Whitepaper

NVIDIA GF100

World's Fastest GPU Delivering Great Gaming Performance with True Geometric Realism

V1.5

Dedicated to the World’s PC Gamers

Table of Contents

Introducing GF100

Over the years, the continuing and insatiable demand for high quality 3D graphics has driven NVIDIA to create significant GPU architectural innovations. In 1999, the GeForce 256 enabled hardware transform and lighting. In 2001, GeForce 3 introduced programmable shading. Later, GeForce FX provided full 32-bit floating point precision throughout the GPU. And in 2006, GeForce 8 introduced a powerful and efficient unified, scalar shader design. Each GPU we designed was intended to take graphics closer to reality, and to distinguish the PC as the most dynamic and technologically advanced gaming platform.

NVIDIA's latest GPU, codenamed **GF100**[1], is the first GPU based on the *Fermi* architecture. GF100 implements all DirectX 11 hardware features, including tessellation and DirectCompute, among others. GF100 brings forward a vastly improved compute architecture designed specifically to support next generation gaming effects such as raytracing, order-independent transparency, and fluid simulations.

Game performance and image quality receive a tremendous boost, and GF100 enables film-like **geometric realism** for game characters and objects. Geometric realism is central to the GF100 architectural enhancements for graphics. In addition, PhysX simulations are much faster, and developers can utilize GPU computing features in games most effectively with GF100.

In designing GF100, our goals were to deliver:

- **Exceptional Gaming Performance**
- **First-rate image quality**
- **Film-like Geometric Realism**
- **A Revolutionary Compute Architecture for Gaming**

Exceptional Gaming Performance

First and foremost, GF100 is designed for gaming performance leadership. Based on Fermi's third generation Streaming Multiprocessor (SM) architecture, GF100 doubles the number of CUDA cores over the previous architecture.

The geometry pipeline is significantly revamped, with vastly improved performance in geometry shading, stream out, and culling. The number of ROP (Raster Operations) units per ROP partition is doubled and fillrate is greatly improved, enabling multiple displays to be driven with ease. 8xMSAA performance is vastly improved through enhanced ROP compression. The additional ROP units also better balance overall GPU throughput even for portions of the scene that cannot be compressed.

First-rate image quality

GF100 implements a new 32xCSAA (Coverage Sampling Antialiasing) mode based on eight multisamples and 24 coverage samples. CSAA has also been extended to support alpha-to-coverage (transparency multisampling) on all samples, enabling smoother rendering of foliage and transparent textures. GF100 produces the highest quality antialiasing for both polygon edges and alpha textures with minimal performance penalty. Shadow mapping performance is greatly increased with hardware accelerated DirectX 11 four-offset Gather4.

[1] "GF" denotes that the chip is a **G**raphics solution based on the ***F**ermi* architecture. "100" denotes that this is the high end part of the "GF" family of GPUs.

Film-like Geometric Realism

While programmable shading has allowed PC games to mimic film in per-pixel effects, geometric realism has lagged behind. The most advanced PC games today use one to two million polygons per frame. By contrast, a typical frame in a computer generated film uses hundreds of millions of polygons. This disparity can be partly traced to hardware—while the number of pixel shaders has grown from one to many hundreds, the triangle setup engine has remained a singular unit, greatly affecting the relative pixel versus geometry processing capabilities of today's GPUs. For example, the GeForce GTX 285 has more than 150× the shading horsepower of the GeForce FX, but less than 3× the geometry processing rate. The outcome is such that pixels are shaded meticulously, but geometric detail is comparatively modest.

In tackling geometric realism, we looked to movies for inspiration. The intimately detailed characters in computed generated films are made possible by two key techniques: tessellation and displacement mapping. Tessellation refines large triangles into collections of smaller triangles, while displacement mapping changes their relative position. In conjunction, these two techniques allow arbitrarily complex models to be formed from relatively simple descriptions. Some of our favorite movie characters, such as Davy Jones from *Pirates of the Caribbean* were created using these techniques.

GF100's entire graphics pipeline is designed to deliver high performance in tessellation and geometry throughput. GF100 replaces the traditional geometry processing architecture at the front end of the graphics pipeline with an entirely new distributed geometry processing architecture that is implemented using multiple "PolyMorph Engines" . Each PolyMorph Engine includes a tessellation unit, an attribute setup unit, and other geometry processing units. Each SM has its own dedicated PolyMorph Engine (we provide more details on the Polymorph Engine in the GF100 architecture sections below). Newly generated primitives are converted to pixels by four Raster Engines that operate in parallel (compared to a single Raster Engine in prior generation GPUs). On-chip L1 and L2 caches enable high bandwidth transfer of primitive attributes between the SM and the tessellation unit as well as between different SMs. Tessellation and all its supporting stages are performed in parallel on GF100, enabling breathtaking geometry throughput.

While GF100 includes many enhancements and performance improvements over past GPU architectures, the ability to perform **parallel geometry processing** is possibly the single most important GF100 architectural improvement. The ability to deliver setup rates exceeding one primitive per clock while maintaining correct rendering order is a significant technical achievement never before done in a GPU.

Revolutionary Compute Architecture for Gaming

The rasterization pipeline has come a long way, but as games aspire to film quality, graphics is moving toward advanced algorithms that require the GPU to perform general computation along with programmable shading. G80 was the first NVIDIA GPU to include compute features. GF100 benefits from what we learned on G80 in order to significantly improve compute features for gaming.

GF100 leverages Fermi's revolutionary compute architecture for gaming applications. In graphics, threads operate independently, with a predetermined pipeline, and exhibit good memory access locality. Compute threads on the other hand often communicate with each other, work in no predetermined fashion, and often read and write to different parts of memory. Major compute features improved on GF100 that will be useful in games include faster context switching between graphics and PhysX, concurrent compute kernel execution, and an enhanced caching architecture which is good for irregular algorithms such as ray tracing and AI algorithms. We will discuss these features in more detail in subsequent sections of this paper.

Vastly improved atomic operation performance allows threads to safely cooperate through work queues, accelerating novel rendering algorithms. For example, fast atomic operations allow transparent objects to be rendered without presorting (order independent transparency) enabling developers to create levels with complex glass environments.

For seamless interoperation with graphics, GF100's GigaThread engine reduces context switch time to about 20 microseconds, making it possible to execute multiple compute and physics kernels for each frame. For example, a game may use DirectX 11 to render the scene, switch to CUDA for selective ray tracing, call a Direct Compute kernel for post processing, and perform fluid simulations using PhysX.

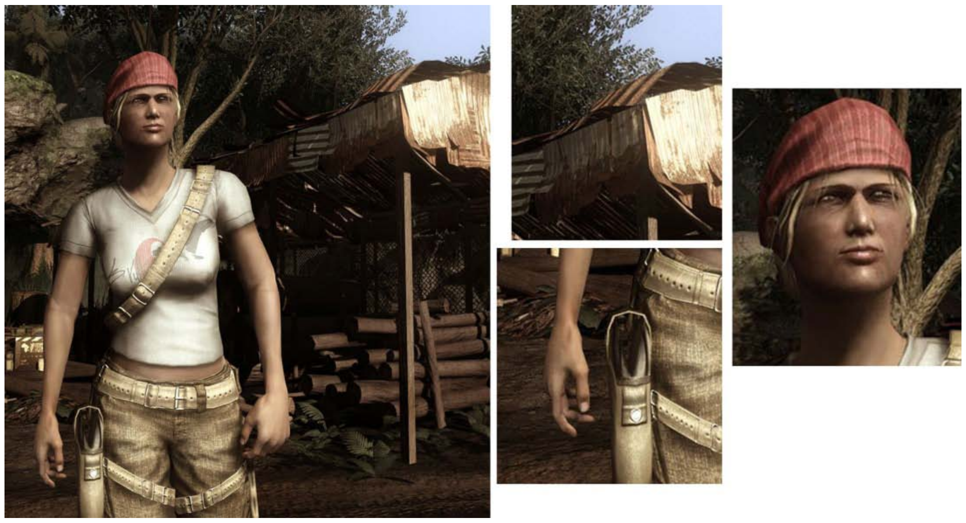
Geometric Realism

Tessellation and Displacement Mapping Overview

While tessellation and displacement mapping are not new rendering techniques, up until now, they have mostly been used in films. With the introduction of DirectX 11 and NVIDIA's GF100, developers will be able to harness these powerful techniques for gaming applications. In this section we will discuss some of the characteristics and benefits of tessellation and displacement mapping in the context of game development and high-quality, realtime rendering.

