error-correcting code (ECC), faster atomics, and faster reductions.[1,4]

Recently released games and our launch demos show the benefits of Microsoft DX11 tessellation[6] and physics simulation. (GT200 does not support tessellation.) The added tessellation shaders control patch, triangle, and geometry creation inside the GPU. Our Supersonic Sled demonstration shows that we can generate more detailed terrain, with less required memory bandwidth using tessellation. The GF100 has a graphics double data rate version 5 (GDDR5) DRAM memory system of up to 6 Gbytes to provide high throughput for both compute and graphics.

Real-time physics simulation creates higher visual realism. Nvidia has developed an open standard physics API, called PhysX, with support on the GF100. PhysX does visual simulation in the Supersonic Sled demo, computing the interaction among all the objects. The GF100 also has enhancements to the CUDA core instruction set for general-purpose programming[1,5] that benefits physics processing.[5]

The GPU also computes computational graphics techniques such as transparency, motion blur, and post-resolve reconstruction. Additional computational graphics techniques are used in ray tracing and global illumination that can be used in visual styling applications or videogames. The GF100 includes Z compare and blend raster operation (ROP) units and compression technologies that provide greater performance for graphics.

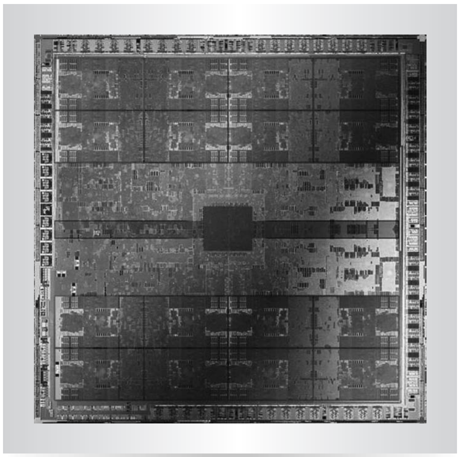
Figure 1. GF100 die photo. The GPU includes 3 billion transistors and was manufactured on the 40-nm Taiwan Semiconductor Manufacturing Company (TSMC) process.

Architecture overview

Figure 2 shows the overall GF100 architecture. Key features include six memory controllers surrounding the outside, four graphics processor clusters (GPCs), a GigaThread engine, a host interface, and an on-chip shared Level 2 (L2) read/write cache. The GF100 has a distributed rasterizer with a rasterizer in each GPC, while the GT200 has a single rasterizer. Each GPC contains four streaming multiprocessors (SMs). There are also 16 PolyMorph engines, one for each SM.

Streaming multiprocessor architecture

The GF100 SM has 32 CUDA cores, four times the number per SM in the GT200. Each GPC includes four SMs, for a total of 512 CUDA cores in the GF100. Each SM includes a configurable cache or local memory for a total of 48 Kbytes of shared memory with 16 Kbytes of L1 cache. This memory can be reconfigured to use 16 Kbytes of shared memory and 48 Kbytes of L1 cache. There was no L1 cache for the SM in the GT200 architecture. Figure 3 shows a single SM with 32 CUDA cores.

We enhanced the instruction set architecture (ISA) to add more 32-bit integer operations and fused multiply add (FMA) floating-point operations. Each CUDA core has floating-point and integer logic and executes in parallel using instructions and operands from the shared instruction issue and register file. Both CUDA and graphics use the four texture units and L1 texture cache. Each SM is paired with a PolyMorph engine that has special fixed-function and programmable logic for graphics attribute processing and tessellation.

Cache architecture

The GF100 has several cache hierarchies to keep data on-chip for the life of graphics or CUDA processing. The L1 data cache can be used for register spilling, for stack operations, or to improve efficiency of global load and store operations. The L1 cache is backed up by a shared L2 cache. The L2 cache is a read/write cache with write-back replacement policy. The GT200 uses a read-only L2 cache for textures. For graphics, the L2 cache provides the on-chip storage for vertex data, vertex attributes—positions, color, and so on—and rasterized pixels. For CUDA, the L2 read/write cache provides more on-chip storage for global loads and stores. CUDA also benefits from L2 caching of the texture loads because that path is used

