NVIDIA®

Whitepaper

# Bringing High-End Graphics to Handheld Devices

## Table of Contents

## Executive Summary

While the convergence of mobile phone and computing technology has been building for years, we are now starting to experience exciting and disruptive new applications and usage models as the next generation of mobile devices begin to deliver on the promise of a truly mobile computing experience.

Consumer expectations for new mobile devices are very high. These new devices are more immersive and interactive than ever before, which makes the user hyper-aware of device performance. Consumers are demanding highly responsive user interfaces, uncompromised web browsing performance, visually compelling online and offline gaming experiences, and access to all of their content, while extending battery life beyond levels found in current mobile phones.

The **NVIDIA® Tegra™ 2** mobile processor is a multi-core system-on-a-chip (SoC) that is specifically designed to deliver the performance required for current and future mobile use cases. Implementing a powerful pipelined vertex and pixel processing architecture, the Tegra 2 ultra low power **GeForce™ GPU** core includes several features that reduce power consumption and increase graphics quality.

The GeForce GPU provides outstanding graphics performance for next-generation mobile 3D games, smooth HD video playback, exceptional online Flash gaming performance, and highly responsive GPU-accelerated user interfaces without compromising mobile power budgets. A high level block diagram of NVIDIA Tegra 2 SoC with the GeForce GPU core is shown in Figure 1.

## The OpenGL ES 2.0 Graphics Processing Pipeline

OpenGL ES is the standard Application Programming Interface (API) used by developers to write graphics applications for mobile devices such as smartphones, tablets, and portable gaming devices. The OpenGL ES API is a subset of the desktop OpenGL API specification, and defines a flexible and powerful low level interface between the graphics application and the GPU hardware. The most recent OpenGL ES 2.x specification targets modern GPU pipelines that are fully programmable, and replaces all fixed function elements of the API with programmable shading.

Most mobile GPU architectures adhere to the OpenGL ES API standards and they primarily implement the logical processing pipeline as defined by the OpenGL ES API. This logical processing pipeline is illustrated in Figure 2.