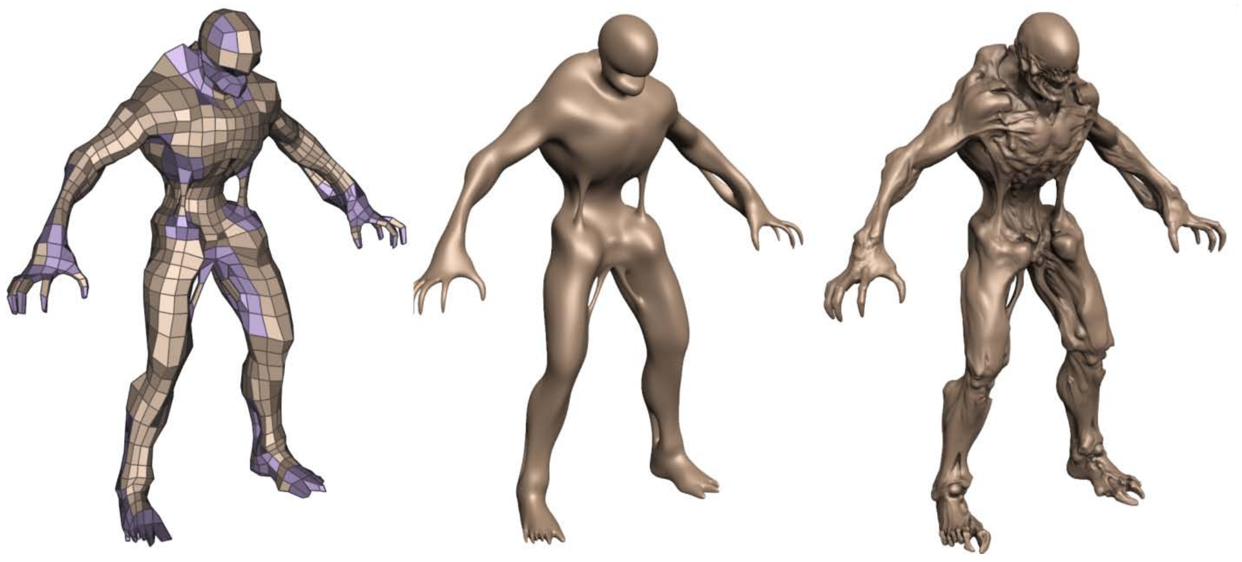
Game assets such as objects and characters are typically created using software modeling packages like Mudbox, ZBrush, 3D Studio Max, Maya, or SoftImage. These packages provide tools based on surfaces with displacement mapping to aid the artist in creating detailed characters and environments. Today, the artist must manually create polygonal models at various levels of detail as required by the various rendering scenarios in the game in order to maintain playable frame-rates. These models are meshes of triangles with associated texture maps needed for proper shading. When used in a game, the model information is sent per frame to the GPU through its host interface. Game developers tend to use relatively simple geometric models due to the limited bandwidth of the PCI Express bus and the modest geometry throughput of current GPUs.

Even in the best of game titles, there are geometric artifacts due to limitations of existing graphics APIs and GPUs. The result of compromising geometric complexity can be seen in the images below. The holster has a heavily faceted or segmented strap. The corrugated roof, which should look wavy, is in fact a flat surface with a striped texture. Finally, like most characters in games, this person wears a hat, carefully sidestepping the complexity of rendering hair.


Due to limitations in existing graphics APIs and GPUs, even graphically advanced games are forced to make concessions in geometric detail.

Using GPU-based tessellation, a game developer can send a compact geometric representation of an object or character, and the tessellator unit can produce the correct geometric complexity for the specific scene. We'll now go into greater detail discussing the characteristics and benefits of tessellation in combination with displacement mapping.

Consider the character below. On the left we see the quad mesh used to model the general outline of the figure. This representation is quite compact, even when compared to typical game assets. The image of the character in the middle was created by finely tessellating the description on the left. The result is a very smooth appearance, free of any of the faceting that resulted from limited geometry. Unfortunately this character, while smooth, is no more detailed than the coarse mesh. The image on the right was created by applying a displacement map to the smoothly tessellated character in the middle. This character has a richness of geometric detail that you might associate with film production.

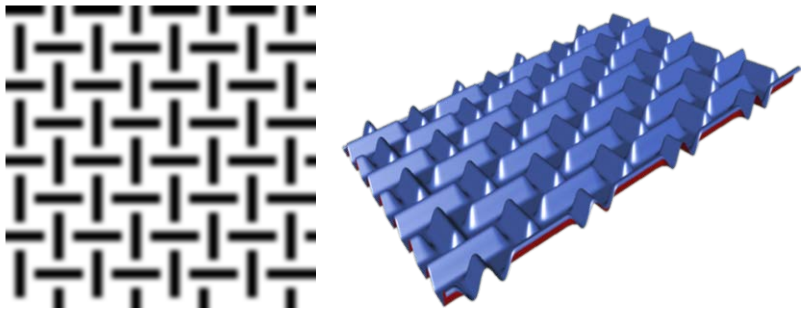


The "Imp" © Kenneth Scott, id Software 2008

Benefits of Tessellation with Displacement Mapping

There are a number of benefits to using tessellation with displacement mapping. The representation is compact, scalable and leads to efficient storage and computation. The compactness of the description means that the memory footprint is small and little bandwidth is consumed pulling the constituent vertices on to the GPU. Because animation is performed on the compact description, more compute intensive, sophisticated, realistic movement is possible. The on-demand synthesis of triangles creates the ability to match the geometric complexity and the number of triangles generated to the situation for the specific character as it appears in a given frame.

This ability to control geometric level of detail (LOD) is very powerful. Because it is on-demand and the data is all kept on-chip, precious memory bandwidth is preserved. Also, because one model may produce many LODs, the same game assets may be used on a variety of platforms, from a modest notebook to a Quad SLI system for example.



When a displacement map (left) is applied to a flat surface, the resulting surface (right) expresses the height information encoded in the displacement map.

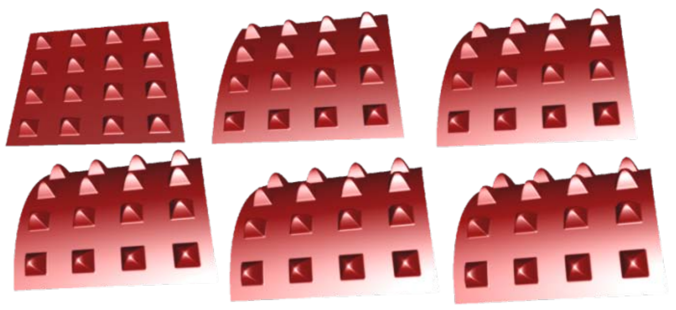
The character can also be tailored to how it appears in the scene, if it is small then it gets little geometry, if it is close to the screen it is rendered with maximum detail. Additionally, scalable assets mean that developers may be able to use the same models on multiple generations of games and future GPUs where performance increases enable even greater detail than was possible when initially deployed in a game. Complexity can be adjusted dynamically to target a given frame rate. Finally, models that are rendered using tessellation with displacement mapping much more closely resemble those used natively in the tools used by artists, freeing artists from the overhead work of creating models with different LODs.

Displacement mapping is a very powerful modeling and rendering technique. A displacement map is a texture that expresses height information. When applied to a model, the displacement map is used to alter the relative position of vertices in the model. Displacement mapping allows complex geometry to be stored in a compact map. In this way, displacement maps can be regarded as a form of geometry compression.

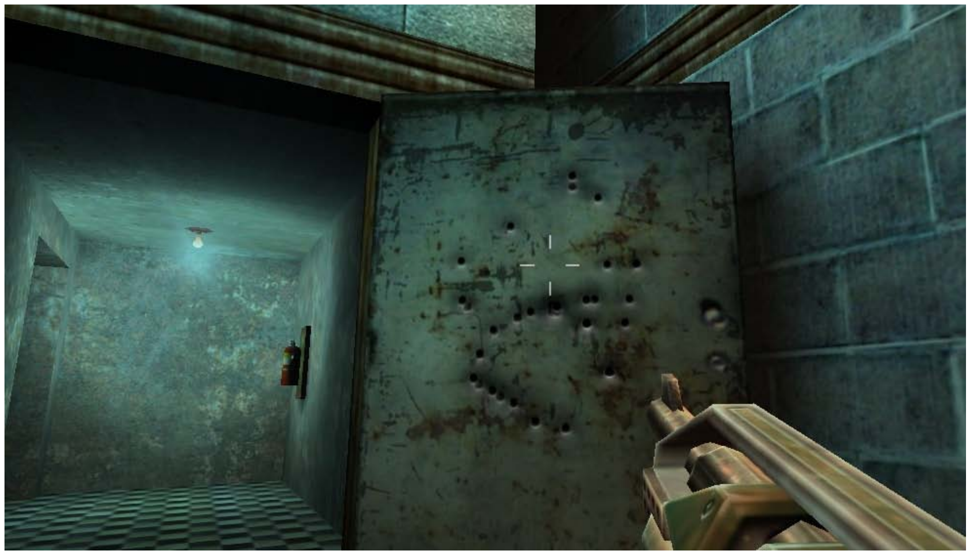
Unlike emboss maps, normal maps, and parallax maps which merely alter the appearance of pixels, displacement maps alter the position of vertices. This enables self occlusion, accurate shadows, and robust behavior at the edges of silhouettes.

Displacement mapping is complementary to existing bump mapping techniques. For example, displacement maps can be used to define major surface features while finer grained techniques such as normal mapping are used for low level details such as scratches and moles.

In addition to being a simple way to create complex geometry, displacement mapped geometry also behaves naturally when animated. Consider the simple example to the right—the blunt spikes follow the base shape as it is bent. Displacement mapped characters behave similarly. Consider the Imp character on the preceding page. It is animated by manipulating the coarse control hull (left). The displacement mapped character (right) naturally follows the animation of the underlying surface.


Displaced surfaces behave naturally with animation.