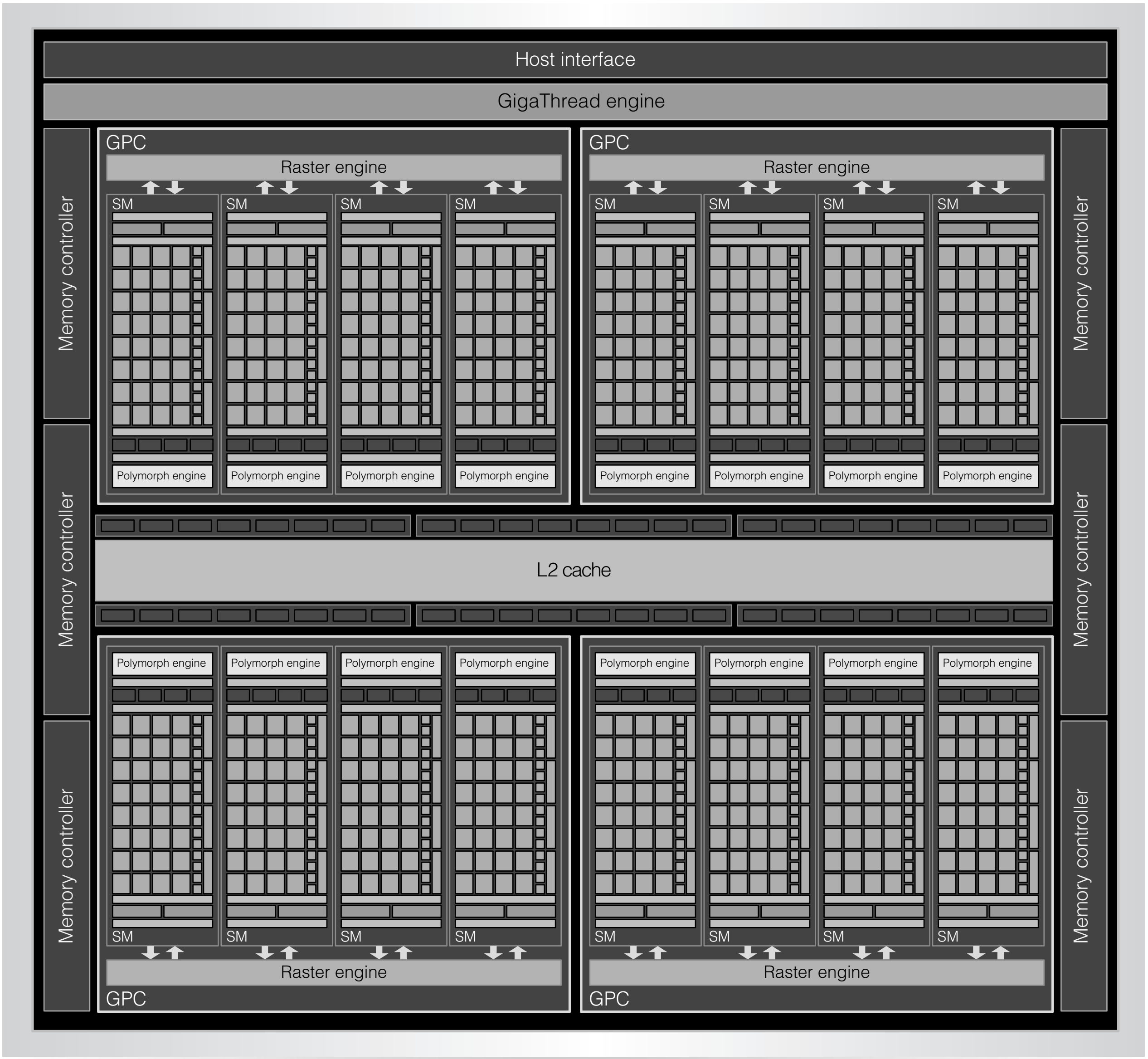

Figure 2. GF100 architectural overview. Each graphics processor cluster contains four streaming multiprocessors (SMs).

for additional loads. Table 1 compares the GT200 and GF100 caches.

Memory system

The memory system incorporates multiple memory controllers. Figure 2 shows the memory controllers and unified L2 cache. The memory controllers, L2 cache, and ROP units are closely coupled to scale across the product family. Because the L2 cache is unified and all clients use it as a read-writeable cache, requests are shared among the multiple engines, such as the PolyMorph and texture engines. To support tessellation efficiently, the PolyMorph engine data stay on-chip in the cache, while often, texture maps are large enough that they must be fetched from off-chip and streamed through the cache. The cache naturally does this by replacing older data.