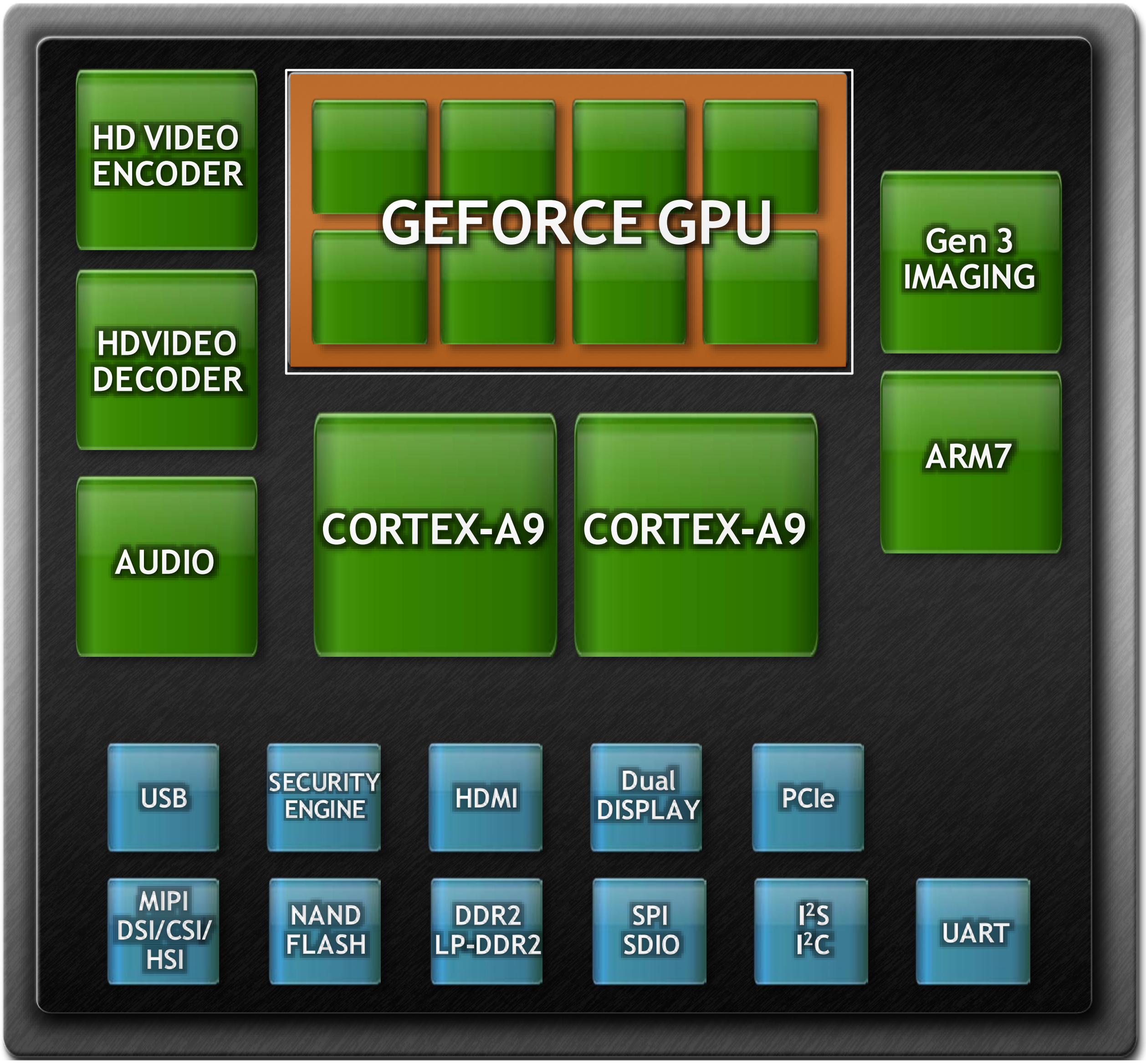


**Figure 1 NVIDIA Tegra 2 with GeForce GPU**

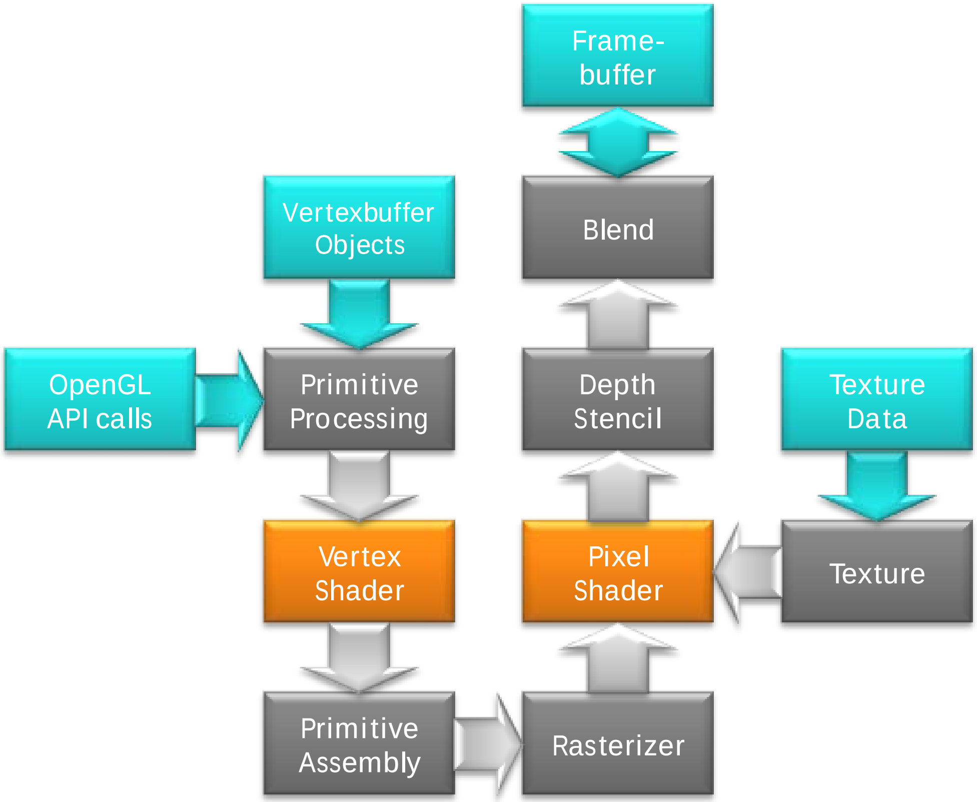


**Figure 2 OpenGL ES 2.0 Logical Graphics Processing Pipeline**

In order to display a scene as defined in a game or graphics application, the application developer must first use 3D modeling software to create various 3D object and character models. The objects and characters can each be comprised of grids of hundreds or thousands or even millions of connected triangles, depending on the desired level of geometric realism. For example, Figure 3 shows how a developer would define a dolphin as a grid of connected triangles.

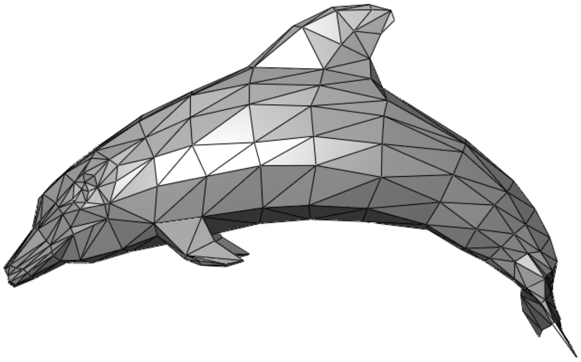


**Figure 3 Triangle Mesh on a three dimensional image**

Next, the models can be used by 3D game software or other 3D applications and placed into a simulated three dimensional scene or “3D world”. The 3D world is defined using an X-Y-Z coordinate system, and the 3D objects or characters are placed at specific locations in the world. Every triangle in an object is defined by its three vertices, and each vertex is comprised of groups of numerical values that represent its properties such as its XYZ location in the 3D world, color value (RGB), alpha transparency level, texture coordinates, normal vectors, and more. Groups of vertices that define a particular portion of an object, such as the forearm of a warrior character, are then grouped into a vertex buffer, which looks like a stream of raw vertices. A 3D scene will be comprised of many vertex buffers.

The 3D software issues an OpenGL ES call to the GPU driver, pointing to the location of the vertex buffer in shared system memory, allowing the GPU to directly access and process the vertex data. The primitive processing stage of the OpenGL pipeline occurs in the GPU and converts inbound vertex data to a format and organization that is used by the GPU. Vertices are then passed to the Vertex shader, where a vertex shader program can run various matrix transforms and lighting calculations to move the vertices to new X, Y, or Z locations, or change lighting values, among other things.

The transformed vertices are assembled into primitives such as triangles, lines, or points. The primitives are then converted by the rasterization stage into pixels fragments in preparation for the pixel shader stage. The pixel fragments are now in a 2D screen space format. The pixel shader stage will run a pixel shader program to process each pixel, possibly applying new lighting or color values, or applying textures, or performing various other operations to ultimately compute a final color value to apply to the pixel.