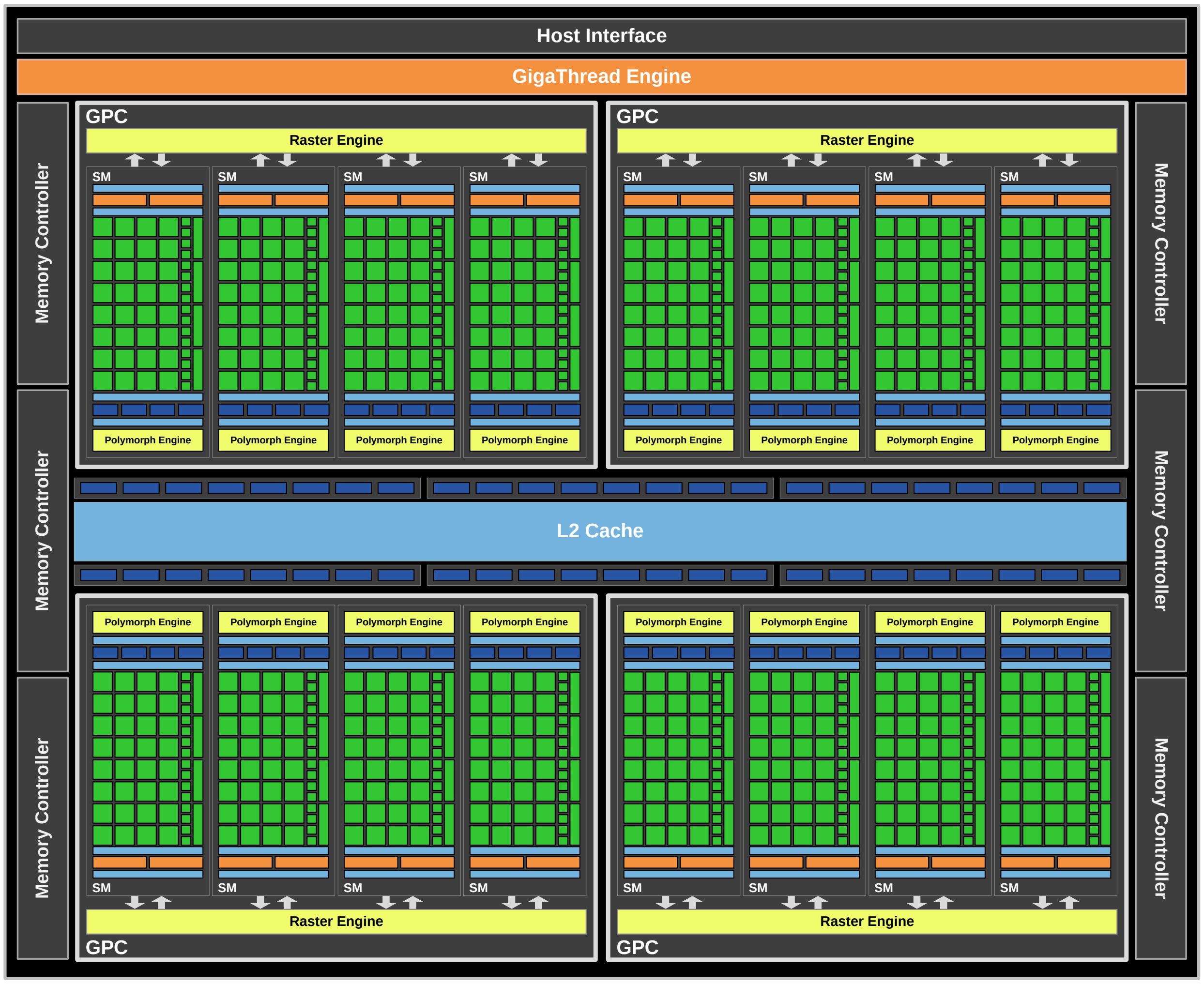
Finally, one of the most interesting aspects of displacement maps is the ability to easily modify them during game play. In today’s games, spraying a metal door with bullets leaves a trail of bullet “decals”, but the shape of the door will not be altered. With displacement mapping, the same decal textures can be used to alter the displacement map, allowing a player to deform both the appearance and underlying structure of game objects.



Today's games employ decals to depict altered surfaces. With displacement mapping, bullet decals can be used to alter the underlying geometry of objects.

GF100 Architecture In-Depth

GF100 GPUs are based on a scalable array of Graphics Processing Clusters (GPCs), Streaming Multiprocessors (SMs), and memory controllers. A full GF100 implements four GPCs, sixteen SMs and six memory controllers. We expect to launch GF100 products with different configurations of GPCs, SMs, and memory controllers to address different price points. For the purpose of this whitepaper, we will focus on the full GF100 GPU.



GF100 block diagram showing the Host Interface, the GigaThread Engine, four GPCs, six Memory Controllers, six ROP partitions, and a 768 KB L2 cache. Each GPC contains four PolyMorph engines. The ROP partitions are immediately adjacent to the L2 cache.

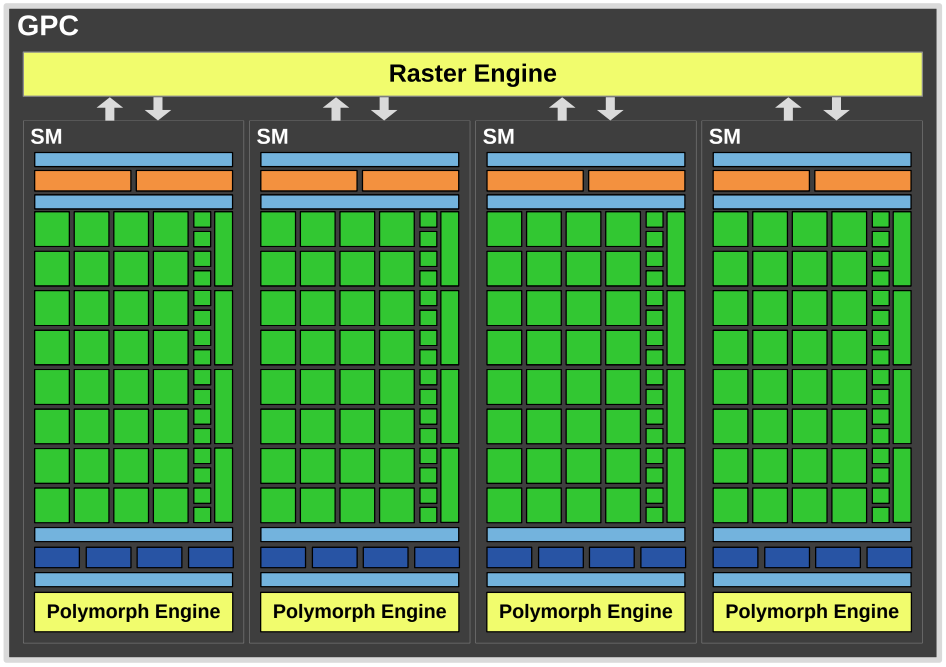
CPU commands are read by the GPU via the Host Interface. The GigaThread Engine fetches the specified data from system memory and copies them to the framebuffer. GF100 implements six 64-bit GDDR5 memory controllers (384-bit total) to facilitate high bandwidth access to the framebuffer. The GigaThread Engine then creates and dispatches thread blocks to various SMs. Individual SMs in turn schedules warps (groups of 32 threads) to CUDA cores and other execution units. The GigaThread Engine also redistributes work to the SMs when work expansion occurs in the graphics pipeline, such as after the tessellation and rasterization stages.

GF100 implements 512 CUDA cores, organized as 16 SMs of 32 cores each. Each SM is a highly parallel multiprocessor supporting up to 48 warps at any given time. Each CUDA core is a unified processor core that executes vertex, pixel, geometry, and compute kernels. A unified L2 cache architecture services load, store, and texture operations.

GF100 has 48 ROP units for pixel blending, antialiasing, and atomic memory operations. The ROP units are organized in six groups of eight. Each group is serviced by a 64-bit memory controller. The memory controller, L2 cache, and ROP group are closely coupled—scaling one unit automatically scales the others.

GPC Architecture

GF100's graphics architecture is built from a number of hardware blocks called Graphics Processing Clusters (GPCs). A GPC contains a Raster Engine and up to four SMs.



Graphics Processing Cluster (GPC)

The GPC is GF100's dominant high-level hardware block. It features two key innovations—a scalable Raster Engine for triangle setup, rasterization, and Z-cull, and a scalable PolyMorph Engine for vertex attribute fetch and tessellation. The Raster Engine resides in the GPC, whereas the PolyMorph Engine resides in the SM.

As its name indicates, the GPC encapsulates all key graphics processing units. It represents a balanced set of vertex, geometry, raster, texture, and pixel processing resources. With the exception of ROP functions, a GPC can be thought of as a self contained GPU, and a GF100 has four GPCs!

On prior NVIDIA GPUs, SMs and Texture Units were grouped together in hardware blocks called Texture Processing Clusters (TPCs). On GF100, each SM has four dedicated Texture Units, eliminating the need for TPCs. For simplicity, we will only refer to the SM going forward.

Parallel Geometry Processing

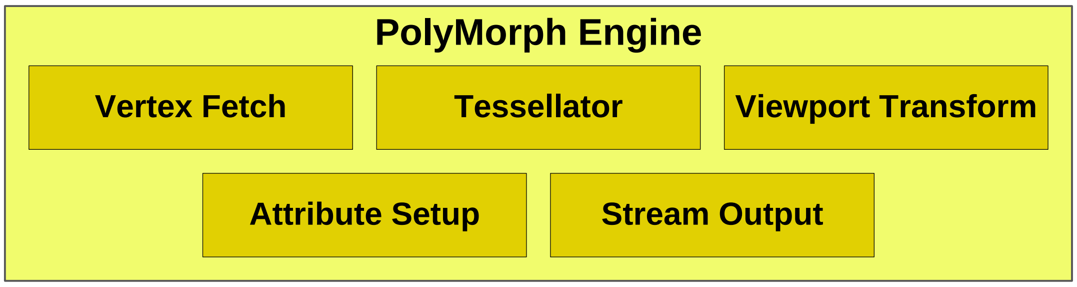
Previous GPU designs have used a single monolithic front-end to fetch, assemble, and rasterize triangles. This fixed pipeline provided a fixed amount of performance to an arbitrary number of parallel execution cores. As applications differ in their workload, this pipeline was often bottlenecked or underutilized. The difficulty of parallelizing rasterization while maintaining API order also discouraged major innovations in this area. While the single front-end design has worked well in past GPU designs, it became a major roadblock as the need for geometric complexity increased.