For graphics, we use blocking formats to efficiently stride across memory, satisfying the conflicting requirements to support texture fetches, color, and Z surfaces. We support paged memory,

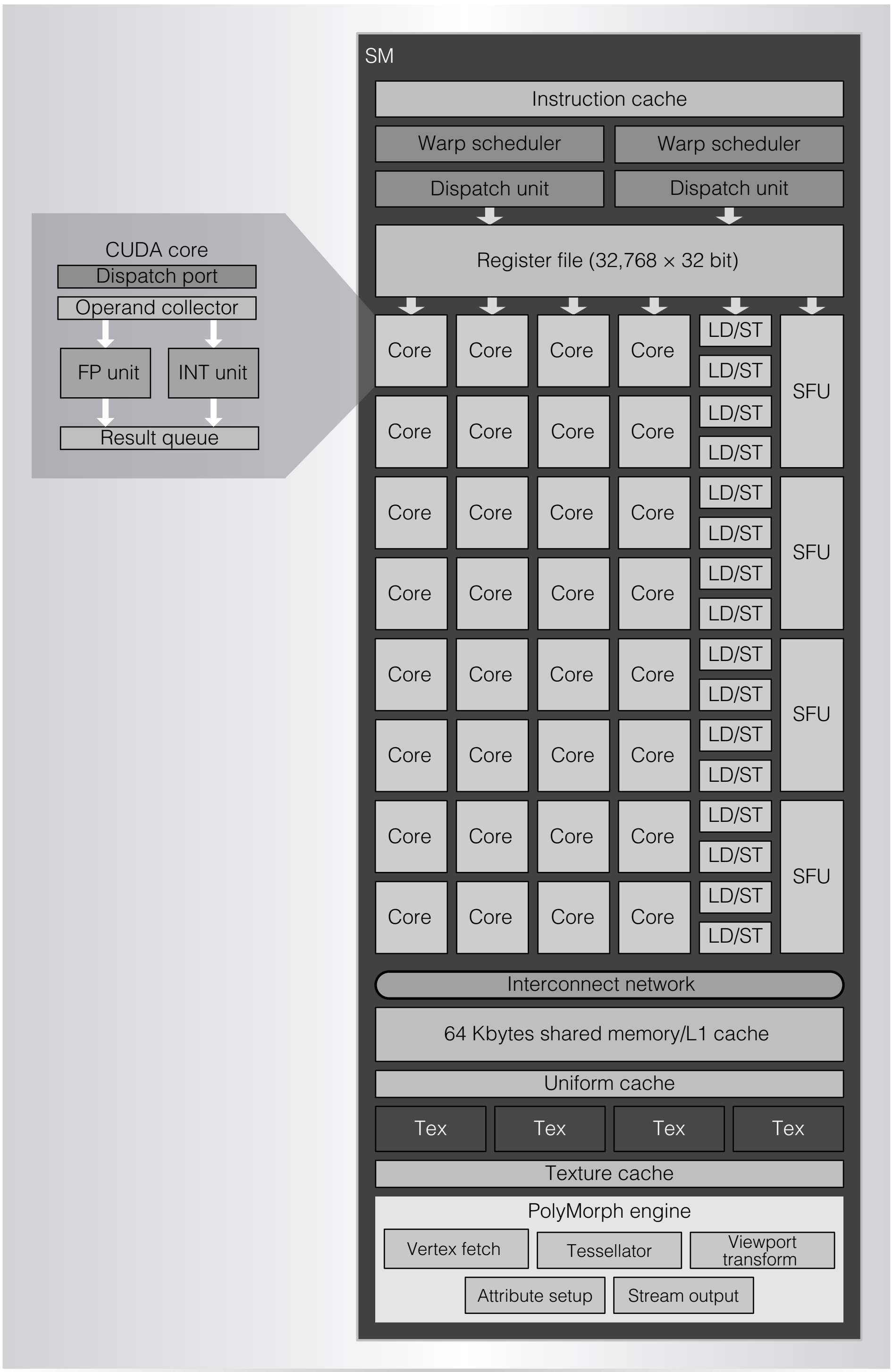

Figure 3. GF100 streaming multiprocessor architecture. A single streaming multiprocessor contains 32 CUDA cores.

with support for multiple page sizes, tailored to efficient graphics processing. We have a 40-bit address space supporting large frame buffers, and we use small and large page sizes to improve heterogeneous computing by sharing and migrating data to and from the system memory.

Table 1. GT200 versus GF100 caches.