In the classic OpenGL pipeline, Z-buffer testing will then be performed on each pixel to determine if is closer to the viewer's eye than the existing pixel at the same screen location in the frame buffer. If the new pixel is determined to be closer to the viewer, based on its Z value, it replaces the existing pixel value in the frame buffer, but if it is behind the existing pixel on the Z-axis, it will be thrown away. (Note that the frame buffer may be located in the system memory space that is shared with the CPU, or it may be dedicated memory, such as used in most discrete graphics cards). If the visible pixel has an alpha value that indicates it is partially transparent, it will be blended with the existing pixel in the frame buffer at the same screen location. If anti-aliasing is enabled, pixel color values may be modified to create smoother looking edges to reduce the stair-stepping effect before writing to the frame buffer.

Both the Vertex Shaders and Pixel Shaders defined by the OpenGL ES 2.0 specifications are fully programmable, enabling application developers to create complex and unique vertex and pixel shading effects.

The GeForce GPU processing pipeline in the NVIDIA Tegra mobile processor is similar to that defined by the OpenGL 2.0 specifications, but it has several proprietary optimizations that enable it to deliver performance of a pipelined GPU architecture while remaining within the power budgets required for mobile devices.

## Ultra Low Power GeForce GPU Architecture

NVIDIA is the industry leader and recognized innovator in visual computing. NVIDIA's new ultra low power GeForce GPU in the Tegra processor is derived from the desktop GeForce GPU architecture, but is specifically tailored to meet the growing demands of current and future mobile graphics use cases.

Based on NVIDIA's acclaimed GeForce GPU architecture, the ultra low power GeForce GPU in Tegra is highly customized and modified to deliver console quality gaming, while consuming ultra low power. The GeForce architecture is a fixed function pipelined architecture that includes fully programmable pixel and vertex shaders, along with an advanced texture unit that supports high quality Anisotropic Filtering.

Figure 4 shows the graphics processing pipeline of the GeForce GPU in the Tegra mobile processor.

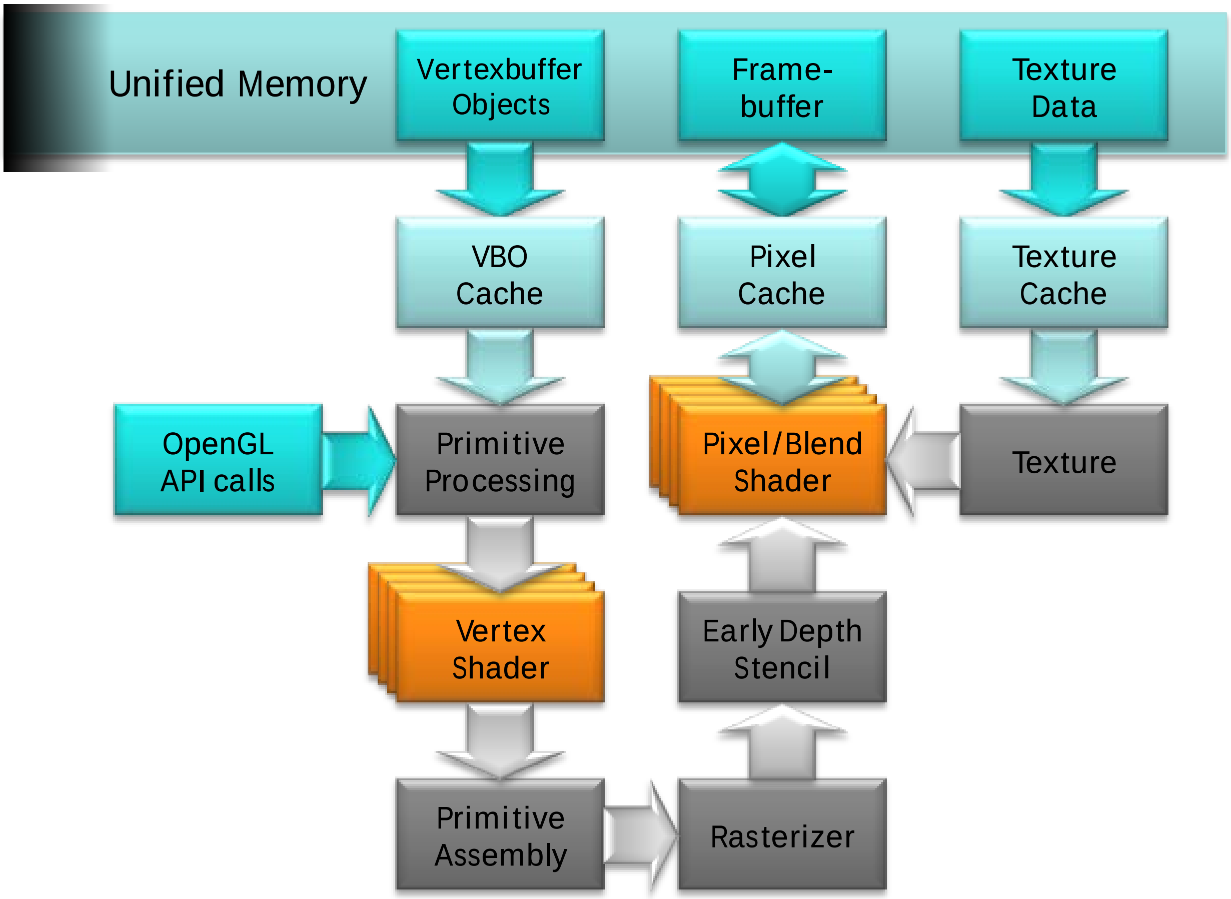


**Figure 4 GeForce GPU Architecture in NVIDIA Tegra**

The GeForce GPU includes four pixel shader cores and four vertex shader cores for high speed vertex and pixel processing. The GPU pipeline uses an 80-bit RBGA pixel format with FP20 data precision in the pixel pipeline, and FP32 precision in the vertex pipeline. It also implements a unique and proprietary Anisotropic Filtering (AF) algorithm that is superior to AF techniques used on many desktop GPUs. The architecture supports advanced features such as High Dynamic Range (HDR) lighting, Multiple Render Targets (MRTs), and non-power of two texture support. The architecture supports both DXT and ETC texture formats.