The use of tessellation fundamentally changes the GPU's graphics workload balance. With tessellation, the triangle density of a given frame can increase by multiple orders of magnitude, putting enormous strain on serial resources such as the setup and rasterization units. To sustain high tessellation performance, it is necessary to rebalance the graphics pipeline.

To facilitate high triangle rates, we designed a scalable geometry engine called the PolyMorph Engine. Each of the 16 PolyMorph engines has its own dedicated vertex fetch unit and tessellator, greatly expanding geometry performance. In conjunction, we also designed four parallel Raster Engines, allowing up to four triangles to be setup per clock. Together, they enable breakthrough triangle fetch, tessellation, and rasterization performance.

The PolyMorph Engine

The PolyMorph Engine has five stages: Vertex Fetch, Tessellation, Viewport Transform, Attribute Setup, and Stream Output. Results calculated in each stage are passed to an SM. The SM executes the game's shader, returning the results to the next stage in the PolyMorph Engine. After all stages are complete, the results are forwarded to the Raster Engines.



The first stage begins by fetching vertices from a global vertex buffer. Fetched vertices are sent to the SM for vertex shading and hull shading. In these two stages vertices are transformed from object space to world space, and parameters required for tessellation (such as tessellation factor) are calculated. The tessellation factors (or LODs) are sent to the Tessellator.

In the second stage, the PolyMorph Engine reads the tessellation factors. The Tessellator dices the patch (a smooth surface defined by a mesh of control points) and outputs a mesh of vertices. The mesh is defined by patch (u,v) values, and how they are connected to form a mesh.

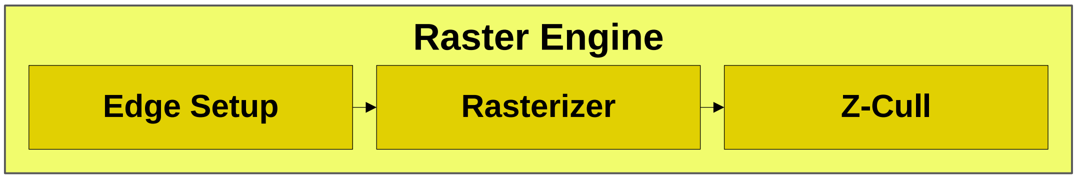
The new vertices are sent to the SM where the Domain Shader and Geometry Shader are executed. The Domain Shader calculates the final position of each vertex based on input from the Hull Shader and Tessellator. At this stage, a displacement map is usually applied to add detailed features to the patch. The Geometry Shader conducts any post processing, adding and removing vertices and primitives where needed. The results are sent back to the PolyMorph Engine for the final pass.

In the third stage, the PolyMorph Engine performs viewport transformation and perspective correction. Attribute setup follows, transforming post-viewport vertex attributes into plane equations for efficient shader evaluation. Finally, vertices are optionally "streamed out" to memory making them available for additional processing.

On prior architectures, fixed function operations were performed with a single pipeline. On GF100, both fixed function and programmable operations are parallelized, resulting in vastly improved performance.

Raster Engine

After primitives are processed by the PolyMorph Engine, they are sent to the Raster Engines. To achieve high triangle throughput, GF100 uses four Raster Engines in parallel.



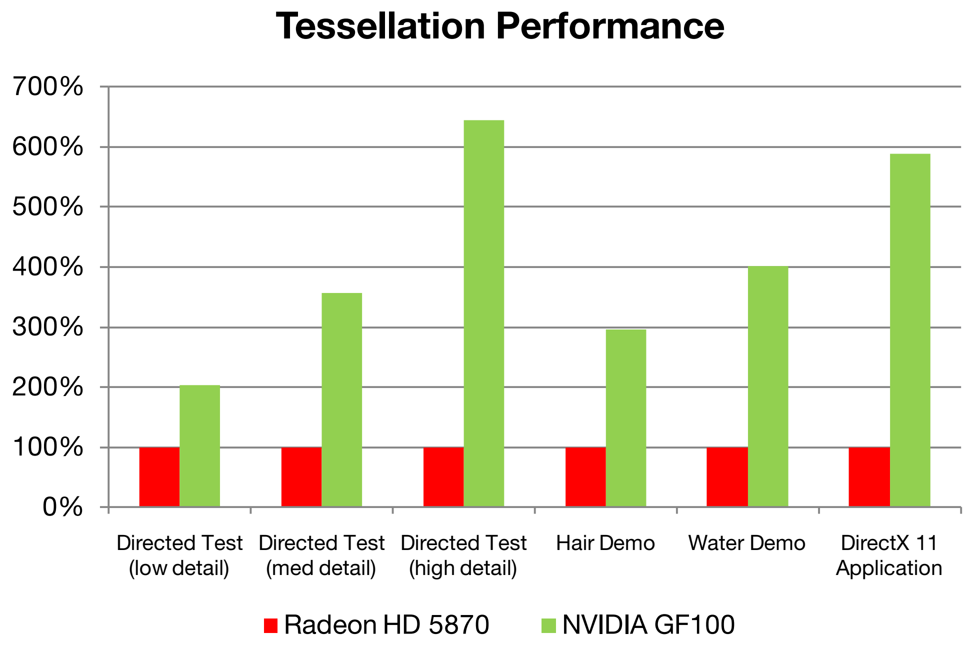
The Raster Engine is composed of three pipeline stages. In the edge setup stage, vertex positions are fetched and triangle edge equations are computed. Triangles not facing the screen are removed via back face culling. Each edge setup unit processes up to one point, line, or triangle per clock.

The Rasterizer takes the edge equations for each primitive and computes pixel coverage. If antialiasing is enabled, coverage is performed for each multisample and coverage sample. Each Rasterizer outputs eight pixels per clock for a total of 32 rasterized pixels per clock across the chip.

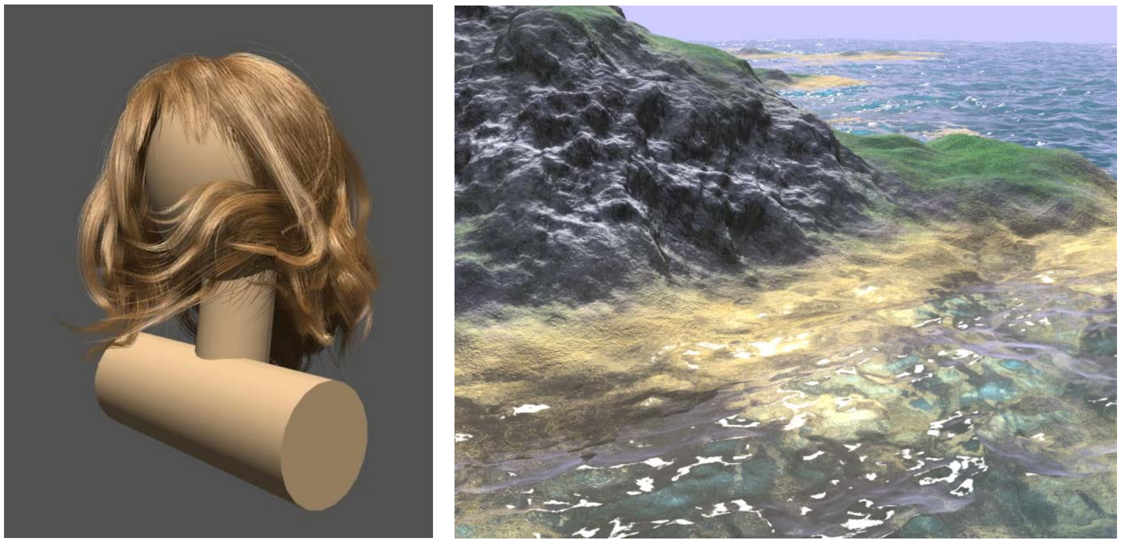
Pixels produced by the rasterizer are sent to the Z-cull unit. The Z-cull unit takes a pixel tile and compares the depth of pixels in the tile with existing pixels in the framebuffer. Pixel tiles that lie entirely behind framebuffer pixels are culled from the pipeline, eliminating the need for further pixel shading work.

Recap of the GPC Architecture

The GPC architecture is a significant breakthrough for the geometry pipeline. Tessellation requires new levels of triangle and rasterization performance. The PolyMorph Engine dramatically increases triangle, tessellation, and Stream Out performance. Four parallel Raster Engines provide sustained throughout in triangle setup and rasterization. By having a dedicated tessellator for each SM, and a Raster Engine for each GPC, GF100 delivers up to 8× the geometry performance of GT200.



The left three bars in the chart show tessellated geometry performance for three directed tests that focus exclusively on tessellation performance. As geometric complexity is increased, GF100's performance lead over the competition increases. The Hair and Water demos include both shading and compute operations in addition to geometry processing. The rightmost bar shows the performance of a tessellation state bucket (a set of draw calls from a frame) from a DirectX 11 application.