Feature	GT200	GF100	Benefit
L1 texture cache (per streaming multiprocessor)	12 Kbytes	12 Kbytes	Fast texture filtering
Dedicated L1 load/store cache	N/A	16 or 48 Kbytes	Efficient physics and ray tracing
Total shared memory	16 Kbytes	16 or 48 Kbytes	More data reuse among threads
L2 cache	256 Kbytes (Tex read only)	768 Kbytes (all clients read/write)	Greater texture coverage and robust compute performance

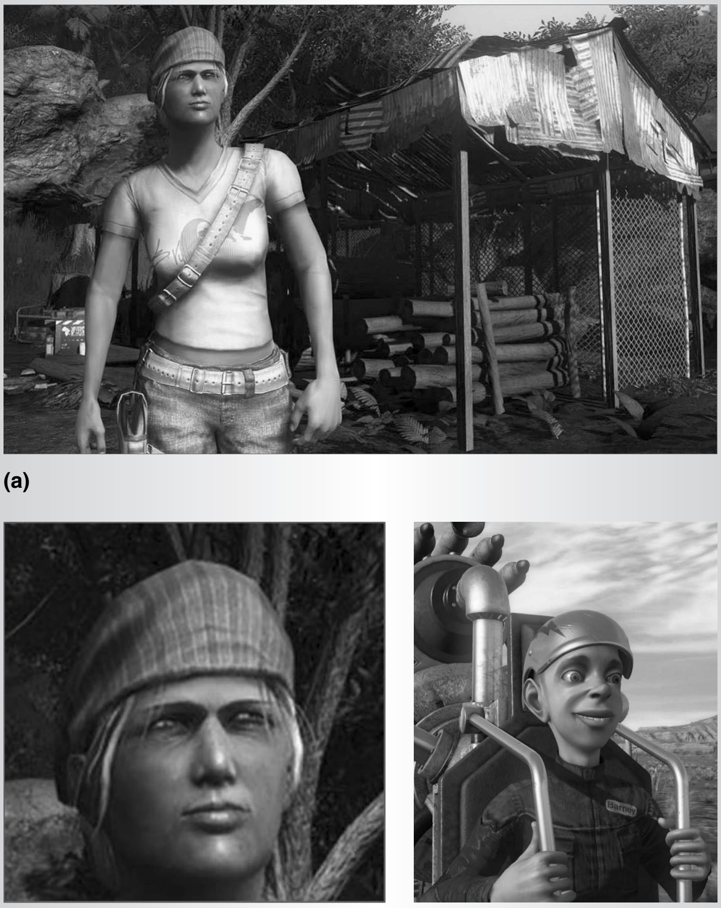

Figure 4. Comparing visual artifacts. The character from DX10 *Far Cry 2* (courtesy of Ubisoft) has a bandana to hide its lack of hair (a). The up-close image shows the coarse silhouette (b). In comparison, the image of the Supersonic Sled pilot (copyright Nvidia) is more detailed due to the use of tessellation (c).

Scaling games to film quality

Computer games have visual artifacts resulting from too little geometric detail. Figure 4a shows a character from *Far Cry 2* (courtesy of Ubisoft). The bandana covering is used to hide the character's lack of hair. The up-close image in Figure 4b shows a coarse silhouette. The blocked edges at the silhouette of the character's bandana result from limited geometry or triangles used in the model. The bandana's texture is good, but the insufficient geometry detail shows up on the edges. And the character might not need the bandana, if it was not too expensive to render geometry for its hair.

Figure 4c shows a tessellated character from our real-time demonstration program, Supersonic Sled. Tessellation is the process of going from models, to patches, to finer geometric detail. This process is more efficient, and therefore, more geometry is used. The pilot's textures and silhouette are detailed because there is a lot of geometry to model the character. This demonstrates tessellation's benefit for scaling games to achieve film-rendering quality.

Tessellation

Tessellation is a significant new feature in the GF100 specified by Microsoft DX11. To understand the architecture, it's important to understand the processing steps.

Figure 5 shows the key steps in tessellation processing using the Imp character, courtesy of the game engine and game developer id Software. In Figure 5a, the character is shown in *quad patches*, or the control patches that modelers use to create characters. The Imp is rendered smoothly by interpolating the quad patches to fine geometry (see Figure 5b). Finally, in Figure 5c, the fine geometry is displaced to create more detail in the final character geometry. The renderings do not use texture maps to illustrate how more geometry