Although the GeForce GPU architecture is a pipelined architecture similar to that defined by the OpenGL ES 2.0 standards, it includes several special features and customizations to significantly reduce power consumption and deliver increased performance and graphics quality. Some of the unique features implemented in the GPU core and NVIDIA Tegra 2 mobile processor include:

- Early-Z support to filter out non-visible pixels
- Integrated Pixel Shader and Blend Unit for programming flexibility and higher performance
- Pixel Cache, Texture cache, Vertex, and Attribute Caches to reduce memory transactions
- Unique 5x Coverage Sampling Anti-aliasing (CSAA) technique that achieves higher image quality at lower memory bandwidth
- Advanced Anisotropic Filtering (AF) for high detail textures
- A custom Memory Controller developed in-house that improves GPU performance and reduces power consumption
- Numerous Power Management features for ultra low power consumptions.

## Early–Z Technology

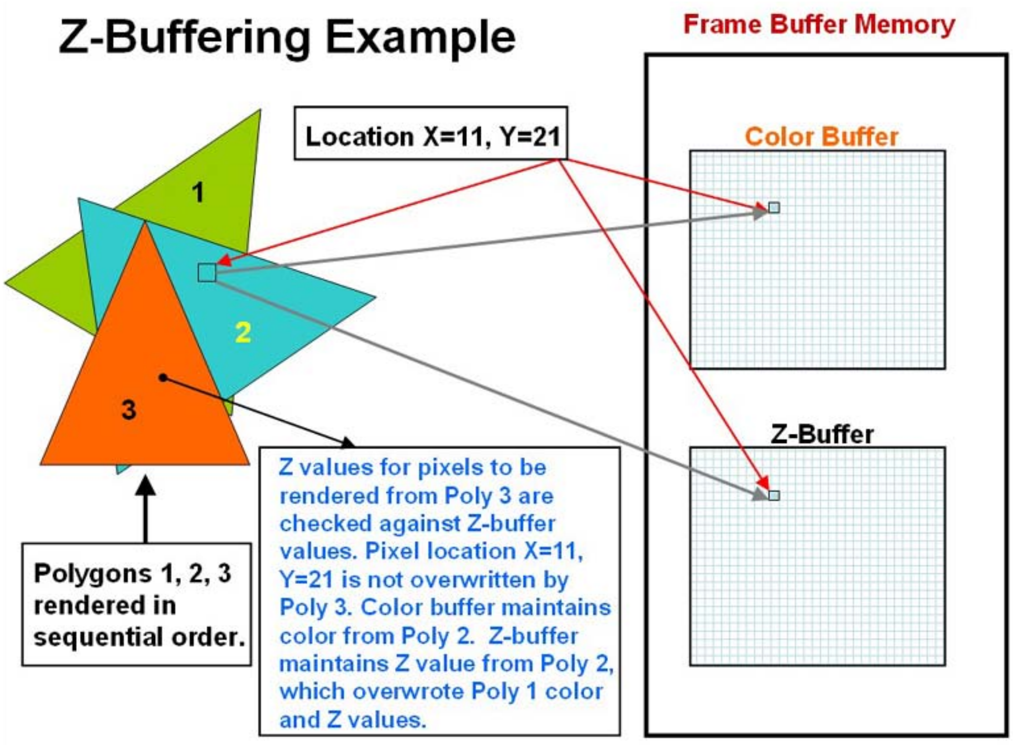
Modern GPUs use a Z-buffer (also known as depth buffer) to track which pixels in a scene are visible to the eye, and do not need to be displayed because they are occluded by other pixels. Every pixel has corresponding Z information in the Z-buffer.

For review, a single 3D frame is processed and converted to a 2D image for display on a monitor. The frame is constructed from a sequential stream of vertices sent from the host to the GPU. Polygons are assembled from the vertex stream, and 2D screen-space pixels are generated and rendered.

In the course of constructing a single 2D frame in a given unit of time, such as 1/60th of a second, multiple polygons and their corresponding pixels may overlay the same 2D screen-based pixel locations. This is often called *depth complexity*, and modern games might have depth complexities of three or four or higher, where three, four, or more pixels rendered in a frame overlay the same 2D screen location.

Imagine polygons (and resulting pixels) for a wall being processed first in the flow of vertices to build a scene. Next, polygons and pixels for a chair in front of the wall are processed. For a given 2D pixel location onscreen, only one of the pixels can ultimately be visible to the viewer—a pixel for the chair or a pixel for the wall. The chair is closer to the viewer, so its pixels are displayed. (Note that some objects may be transparent, and pixels for transparent objects can be blended with opaque or transparent pixels already in the background, or with pixels already in the frame buffer from a prior frame).

Figure 5 shows a simple Z-buffering example for a single pixel location. Note that we did not include actual Z-buffer data in the Z-buffer location.