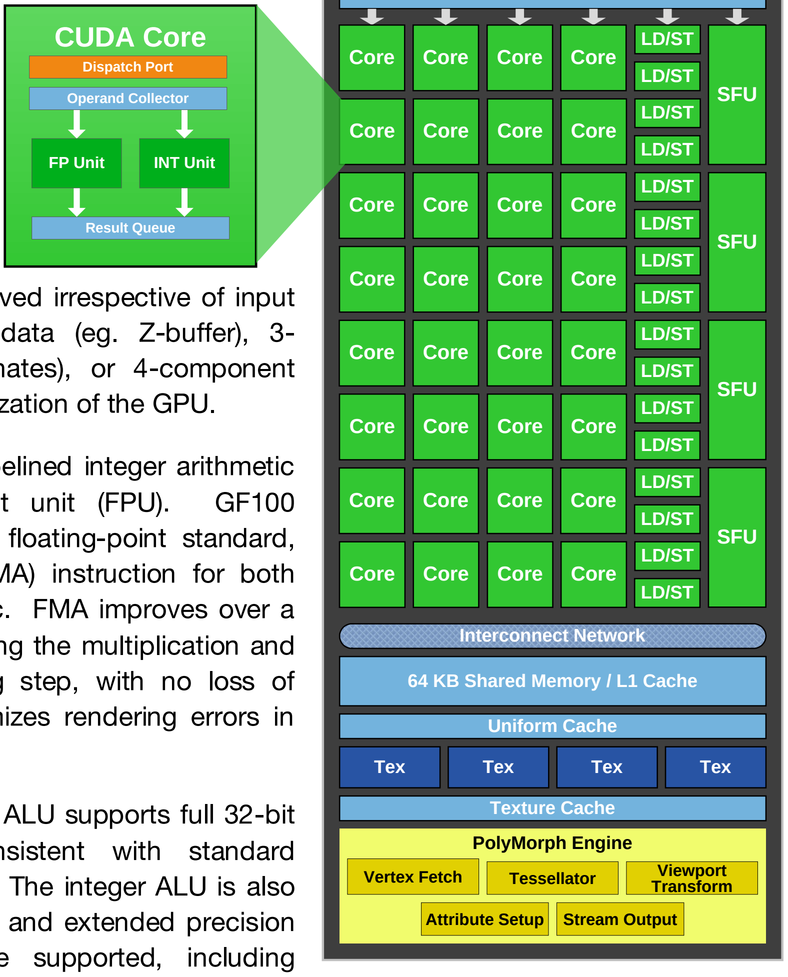


The Hair demo (left) uses tessellation, geometry shaders, and physical simulations. The Water demo (right) uses tessellation in large environments.

Third Generation Streaming Multiprocessor

The third generation SM introduces several architectural innovations that make it not only the most powerful SM yet built, but also the most programmable and efficient.

512 High Performance CUDA cores



Each SM features 32 CUDA processors—a fourfold increase over prior SM designs. GF100's CUDA cores are designed for maximum performance and efficiency across all shader workloads. By employing a scalar architecture, full performance is achieved irrespective of input vector size. Operations on scalar data (eg. Z-buffer), 3-component vectors (eg. 3D coordinates), or 4-component vectors (eg. RGBA color) attain full utilization of the GPU.

Each CUDA processor has a fully pipelined integer arithmetic logic unit (ALU) and floating point unit (FPU). GF100 implements the new IEEE 754-2008 floating-point standard, providing the fused multiply-add (FMA) instruction for both single and double precision arithmetic. FMA improves over a multiply-add (MAD) instruction by doing the multiplication and addition with a single final rounding step, with no loss of precision in the addition. FMA minimizes rendering errors in closely overlapping triangles.

In GF100, the newly designed integer ALU supports full 32-bit precision for all instructions, consistent with standard programming language requirements. The integer ALU is also optimized to efficiently support 64-bit and extended precision operations. Various instructions are supported, including Boolean, shift, move, compare, convert, bit-field extract, bit-reverse insert, and population count.

Streaming Multiprocessor (SM)

16 Load/Store Units

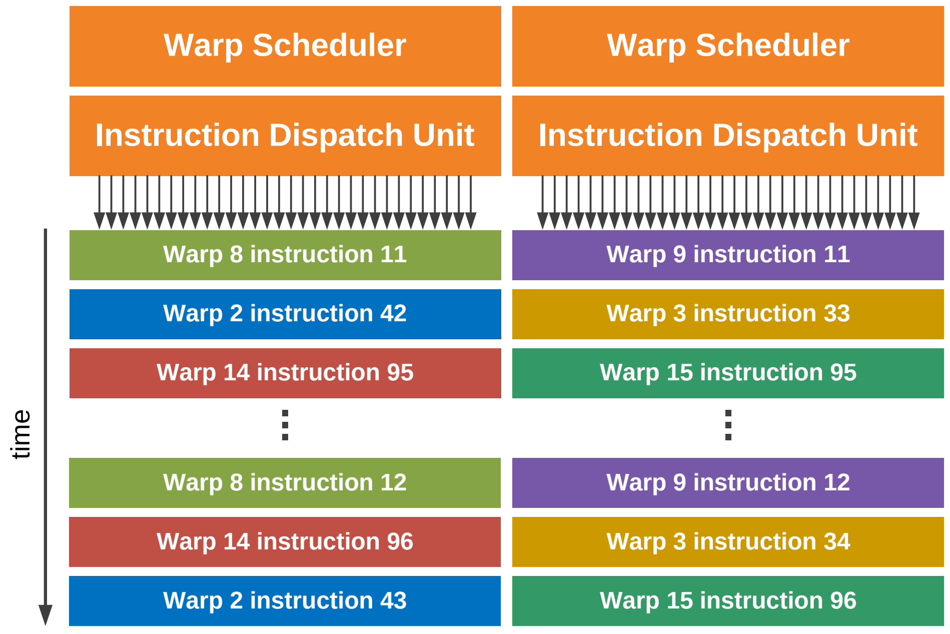
Each SM has 16 load/store units, allowing source and destination addresses to be calculated for sixteen threads per clock. Supporting units load and store the data at each address to cache or DRAM.

Four Special Function Units

Special Function Units (SFUs) execute transcendental instructions such as sin, cosine, reciprocal, and square root. Graphics interpolation instructions are also performed on the SFU. Each SFU executes one instruction per thread, per clock; a warp (32 threads) executes over eight clocks. The SFU pipeline is decoupled from the dispatch unit, allowing the dispatch unit to issue to other execution units while the SFU is occupied. Complex procedural shaders especially benefit from dedicated hardware for special functions.

Dual Warp Scheduler

The SM schedules threads in groups of 32 parallel threads called warps. Each SM features two warp schedulers and two instruction dispatch units, allowing two warps to be issued and executed concurrently. GF100's dual warp scheduler selects two warps, and issues one instruction from each warp to a group of sixteen cores, sixteen load/store units, or four SFUs. Because warps execute independently, GF100's scheduler does not need to check for dependencies from within the instruction stream. Using this elegant model of dual-issue, GF100 achieves near peak hardware performance.



Dual warp scheduling

Most instructions can be dual issued—two integer instructions, two floating instructions, or a mix of integer, floating point, load, store, and SFU instructions can be issued concurrently. Double precision instructions do not support dual dispatch with any other instruction.

Texture Units

Each SM has four texture units. Each texture unit computes a texture address and fetches four texture samples per clock. Results can be returned filtered or unfiltered. Bilinear, trilinear, and anisotropic filtering modes are supported.

The goal with GF100 was to improve delivered texture performance through improved efficiency. This was achieved by moving the texture units within the SM, improving the efficiency of the texture cache, and running both the texture units and texture cache at a higher clock speed.

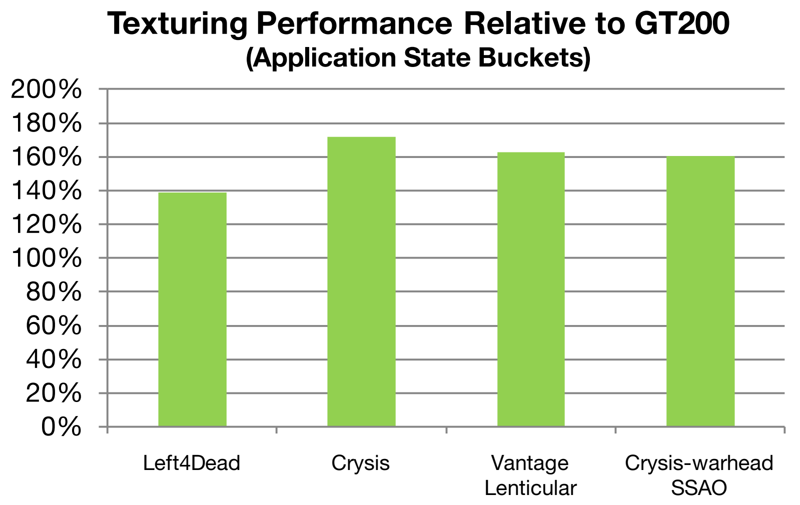
In the previous GT200 architecture, up to three SMs shared one texture engine containing eight texture filtering units. In the GF100 architecture, each SM has its own dedicated texture units and a dedicated texture cache. Also, the internal architecture of the texture units has been significantly enhanced. The net effect is a significant improvement in the delivered texture performance in real-world use cases such as shadow mapping and screen space ambient occlusion.

GF100's dedicated L1 texture cache has been redesigned for greater efficiency. Further, by having a unified L2 cache, the maximum cache size available for texture is three times higher than GT200, improving hit rates in texture heavy shaders.

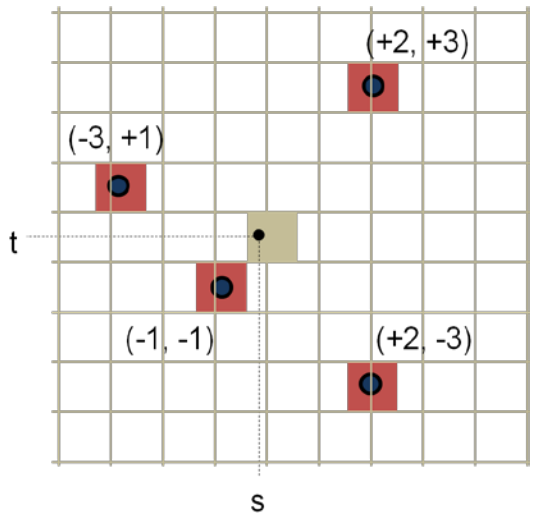
The texture units on previous architectures operated at the core clock of the GPU. On GF100, the texture units run at a higher clock, leading to improved texturing performance for the same number of units.