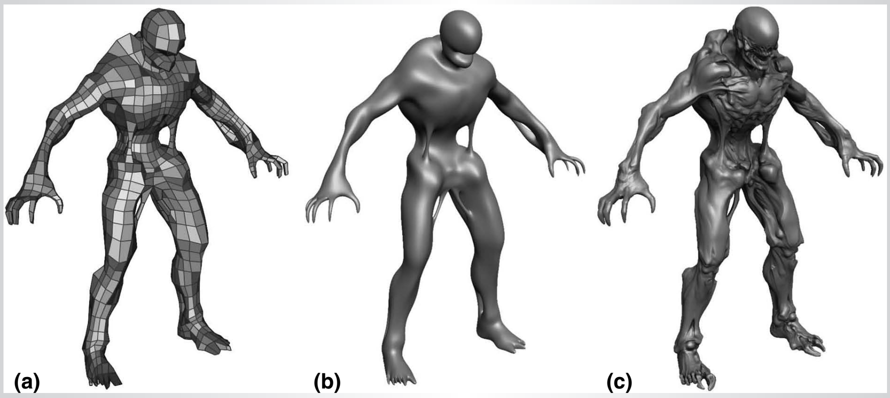

Figure 5. The id Software Imp. Modelers used quad patches to create the character (a). The character is tessellated, or rendered smoothly by interpolating the quad patches to fine geometry (b), and then displacement mapped to create more detail (c). (Copyright id Software, a ZeniMax Media company. All rights reserved.)

detail is created. Textures can provide further character detail.

Our architecture implements the new processing pipeline demonstrated in Figure 5. First, vertex data are processed by being projected to the screen. Next, the patches are processed, putting vertices together to define them. The patch is used as input to a hull shader, and it outputs tessellation factors and modes. The vertices and patches can be fetched from memory. The tessellation factor controls geometry detail and encodes how much new geometry to create. We make this process dynamic, computing it on the fly, because the amount of geometry needed changes as the size of the character changes on the screen.

The tessellator takes tessellation factors and modes and then outputs triangles and lines. A significant data expansion can occur at this point, but we can do it efficiently because it occurs on-chip.

In the next step, the domain shader uses the tessellation geometry and might apply displacements that pull the geometry to new locations. The results are the final geometry positions. The rendered output from the resulting triangles is the tessellated, displacement mapped Imp in Figure 5c.

Figure 6 shows the tessellation in the DX11 graphics pipeline that adds new stages to the existing graphics primitive processing pipeline. The new stages are the patch assembly, hull, tessellator, and domain shader. The pipeline blocks are implemented using a mix of fixed-function hardware and the SM shaders. Each step in the pipeline is a functional block that calculates the resulting graphics triangles. The patch assembly takes vertices from the vertex processing stage. The hull shader uses a quad patch and computes a tessellation factor that controls how many subdivisions to create for the patch. This is predata expansion, so we use a single arrow to show where a single transfer for each patch is made to the tessellator. A single patch can create many triangles, and we use multiple arrows to show where this results in data expansion in the GPU. The domain shader then computes the fine detail, using geometric displacements on the fine geometry. Figure 5c shows how the displacements change the final rendered appearance. The domain shader provides data to the legacy primitive assembly. Primitive assembly creates triangles using three vertices, their colors, and other attributes. (More details on tessellation are available elsewhere.[6])

Figure 7 shows how the shader horsepower for different generations of Nvidia GPUs has increased. There has been a steady improvement in shader teraflops per second, but the geometry (giga-triangles per second) has not increased at the same rate. There was a larger increase in giga-triangles