Polygons 1, 2, 3 rendered in sequential order.

Z values for pixels to be rendered from Poly 3 are checked against Z-buffer values. Pixel location X=11, Y=21 is not overwritten by Poly 3. Color buffer maintains color from Poly 2. Z-buffer maintains Z value from Poly 2, which overwrote Poly 1 color and Z values.

**Figure 5 Example of Z-buffering**

Z comparisons for individual pixel data as defined in the OpenGL ES2.0 logical pipeline happen after the pixels are processed by the pixel shader. The problem with evaluating individual pixels after the pixel shading process is that pixels must traverse nearly the entire pipeline to ultimately discover some are occluded and will be discarded. With complex shader programs that have hundreds or thousands of processing steps, all the processing is wasted on pixels that will never be displayed! More importantly, processing these pixels involve significant amount of transactions between the GPU and shared system memory in the case of mobile devices. Since system memory resides off-chip, the memory transactions consume significant amounts of power and can rapidly drain battery power.

What if an Early-Z technique could be employed to test Z values of pixels **before** they entered the pixel shading pipeline? Much useless work could be avoided, improving performance and conserving power.

The GeForce GPU pipeline introduces an Early-Z stage in the pipeline that is placed before the pixel shader stage. The Early-Z implementation in the Tegra GeForce GPU is an optimized version of the implementation used in high end desktop GeForce GPUs. The Early-Z operation tests all the pixels for Z-depth and passes on to the pixel shader block only those pixels that are visible. By performing an Early-Z operation, the GeForce architecture fetches Z, color, and texture data only for the visible pixels that pass the Z-test. Early-Z is highly efficient and is accurately able to detect and discard hidden pixels.

The main benefits of the Early-Z processing are that it not only significantly reduces power consumption by reducing the memory traffic between the GPU and off-chip system memory, but is also faster than other Z comparison algorithms. Early-Z is highly efficient and is able to identify and discard hidden pixels most of the time. But very rarely, for some special scenes, the programmer may require pixels to be hidden after the pixel shading is done. For these rare cases, the GeForce pipeline implements a late stage depth calculation and blend within the integrated pixel shader and blend unit.

### Integrated Pixel Shader and Programmable Blend Unit

The OpenGL ES 2.0 logical GPU pipeline defines a separate stage for pixel blending that is performed after pixel shading. The fixed function blend unit defined by the logical pipeline supports only a limited set of blend operations. The GeForce pipeline integrates the blend unit with the pixel shader, making it a fully programmable blender. Due to the integrated design, the pixel shader can harness the processing power of the blender when there are no blend operations in progress. In addition, the programmable blender allows GeForce GPU to implement blend modes that are not defined by the OpenGL spec. For example, Adobe's Flash Player uses several blend modes that are not supported by OpenGL, but the GeForce GPU is able handle these blend modes due to its programmable blender.

### Pixel and Texture Caches Reduce Memory Transactions

The traditional OpenGL GPU pipeline specifies that pixel information such as texture, depth, color, and other attribute values are stored in system memory (or frame buffer memory). The pixel information is moved to and from memory during the pixel processing stage. This requires a significant number of off-chip system memory transactions, and thus consumes large amounts of power. The GeForce architecture has implemented on-chip pixel, texture, vertex, and attribute caches along with unique cache management algorithms to not only reduce the system memory transactions, but also maximize the utilization of these caches.

The pixel cache is used to store on-chip Z and color values of pixels and can be reused for all pixels that are accessed repeatedly, such as User Interface components. Also, due to good spatial and temporal locality of pixel color and depth data in many other graphics scene images, pixel caches deliver good cache hit ratios and lower the need to access system memory for data.

Texture data has good spatial and temporal locality. A given pixel typically uses many of the same texture elements (texels) as an adjacent pixels during texture filtering operations such as bilinear filtering, and textures often remain the same for at least a few frames of the image. Thus caching texture data on-chip will result in good reuse of texture data and significantly reduce system memory accesses to fetch texture data.

Designing efficient texture caches is extremely complicated and using generic cache management techniques would result in poor hit ratios. The pixel and texture cache designs of the GeForce GPU are based on over ten years of expertise in developing efficient caching techniques for NVIDIA desktop GPUs. The texture cache in the ultra low powe GeForce GPU delivers over ninety percent hit rate on average, while the pixel cache delivers fifty percent hit rates for many situations. Thus the GeForce GPU achieves significant power savings by caching large portions of color, Z, and texture data by minimizing accesses to system memory.

### Coverage Sampled Anti-Aliasing

The NVIDIA GeForce GPU core in Tegra implements a unique anti-aliasing technique known as Coverage Sample Anti-Aliasing (CSAA). Aliasing is the stair-like jagged edges that appear on images where there should be smooth lines or edges, and anti-aliasing (AA) techniques are used in computer graphics to make these jagged lines appear smooth. Aliasing effects occur when high resolution images are displayed on lower resolution displays, or when higher resolution images are converted to lower resolution.

Traditionally, GPUs have used AA techniques known as Multi-Sample AA (MSAA) and Super-Sample AA (SSAA) to reduce Aliasing effects. These techniques require the pixels of the resolution image to be rendered multiple times (e.g. two times for 2x AA, four times for 4x AA etc.), creating smooth lines and edges by calculating a final pixel from multiple rendered samples. (When anti-aliasing is enabled, a pixel region is divided into a number of sub-pixel regions that correspond to the anti-aliasing level, such as four sub-pixel regions for 4xAA, and each sub-pixel region includes a sample location that can be are used to determine polygon coverage for calculating final pixel color). Due to multiple rendering passes, the classic anti-aliasing techniques use significant amounts of memory bandwidth and are not suitable for mobile GPUs, where power consumption is of paramount concern.