GF100's texture units also add support for DirectX 11's BC6H and BC7 texture compression formats, reducing the memory footprint of HDR textures and render targets.

The texture units also support jittered sampling through DirectX 11's four-offset Gather4 feature, allowing four texels to be fetched from a 64 × 64 pixel grid with a single texture instruction. GF100 implements DirectX 11 four-offset Gather4 in hardware, greatly accelerating shadow mapping, ambient occlusion, and post processing algorithms. With jittered sampling, games can implement smoother soft shadows or custom texture filters efficiently.

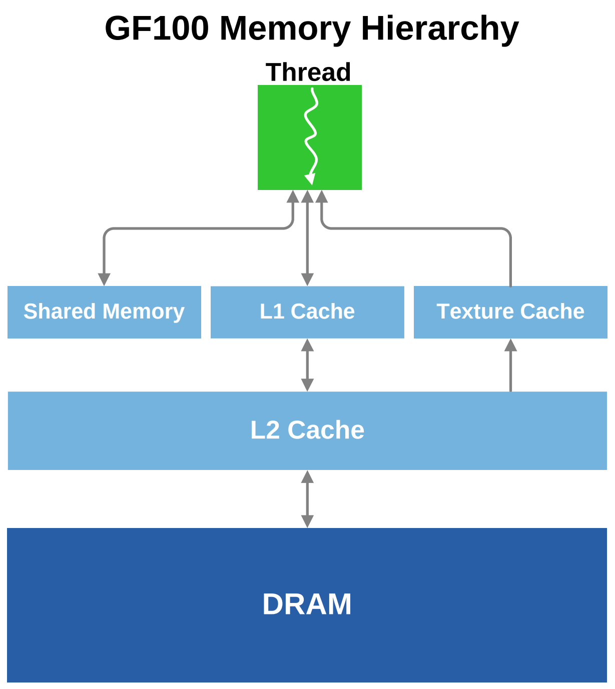


While GT200 has more texture units than GF100, GF100 delivers higher real world performance thanks to improved efficiency.



The soft shadows in 3DMark 2006 (left) are implemented through jittered sampling of a texture map (right). GF100 implements jittered sampling in hardware, delivering up to 2x the performance of GT200.

64 KB Configurable Shared Memory and L1 Cache



Shared memory—a fast, programmable on-chip memory, is one of the key architectural innovations of the first generation CUDA architecture. By facilitating inter-thread communication, shared memory enabled a broad range of applications to run efficiently on the GPU. Shared memory has since been adopted by all major GPU computing standards and competing architectures.

Recognizing the crucial role played by shared memory and the importance of maintaining data locality, we once again extended the GPU memory model. GF100 incorporates a dedicated L1 cache per SM.

The L1 cache works as a counterpart to shared memory—while shared memory improves memory access for algorithms with well defined memory access, the L1 cache improves memory access for irregular algorithms where data addressees are not known beforehand.

On GF100 GPUs, each SM has 64 KB of on-chip memory that can be configured as 48 KB of Shared memory with 16 KB of L1 cache, or as 16 KB of Shared memory with 48 KB of L1 cache.

For graphics programs, GF100 makes use of the 16 KB L1 cache configuration. The L1 cache acts as a buffer for register spills, allowing graceful performance scaling with register usage. For compute programs, the L1 cache and shared memory enables threads within the same thread block to cooperate, facilitates extensive reuse of on-chip data, and reduces off-chip traffic. Shared memory is a key enabler for many high-performance CUDA applications.

L2 Cache

GF100 has a 768 KB unified L2 cache that services all load, store, and texture requests. The L2 provides efficient, high speed data sharing across the GPU. Algorithms for which data addresses are not known beforehand, such as physics solvers, ray tracing, and sparse data structures especially benefit from a hardware cache. Post processing filters that require multiple SMs to read the same data require fewer trips to memory, improving bandwidth efficiency.

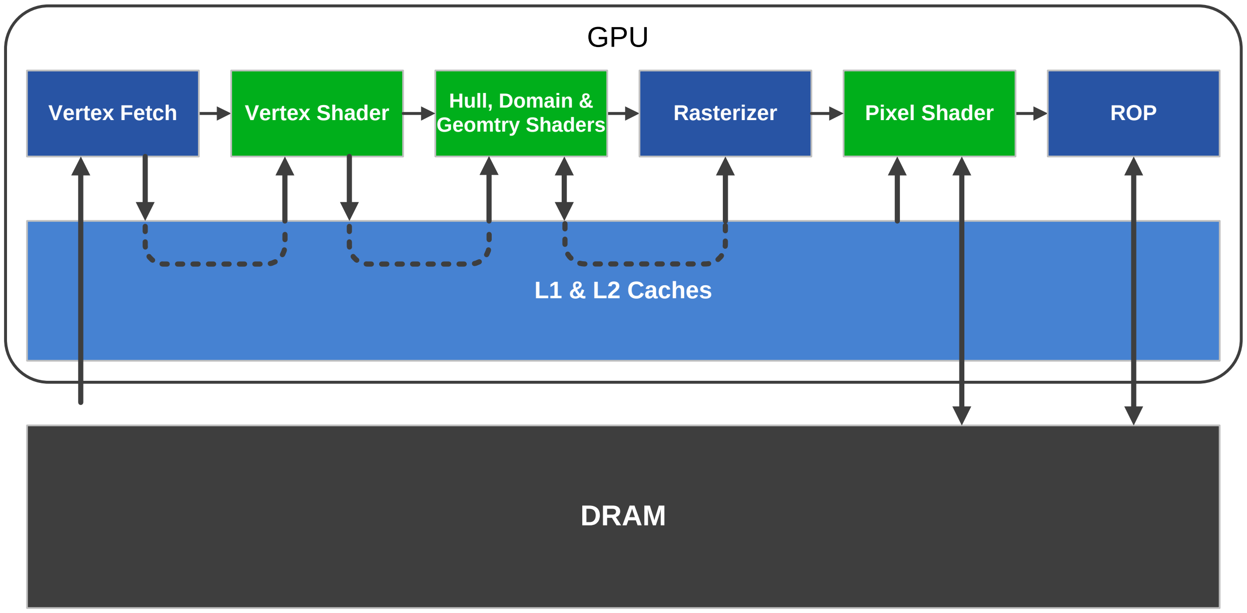
A unified cache is more efficient than separate caches. In a non-unified cache design, if one cache is oversubscribed, it cannot use the unmapped portions of other caches. Cache utilization will always be less than the theoretical peak. GF100's unified L2 cache dynamically load balances between different requests, allowing full utilization of the cache. The L2 cache replaces the L2 texture cache, ROP cache, and on-chip FIFOs on prior GPUs.

A unified cache also ensures memory access instructions arrive in program order. Where read and write paths are separate (such as a read only texture path and a write only ROP path), read after write hazards may occur. A unified read/write path ensures program correctness, and is a key feature that allows NVIDIA GPUs to support generic C/C++ programs.

	GT200	GF100	Benefits
L1 Texture Cache (per quad)	12KB	12 KB	Fast texture filtering
Dedicated L1 LD/ST Cache	None	16 or 48 KB	Efficient physics and ray tracing
Total Shared Memory	16 KB	16 or 48 KB	More data reuse among threads
L2 Cache	256 KB (Texture read only)	768 KB (all clients read/write)	Greater texture coverage, robust compute performance

GT200 and GF100 cache architectures compared

GF100 L2 cache is read/write and fully coherent compared to GT200 L2 cache which is read-only. Evicting data out of L2 is handled by a priority algorithm that includes various checks to help ensure needed data stays resident in the cache.



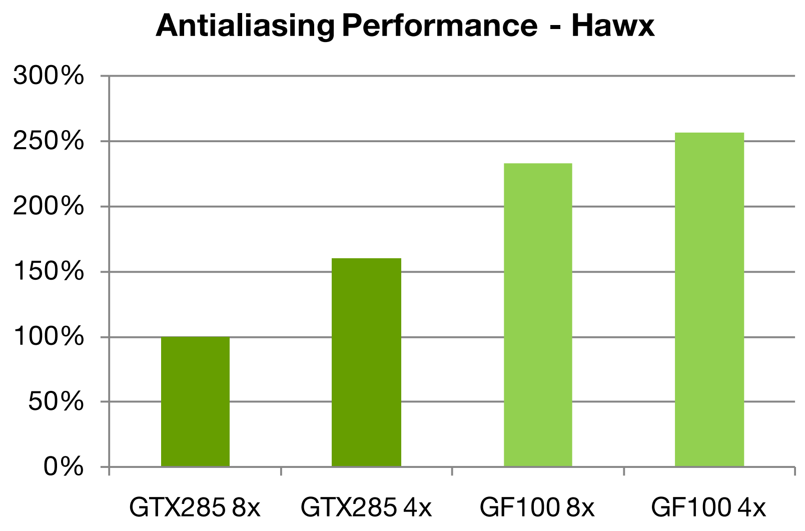
GF100's cache architecture enables efficient communication between pipeline stages and reduces off-chip memory traffic.

New ROP Units with Improved Antialiasing

GF100's ROP subsystem has been redesigned for improved throughput and efficiency. One GF100 ROP partition contains eight ROP units, a twofold improvement over prior architectures. Each ROP unit can output a 32-bit integer pixel per clock, an FP16 pixel over two clocks, or an FP32 pixel over four clocks. Atomic instruction performance is also vastly improved—atomic operations to the same address execute up to 20 times faster than GT200, and operations to contiguous memory regions execute up to 7.5 times faster.