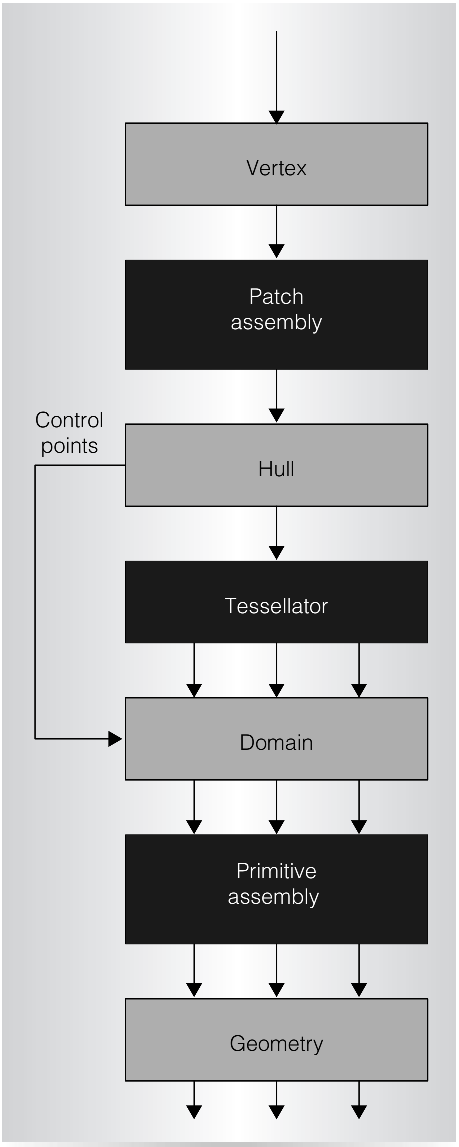

Figure 6. Tessellation in DX11. The new pipeline stages include the patch assembly, hull, tessellator, and domain shader. A single arrow indicates a single transfer for each quad patch to the tessellator. Multiple arrows indicate when a single patch creates many triangles, resulting in a data expansion in the GPU.

per second from GT200/GTX285 to GF100/GTX480. The geometry throughput increase with distributed rasterization is necessary for strong tessellation performance.

To implement high geometry throughput in the GF100, we use distributed raster engines. The parallel tessellation and attribute processing is computed in the multiple PolyMorph engines (see Figure 2). For the GF100, the four raster engines and 16 PolyMorph engines provide eight times the geometry performance of the GT200. Figure 8 shows that, over time, the personal computer games and demonstrations have had only modest increases in the number of polygons per frame. However, new games such as *Metro 2033* and demonstrations such as Stone Giant and Heaven 2.1 have moved to tens of millions of triangles per frame, enabled by tessellation. More games will continue to adopt higher geometry realism given the Fermi architecture and helpful demonstration software.

Physics processing

Physics processing on GPUs provides improved visual realism. In the GF100, the increased shader horsepower provides a more than two-times improvement for physics processing. In Supersonic Sled, the barrels, sled, rock arch, and wood bridge are physically modeled using our open API, PhysX. Additional PhysX functionality includes particle simulation and the smoke and fire from the sled exhaust.

Using physics modeling, animators and game designers can specify the type of effect, but the real-time processing creates more realism. Other effects include water splashes, mud, and blood. There are added features in the GPU to accelerate PhysX, including the faster atomics and reductions, parallel guaranteed synchronization and summing calculations, and the L1 and L2 cache hierarchies we discussed earlier.

Computational graphics

Computational graphics provides better graphics by computing nontraditional solutions. A good example is the motion blur in Supersonic Sled. As the sled races past the background seen from different auto-selected camera angles, the background blurs to indicate the high speed. This blurring is computed as a post-process on the rendered image using multiple render target (MRT) frames of the color buffer and velocity buffer. The velocity buffer contains each pixel's screen space velocity. The velocities are computed using the previous model and camera positions. The velocity image is segmented using image processing. Then the color buffer can be blurred in the proper direction for those background locations.

Figure 9 shows an additional computational graphics approach, ray tracing. Although the Ferrari looks real, it is synthetically rendered on the GPU using path tracing in our Nvidia OptiX libraries. OptiX uses ray clustering and exploits the L1 and L2 caches to get a four-times speedup over the GT200. Many rays are cast from the eye point into the scene, and subsequent reflected and refracted rays are computed to simulate the bouncing and transmission of light. Reflected rays are visible in the shiny cobblestone and car fender. Refracted rays can be seen in the view through the car side window and windshield. Shadow effects under the car also result from computing reflected rays that have no path to direct illumination from the sunlight source. Other computational graphics techniques include AI, where character motion in the game has shown a more than three-times improvement over GT200.

Fermi architecture family scaling

By changing the Fermi architecture to a distributed rasterization system, we achieved a straightforward scaling to the rest of the lineup by using fewer GPCs. Table 2 shows the scaling of units between the GF100 and GF104. Key changes are the modified SM. Whereas the GF100 is targeted for consumer graphics and high-performance computing CUDA, the GF104 is targeted primarily toward the consumer. Therefore, the GF104 does not require ECC and needs less double-precision floating-point throughput.

The GF104 uses an alternate SM architecture with increased texture and FP32 arithmetic throughput, while reducing FP64 throughput and eliminating ECC support. We can create other chips by combining different numbers of units because the Fermi architecture provides a natural scaling to a family of GPUs. With the GT200 architecture, the nonreplicated rasterization engine and lack of a GPC required more work to scale to smaller systems. With fewer GPCs, there are naturally fewer SMs and fewer rasterizers and PolyMorph engines.