CSAA can produce anti-aliased images that rival the quality of higher level MSAA, while minimizing the accesses to system memory. CSAA introduces the concept of a new sample type -- a “coverage sample” -- that is used to improve the accuracy of coverage calculations when determining the percentage a polygon covers a pixel’s area. Note that multiple polygon fragments can cover and overlap the area of a single screen pixel. In some cases, very small polygons may actually be smaller than a pixel's area, and only cover a portion of the pixel. When processed sequentially in a graphics pipeline, portions of multiple polygons, each with different color attributes, may overlap and contribute to the final color of a single pixel in a frame.

Coverage samples differ from previous AA techniques where coverage was always inherently tied to a "real" sample type. In SSAA, each real sample has its own unique color and Z values, and in a 4xAA case, the shader program runs four times, and four textures are fetched – one for each sample (or more in the case of multi-texturing). With 4xAA, the frame buffer is 4x larger than the case of not using anti-aliasing, and is down-filtered to create the final pixel color.

MSAA reduces the shader overhead of the anti-aliasing operation by fetching a single texture color values and applying that single color to all samples, thus reducing shader execution and texture fetch processing. Z/stencil values must still be unique in MSAA to ensure proper Z-ordering of the samples.

CSAA further optimizes the AA process by decoupling simple coverage samples from color/z/stencil/coverage samples, thus reducing bandwidth and storage costs compared to MSAA and SSAA. CSAA uses more coverage samples to calculate coverage levels of the polygons within a given pixel area, thus creating a higher quality AA effect without incurring the memory and power costs of processing additional real color and Z samples.

The GeForce GPU core in the NVIDIA Tegra 2 mobile processor supports 5x CSAA (one real sample and four coverage samples) delivering incredible image quality enhancements at very low memory transaction and power consumption costs.

### Advanced Anisotropic Filtering

Anisotropic filtering is a technique employed to enhance the image quality of textures on surfaces that are at oblique viewing angles. Each pixel on the screen typically requires multiple texture elements to be fetched from texture maps in memory, filtered, and applied to the pixel to change its color. When viewing a surface head-on, perpendicular to the camera or viewer, an equal number of texture elements are usually sampled per pixel using a square sample pattern. However, at extreme viewing angles when the image on the screen extends further along one axis than another, taking equal number of samples in each axis from the texture map would result in texture blurring along the axis that stretches out into the horizon.

The image on the left in Figure 6 shows a runway that stretches out into the horizon, and you can see that the texture detail is blurred for the sections of the runway that are closer to the horizon. Anisotropic filtering techniques intelligently take more texture samples along the axis that is stretched out and preserves the texture details for textures along this axis. The image on the right in Figure 6 shows the runway image that has Anisotropic Filtering applied, and you can see that the runway texture detail has improved significantly.

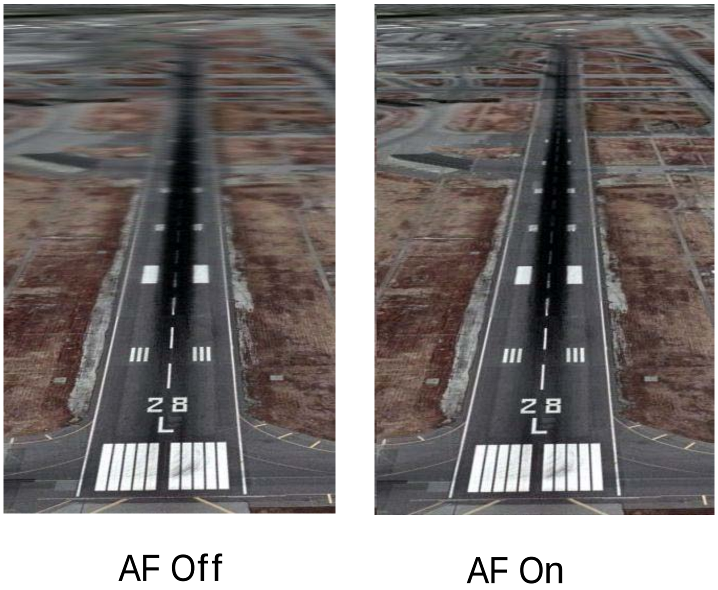


**Figure 6 Improved texture quality from Anisotropic Filtering**

The GeForce GPU in NVIDIA Tegra 2 processor employs a very sophisticated Anisotropic Filtering technique that delivers texture quality enhancement similar to that on high end NVIDIA Fermi GPU architecture-based desktop GPU cards. The GeForce GPU supports up to 16x Anisotropic Filtering. It uses adaptive filtering algorithms and efficient texture cache management techniques to deliver high texture quality without significantly increasing memory transactions.

The advanced AF and CSAA features of the NVIDIA Tegra 2 GeForce GPU will deliver outstanding graphics quality that is far superior to that delivered by competing solutions. As screen sizes and resolutions of mobile devices increase, graphics quality will greatly influence the user experience, and NVIDIA Tegra 2 mobile processor will deliver the best game play and visual experience for modern mobile games and applications.

### Optimized Memory Controller

NVIDIA Tegra 2 mobile processor includes GPU and Memory Controller (MC) cores that have been completely designed from the ground-up based on over a decade of expertise in building highly optimized GPU and memory controller cores. The performance of GPU cores is highly dependent on the efficiency of the MC in delivering bandwidth, and also the latency requirements for graphics processing. Due to the in-house development of both the GPU and the MC, the MC is highly tuned to the specific requirements of the GeForce GPU, and includes several optimizations that enhance GPU performance and reduce power consumption.