Performance for 8xMSAA is significantly increased on GF100 due in improvements in compression efficiency as well as additional ROP units that permit more effective rendering of smaller primitives that cannot be compressed. Increasing geometric realism in scenes increases the requirement for ROP units to perform well when compression is not active.

In the previous generation, performance drop in 8xMSAA modes varied significantly depending on the title; *Tom Clancy's HAWX* is one example of a game that showed low efficiency in 8xMSAA. In GF100 the 8xAA performance is much improved. In 4xAA mode, GF100 is 1.6× faster than GT200. Comparing in 8xAA mode, GF100 is 2.3× faster than GT200, and only 9% slower than the GF100 4xAA mode.

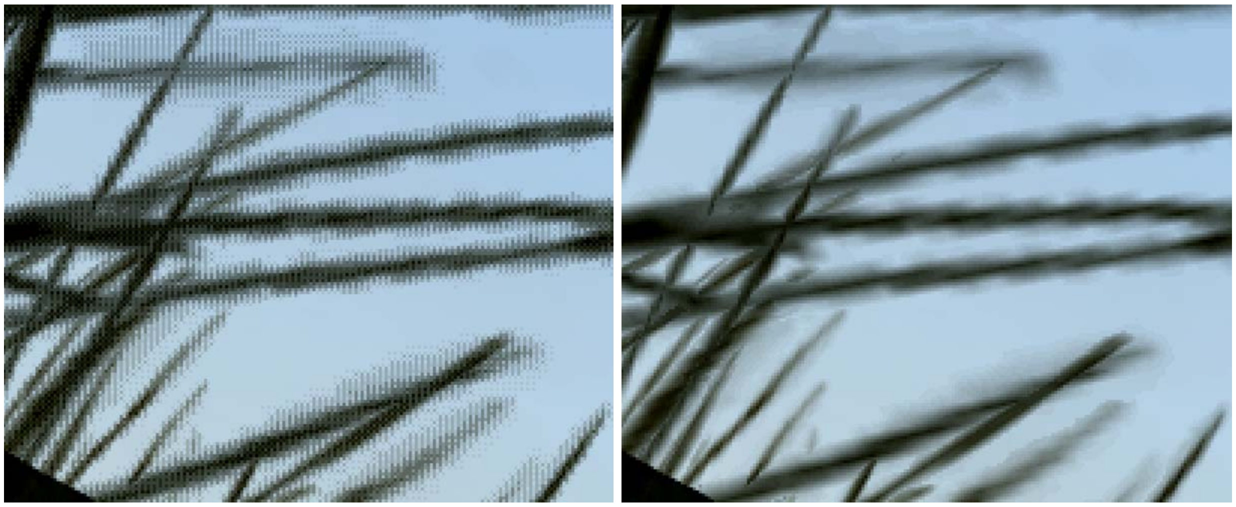


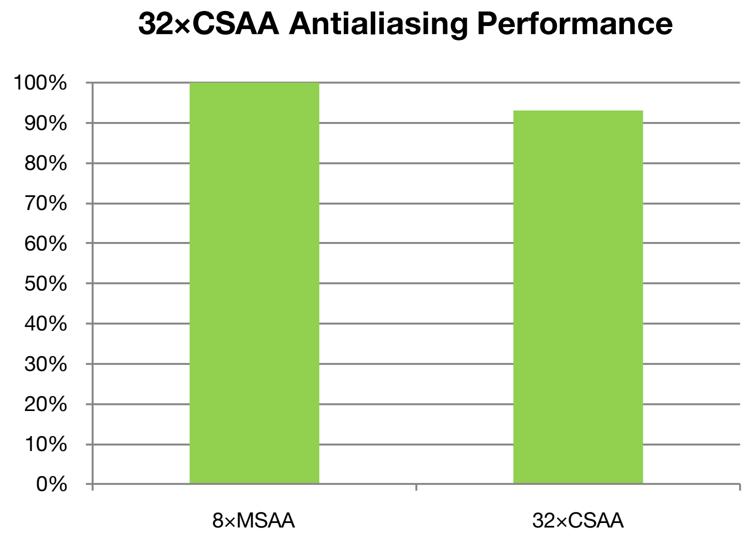
Antialiasing performance, especially 8xMSAA, is significantly improved on GF100.

GF100 also features a new 32x Coverage Sampling Antialiasing (CSAA) mode designed to provide the highest image quality and improve the level of perceived geometric realism in current games using alpha-to-coverage.

Current games are constrained by the limitations of API and GPU horsepower in the amount of geometry they can render. Foliage is a particular challenge. A common technique for foliage is to create an alpha textured billboard containing many leaves, using alpha-to-coverage to eliminate the gaps between the leaves. The quality of the edge is determined by the number of coverage samples. In cases with only four or even eight coverage samples available, aliasing and banding become very obvious, especially when the texture is close to the screen. With 32x CSAA, the GPU has 32 coverage samples available, minimizing banding effects and improving overall image quality.

Transparency Multisampling (TMAA) also benefits from CSAA. TMAA benefits DirectX 9 games that are unable to use alpha-to-coverage directly because it is not exposed in the DirectX 9 API. Instead they use a technique called "alpha test" which produces hard edges for transparent textures. TMAA automatically converts old shader code in the DirectX 9 applications to use alpha-to-coverage, which combined with CSAA, produces greatly improved image quality.


The image on the left shows the TMAA using 16xQ antialiasing (8 multisamples, 8 coverage sample) on older GPUs. The image on the right shows TMAA using 32x antialiasing (8 multisamples, 24 coverage samples) on GF100. Because coverage samples are used as part of GF100's TMAA evaluation, much smoother gradients are produced.



Because coverage samples have low memory storage requirements, 32xCSAA performance is largely comparable to 8xMSAA. An average across a range of games show 32xCSAA performing only 7% slower than 8xMSAA.

Compute Architecture for Graphics

The vast improvements in per-pixel realism in recent years were made possible by programmable shaders. Going forward, programmability will continue to be of overriding importance in allowing developers to create next generation visual effects.

Computer graphics is a set of diverse problems with numerous approaches. Rasterization, ray tracing, and Reyes are well recognized general rendering algorithms. Within each style of rendering, different solutions exist for various sub-problems. Up until now, the GPU has been designed solely for rasterization. As developers continue to search for novel ways to improve their graphics engines, the GPU will need to excel at a diverse and growing set of graphics algorithms. Since these algorithms are executed via general compute APIs, a robust compute architecture is fundamental to a GPU's graphical capabilities. In essence, one can think of compute as the new programmable shader.

G80 was NVIDIA's first generation GPU computing architecture. Its design reflected the desire to extend the GPU's capabilities to solve HPC style problems. For example, one of G80's key innovations, shared memory, was instrumental in accelerating matrix multiplication, the basis of many math and physics algorithms.

GF100's compute architecture is designed to address a wider range of algorithms and to facilitate more pervasive use of the GPU for solving parallel problems. Many algorithms, such as ray tracing, physics, and AI, cannot exploit shared memory—program memory locality is only revealed at runtime. GF100's cache architecture was designed with these problems in mind. With up to 48 KB of L1 cache per SM and a global L2 cache, threads that access the same memory locations at runtime automatically run faster, irrespective of the choice of algorithm.

Another area of improvement in GF100's compute architecture for gaming is in scheduling. G80 and GT200 executed large kernels one at a time with relatively slow context switching. Since HPC applications employ large data sets and are insensitive to latency, this model worked relatively well. In gaming applications, no single kernel dominates, but various small kernels (cloth, fluid, and rigid bodies for example) are executed. On GF100, these kernels execute in parallel, enabling maximum utilization of CUDA cores.

In games that make use of compute, context switches occur at every frame, making their performance highly critical to responsive framerates. GF100 reduces context switch time down to about 20 microseconds, making it possible to perform fine-grained context switching between multiple kernels per frame. For example, a game may use DirectX 11 to render the scene, switch to CUDA for selective ray tracing, call a DirectCompute kernel for post processing, and perform fluid simulations using PhysX.

As developers make more general use of the GPU, better language and debugging support becomes crucial. GF100 is the first GPU to offer full C++ support, the language of choice among game developers. To ease the transition to GPU programming, we've also developed Nexus—a Microsoft Visual Studio programming environment for the GPU. Together with new hardware features that provide better debugging support, developers will be able enjoy CPU-class application development on the GPU.

Next Generation Effects using GPU Computing

Because compute algorithms are general in nature, they can be directly applied to a large variety of visual computing and simulation algorithms. Some examples game developers are looking into for their upcoming games include:

- **Novel rendering algorithms**
    - Ray tracing for accurate reflections and refractions
    - Reyes for detailed displacement mapping and high quality antialiasing
    - Voxel rendering for simulation of volumetric data
- **Image processing algorithms**
    - Custom depth of field kernels with accurate out of focus highlights (bokeh)
    - Histograms for advanced HDR rendering
    - Custom filters for advanced blurring and sharpened effects
- **Physical simulations**
    - Smoothed partical hydrodynamics for advanced fluid simulation
    - Turbluance for detailed smoke and fluid effects
    - GPU Rigid bodies for pervasive use of physical objects
- **AI pathfinding algorithms for greater number of characters in-game**

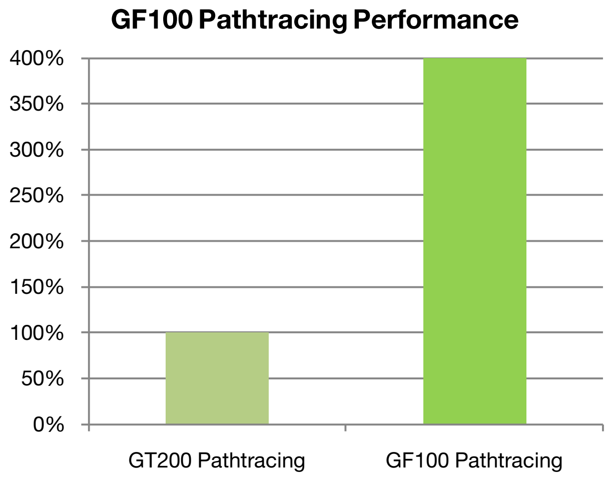
In the following section, we look at two examples in depth: **ray tracing**, and **smoothed particle hydrodynamics**.