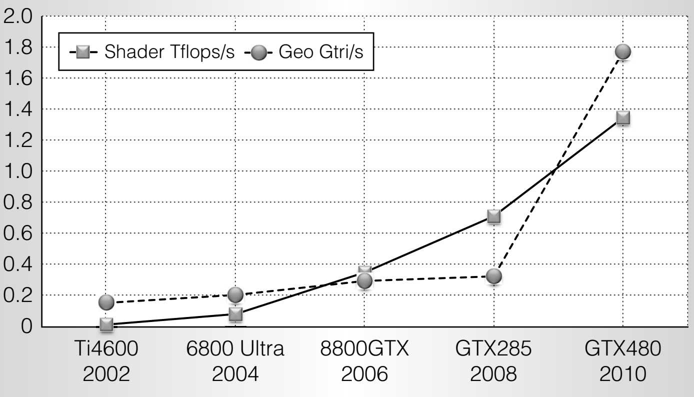

Figure 7. GPU generations showing shader horsepower in teraflops per second and geometry horsepower in giga-triangles per second.

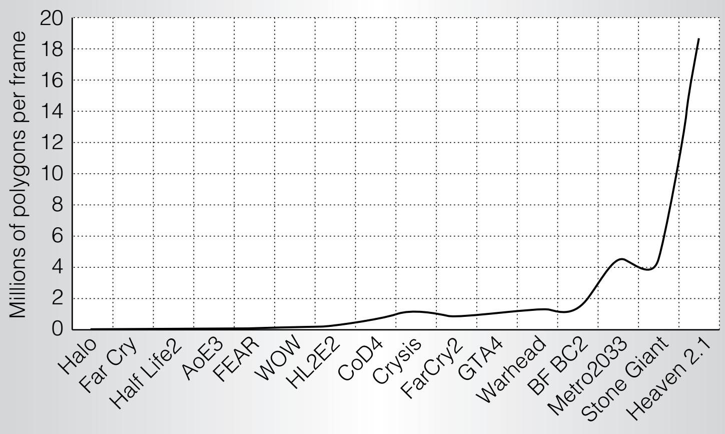

Figure 8. Advances in geometric complexity. Over time, new games and demonstrations enabled by tessellation have moved to tens of millions of triangles per frame.

The memory system is also scalable; the GF100 has six memory controllers, while the GF104 has four. Table 2 also compares the frame buffer pins.

Figure 10 shows the performance scaling for the GF104 to GF100 to GF100 scalable link interconnect. SLI lets us use two GF100s in a single system to provide even higher performance in a scalable way. The highest frame rate is achieved with aliased rendering, where one sample is taken per pixel, but we can get higher quality with anti-aliasing. The result here uses four-times multisampling that provides four times the number of rendered pixels.

Table 2. GF100 versus GF104 scale of units.

GPU	GPC	CUDA cores	Frame buffer pins	ECC	Total L2	Total L1	Tex	Double precision Gflops/sec
GF100	4	512	384	Yes	768 Kbytes	256 Kbytes	16 units	768
GF104	2	384	256	No	512 Kbytes	128 Kbytes	16 units	96

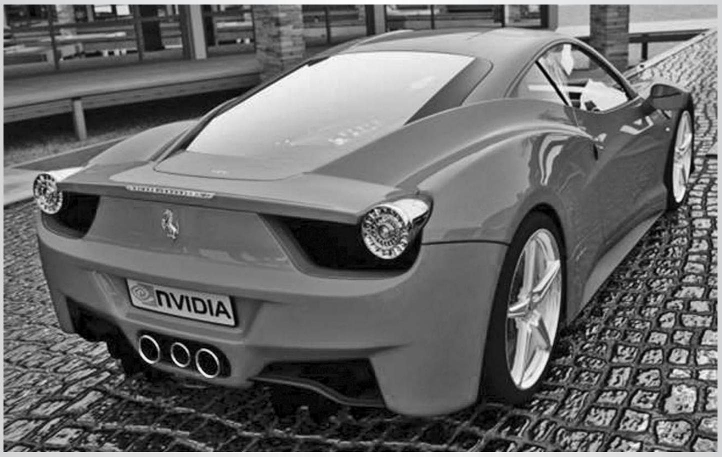
Figure 9. GPU ray tracing with OptiX. The Ferrari is synthetically rendered on the GPU using path tracing in our Nvidia OptiX libraries.