Some of the key optimizations that are included in the MC controller design are:

- **Dynamic Clock Speed Control (DCSC):** DCSC enables the memory controller to quickly ramp up operating frequency in response to advance indicators from the GPU core for system memory accesses, and quickly ramp down operating frequency to a power saving levels when the GPU has completed its memory accesses. Due to the close knit in-house design process, the MC can directly tap into GPU core hardware to proactively anticipate GPU needs and manage its operating levels to meet GPU needs (unlike competitive solutions that use cobbled together mixed-vendor CPU and GPU cores).
- **GPU centric Memory Arbitration:** System memory is one of the most valuable resources in a mobile processor. Various cores such as the CPU, GPU, Video, and Audio cores require responsive and high bandwidth access to system memory. The MC implements advanced arbitration schemes to efficiently grant multiple clients access to system memory.

  The MC core has advanced knowledge of the type and urgency of memory access requests coming in from a GPU client and implements a very fine tuned arbitration scheme that delivers high bandwidth for bandwidth-hungry render and geometry requests, and ultra low latency to service high priority latency-sensitive display and CPU requests. The MC is also aware of the priority of each request generated in the GPU core, and is able to further optimize its performance to meet the demands of these requests.
- **GPU Request Grouping:** Off-chip system memory devices are capable of keeping open only a certain number of memory banks at any given time. When requests access regions of memory that are not contained within currently open banks, the MC has to close the currently opened banks, and then activate new banks that include the desired memory cells or regions. This process not only impacts latency and bandwidth, but also requires higher power consumption.

  Rather than initiate multiple different memory requests that access random banks in different parts of the memory subsystem, the GeForce GPU is aware of the current system memory configuration, and optimizes access patterns. The GPU can group together memory requests that access the same memory bank. The MC controller can also re-order independent memory requests into groups based on their memory bank access pattern. Such features deliver higher efficiency memory accesses, and reduce power consumption by limiting frequent memory bank switching.

### Advanced Power Management

The GeForce GPU core implements several advanced power management techniques to reduce power consumption including:

- **Multiple Levels of Clock Gating:** The GPU implements several levels of clock gating that shut off clocks during idle conditions. A system level power control algorithm controls the power and clocking of all eight cores in the NVIDIA Tegra 2 processor.

When the power control logic detects an idle GPU core, it clock-gates the main trunk clock that feeds into the GPU, effectively driving dynamic power consumption of the GPU to almost zero milli-watts. When the power controller detects the system is in standby mode, it power-gates the GPU core, reducing its power consumption to nearly zero.

- **Local Power Management Features:** The GPU core has several local power management features to further reduce power consumption. It implements several function-level clock gating mechanisms that can clock-gate various different idle blocks within the GPU core. For example, when the pipeline isn't performing any vertex shading tasks, the vertex shader is clock-gated and put in a low power state until the next vertex shading commands are received. Similarly, when the pixel shader is working on tasks such as math calculations that do not require texture fetches, the texture units can be clock-gated. Also, if the GPU is just refreshing the device display and not actively rendering, the memory controller can opportunistically put system memory into low power state.
- **Display Request Grouping:** The GPU groups multiple display requests, and issues these requests in bursts to system memory. Then the GPU informs the memory controller (via timers) about the timing of the next request burst. In the idle period between GPU display request bursts, the memory controller looks for opportunities to aggressively and dynamically put system memory in low power states.
- **Power-Optimized Transistor Design:** The GeForce GPU core was also optimized at the transistor level for ultra low power consumption. Lower leakage transistors are used for blocks that are not timing sensitive, and higher speed transistors are used for critical paths that require high speed operation. Thus the GeForce core is able to achieve low power consumption without compromising performance.
- **Dynamic Voltage and Frequency Scaling (DVFS):** The Tegra 2 mobile processor also implements a highly advanced chip-level DVFS technique that at any given time is able to control the clock frequencies of six main system clocks and the voltage levels of up to two voltage rails. The clocks and voltage rails that are under DVFS control can be selected through software-controlled settings.

  The basic concept of DVFS is to vary the core frequency and voltage of various processing units in the processor to control power consumption. The power consumed on a semiconductor chip is directly proportional to the operating frequency, and is also proportional to the square of the operating voltage.

  When the processor is not working on any tasks, the frequency and voltage levels can be dropped to lower levels to greatly reduce idle power consumption. When an incoming task is detected by any one of the eight cores in NVIDIA Tegra, the event is reported to the global DVFS control block and the frequency and voltage levels are immediately increased to the appropriate operating values to ensure higher performance. The DVFS software intelligently raises the voltage and frequency only up to a level that is required to deliver the performance demanded by the application. The

DVFS algorithm has very fine control over the frequency levels, and can increase or decrease frequency in steps as small as 1 MHz.

## User Benefits

### Higher Performance for Mobile Gaming

Mobile Gaming is a rapidly growing use case, and recent market study data showed that the number of users playing games on smartphones has increased by sixty percent in the 2009-2010 time period[1]. In addition, many games played on mobile devices today have evolved from simple 2D versions, to using complex 3D rendering techniques. Many visually rich and compelling games from the PC and console platforms are becoming available for mobile devices.