Ray tracing

Ray tracing is seen by many as the future of graphics, either by itself or in conjunction with rasterization. With GF100, interactive ray tracing becomes possible for the first time on a standard PC.

Ray tracing has typically been challenging to run efficiently on the GPU. Ray tracing operates recursively, whereas GPUs mostly operate iteratively. Rays have unpredictable directions, requiring lots of random memory access. GPUs typically access memory in linear blocks for efficiency.

GF100's compute architecture was built specifically with ray tracing in mind. GF100 is the first GPU to support recursion in hardware, enabling efficient ray tracing and a host of other graphics algorithms. GF100's L1 and L2 caches greatly improve ray tracing efficiency by improving performance for fine-grained memory accesses. The L1 cache enhances memory locality for neighboring rays, whereas the L2 cache amplifies bandwidth to the framebuffer.



GF100 excels not just at standard ray tracing, but also at advanced global illumination algorithms such as path tracing, which uses a much larger number of rays to collect ambient lighting information from the scene. Early evaluations of path tracing show GF100 performing up to four times faster than GT200.

To sustain performance, a game may use ray tracing selectively. For example, rasterization can be used to perform a first pass on the scene. Pixels that are identified as reflective may be further processed via ray tracing. This hybrid model of rendering enables fast performance with great image quality.

A Bugatti Veyron rendered with path tracing using NVIDIA OptiX technology. OptiX integrates easily with game engines, allowing racing games to leverage ray tracing for near-photorealistic glamour shots in gallery mode.

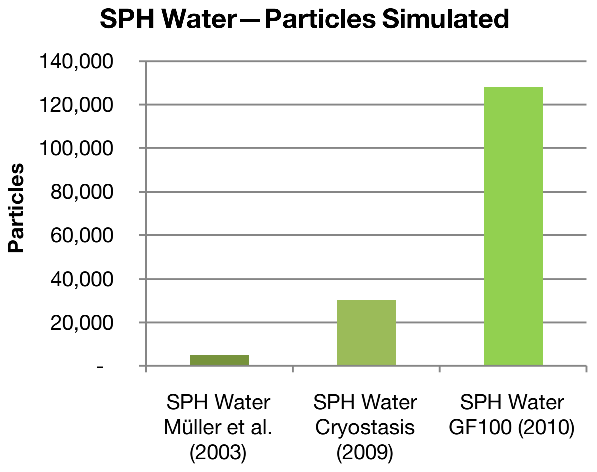
Smoothed Particle Hydrodynamics (SPH)

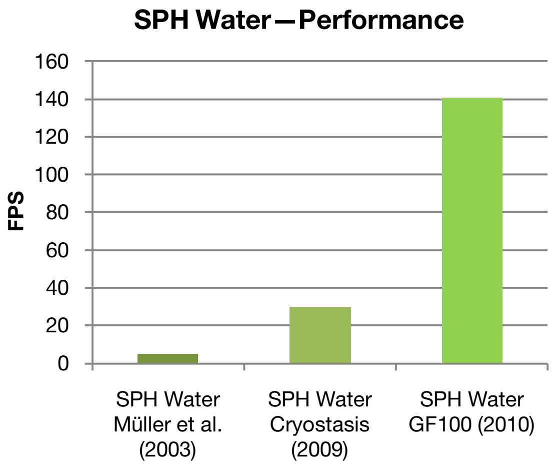
Realistic fluid simulations have long been used in film to create novel characters and dramatic effects. T-1000 in *Terminator 2: Judgment Day* was created with computer generated “liquid metal.” Simulation of vast bodies of water was crucial to achieving the climatic shots in *2012*. While game designers aspire to similar effects, the computational complexity of fluid simulations has prevented their use in realtime applications.

SPH based fluid simulation

In 2003, Müller et al[2]. adopted smoothed particle hydrodynamics (SPH), an astrophysics algorithm, for interactive fluid simulations. Their initial work demonstrated 5,000 SPH particles, enough to simulate a pouring glass of water at 5 frames per second. Muller’s SPH algorithm has since been integrated into the NVIDIA PhysX API. The first game that made use of PhysX SPH, *Cryostasis*, simulated 30,000 water particles at 30 frames per second on the GT200 architecture. Although a technological breakthrough for its time, the water in *Cryostasis* lacked the particle count to achieve fully convincing fluidity, and the high cost of interoperating with graphics limited real world performance.

GF100 is the first GPU to deliver the performance required for high fidelity fluid simulations, and together with an improved SPH solver, enables games designers to incorporate high quality SPH fluid throughout game environments. GF100 is able to simulate over 128,000 SPH particles per frame—sufficient to support large volumes of water and a range of fluid based effects. For example, SPH can be used to model rain, allowing the natural formation of splashes, puddles, and overflow. With different parameters, SPH can be used for blood, which has higher viscosity and different dripping characteristics. In both cases, fluid behavior is based on physical models, guaranteeing consistency and realism.





GF100 excels in simulating a large number of SPH particles with exceptional performance.

[2] M. Müller, D. Charypar, M. Gross, Particle-Based Fluid Simulation for Interactive Applications, in Proceedings of ACM SIGGRAPH Symposium on Computer Animation (SCA) 2003, pp 154-159

The SPH algorithm typically does not make use of shared memory, which has constrained performance on last generation architectures. GF100's robust cache architecture greatly reduces off-chip memory traffic, allowing large number of particles to be simulated without saturating memory bandwidth. Fast context switching further reduces simulation overhead. Thanks to these improvements, GF100 is able to perform SPH simulation (128,000 particles) together with graphics rendering at over 140 frames per second.

NVIDIA 3D Vision Surround

NVIDIA 3D Vision is a combination of high-tech wireless glasses and advanced software that automatically transforms games (400+ and counting) into full stereoscopic 3D.

Powered by GF100 GPUs in NVIDIA SLI configuration, the upcoming NVIDIA 3D Vision Surround technology takes 3D gaming to an entirely new level by delivering fully immersive IMAX 3D-like gaming across three monitors in full stereoscopic 3D.

NVIDIA Surround combines multiple displays to act as one larger display to allow a panoramic view of full-screen games or your desktop.

NVIDIA 3D Vision Surround uses up to 746 million pixels per second of rendering horsepower, or 3x greater than last generation's extreme gaming setup. With tessellation, compute shaders, and PhysX enabled, the demand on the GPU is tremendous. GF100 is architected to enable the highest performance on NVIDIA 3D Vision Surround. Its parallel tessellation and raster engines enable sustained performance in heavily tessellated scenes. And its powerful compute architecture with fast context switching makes compute operations as lightweight as possible.

Yesterday's Extreme Gaming requires 2560 x 1600 x 60 = 245 Million pixels/sec of rendering horsepower.

Next Generation 3D Vision Surround Gaming requires 1920 x 1080 x 120 x 3 = 746 Million pixels/second

GF100 supports 3D Vision Surround when two or more GPUs are paired in an SLI configuration. 3D Vision Surround will be supported across three of the same 3D Vision capable LCDs or projectors at resolutions up to 1920x1080. For those not ready to jump into stereoscopic gaming, NVIDIA Surround will also be supported in non-stereoscopic 3D at resolutions up to 2560 x 1600 across displays that share a common resolution.

Bezel Correction

NVIDIA 3D Vision Surround includes controls that allow for the adjustment of the displays to compensate for monitor bezel gaps, creating a more realistic view of full-screen games. With bezel correction, part of the game view is hidden behind the display bezel so that the bezel appears to be part of the game. This produces a more continuous image across the displays and provides a more realistic experience. It is similar to looking through a cockpit window where the window frames block your view.

Conclusion

PC gaming has been NVIDIA's core passion for sixteen years. With GF100, we continue this commitment to the advancement of 3D graphics and PC gaming.

With up to sixteen tessellation engines and four raster engines, GF100 elevates geometric realism to new heights. Tessellation and displacement mapping—the key techniques employed in films, become a reality for the first time in PC gaming. Characters with life-like detail, game environments with unprecedented complexity—that's what GF100 promises to gamers.

In the pursuit of computational graphics, we have been equally ambitious. GF100 is the world's first and only GPU to support C/C++, recursion, and cached read and writes. Together these features enable game developers to solve the hardest problems of graphics, including ray tracing, order independent transparency, and physics simulations. Once gamers experience the fidelity and consistency brought by computational graphics, the representational effects of the past will seem as aged as fixed function graphics.

Behind all our efforts is the belief that PC gaming is unique. Today's games are multiplatform by design, but it is on the PC that they reach their fullest expression. With GF100, we hope to continue to push the boundaries of PC gaming, and make it the world's most powerful and dynamic gaming platform.

With the graphics and compute horsepower of GF100, we've extended NVIDIA 3D Vision to three monitors, enabling the most immersive gaming experience possible.

With its amazing performance, groundbreaking geometry engine, and world class architecture for computational graphics, GF100 represents a landmark for 3D graphics and PC gaming.