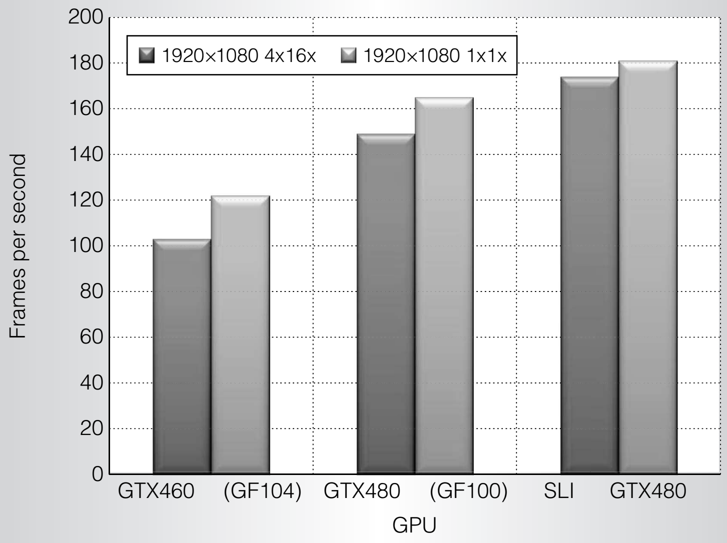

Figure 10. Hawx 2 Fermi GPU performance. We measure the average frames per second for a 1,920 × 1,080 resolution image with aliased (1x AA) no texture aniso and multisampled (4x AA) rendering with 16-times texture anisotropic texture filtering.

The rendered pixels are down filtered after all the drawing is done. The GPU processes four times the pixels and down filters to get higher quality. In this benchmarking, Hawx 2 is using DX11 tessellation to provide high geometry detail.

We continue to investigate architectural innovations that will advance high-performance computing and graphics. Given new markets, competition, and new technologies, GPU architectures must continue to rapidly evolve to provide people with added value. We are focusing on generalized compute, memory systems, graphics algorithms, and ASIC implementation. MICRO

Acknowledgments

We thank Jonah Alben, John Nickolls, John Robinson, Mark Daly, Henry Moreton, John Montrym, and the entire Fermi development team for the success of Fermi.

References

1. NVIDIA, ''Fermi: NVIDIA's Next Generation CUDA Compute Architecture,'' 2009; http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf.
2. C.M. Wittenbrink, E. Kilgariff, and A. Prabhu, ''Fermi GF100: A Graphics Processing Unit (GPU) Architecture for Compute Tessellation, Physics, and Computational Graphics,'' IEEE Hot Chips, presentation, 2010; http://www.hotchips.org/uploads/archive22/HC22.23.110-1-Wittenbrink-Fermi-GF100.pdf.
3. T. Purcell, ''Fast Tessellated Rendering on the Fermi GF100,'' High Performance Graphics Conf., Hot 3D presentation, 2010; http://www.highperformancegraphics.org/previous/www_2010/media/Hot3D/HPG2010_Hot3D_NVIDIA.pdf.

4. J. Nickolls and W.J. Dally, ''The GPU Computing ERA,'' *IEEE Micro,* vol. 30, no. 2, 2010, pp. 56-69.
5. NVIDIA, ''NVIDIA GF100: World's Fastest GPU Delivering Great Gaming Performance with True Geometric Realism,'' 2010; http://www.nvidia.com/object/IO_86775.html.
6. Microsoft, ''MSDN Tessellation Overview'' 2010; http://msdn.microsoft.com/en-us/library/ff476340%28VS.85%29.aspx.

Craig M. Wittenbrink is an architecture director of 3D computer graphics at Nvidia. He has worked in computer architecture, scientific visualization, and GPU architecture and is currently managing GPU architecture design. Wittenbrink has a PhD in electrical engineering from the University of Washington. He's a member of ACM Siggraph, a senior member of IEEE, and a member of the IEEE Computer Society.

Emmett Kilgariff is a vice president of architecture in the GPU group at Nvidia, where he has been responsible for the design of many GeForce chips, including the GF1xx/Fermi product family. Kilgariff has a BS in electrical engineering from Purdue University.

Arjun Prabhu is a vice president of GPU ASIC engineering at Nvidia, where he's responsible for the microarchitecture, logic design, and chip definitions for the Kepler product family for GeForce, Quadro, and Tesla. Prabhu has an MS in electrical engineering from Stanford University.

Direct questions and comments about this article to Craig M. Wittenbrink, Nvidia, 2701 San Tomas Expressway, Santa Clara, CA 95050; cwittenbrink@nvidia.com.