The growing popularity of gaming on mobile devices has attracted top PC game developers, and many PC-class games such as Call of Duty, Brothers In Arms, and Rage are now available on mobile devices. Major PC game developers are investing heavily in mobile gaming, and competition among game developers will drive the creation of visually rich games that use advanced features such as shadows, particle effects, lighting, and physics.

NVIDIA Tegra 2 is the only mobile processer that delivers console quality gaming. Due to the desktop GPU roots of the GeForce GPU core in Tegra, the hardware pipeline of the GPU is optimized and prepared to handle the performance requirements of mobile games that use PC and console quality graphics features. The similarities between the GeForce GPU architecture and its more powerful desktop and console counterparts will also enable developers to easily develop games that are capable of running across multiple platforms. Developers can employ the same rendering techniques for all platforms and then add appropriate graphics detail and complexity for each specific platform.

Companies such as Trendy Entertainment are developing multi-player, cross-platform games by leveraging the commonality between the NVIDIA GeForce GPUs in PCs, Sony Playstation 3, and NVIDIA Tegra 2 based mobile devices.

Benchmark results on popular mobile games such as Need for Speed: Shift, Epic Citadel, Raging Thunder, and Galaxy on Fire show that these games run twenty to fifty percent faster on the NVIDIA Tegra 2 processor than on the fastest available tiling-based mobile GPU processors. These games currently do not employ many advanced graphics features, and the performance advantages of NVIDIA Tegra 2 will further increase when advanced graphics features are implemented in such mobile games.

[1] www.comscore.com/Press_Events/Press_Releases/2010/4/Smartphone_Adoption_Shifting_Dynamics_of_U.S._Mobile_Gaming_Market

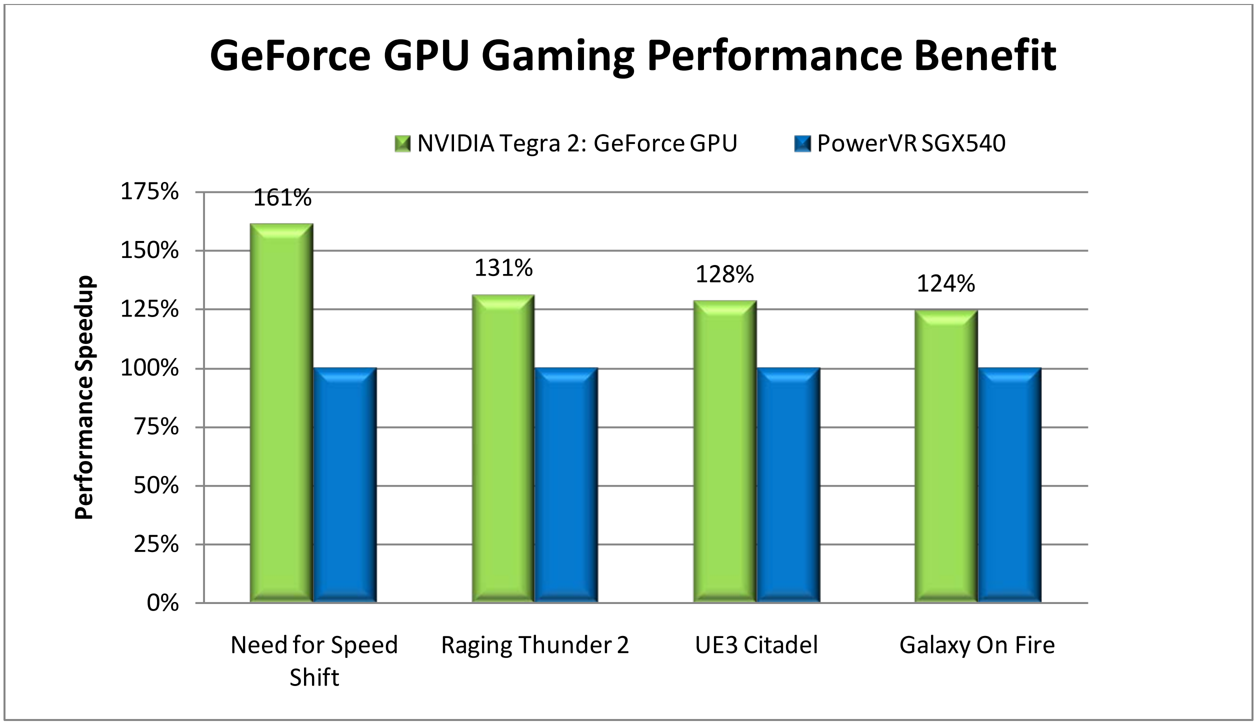


**Figure 7 GeForce GPU Performance on Mobile games**[2]

## Benefits of Hardware Accelerated Adobe Flash

The Adobe® Flash® Player is a widely used Web browser plug-in that is installed on over 99% of desktop PCs. Most websites use Flash to deliver a wide variety of rich content, including videos, animation, and games. Some of the most popular Web-based games such as Farmville®, Bejeweled® and Plants Vs Zombies® are delivered via the Flash platform. Popular video streaming sites such as Youtube.com®, Vimeo.com®, Hulu®, and TV.com® use Adobe Flash platform to deliver video content, and most Web sites have rich interactive content that is based on Flash animation.

The recent release of the mobile version of Adobe Flash Player enables users to access Flash based content and view full featured PC-like versions of Web sites on their mobile devices. However, processing Flash-based content is a performance intensive task and takes up a significant amount of CPU processing power. Browsing a Flash-enabled website on a mobile device could consume most of the CPU processing power, leaving very little for other applications. If a mobile device has applications such a streaming audio, social networking, and email apps running in the background, the CPU may not have sufficient processing power to process all tasks, and thus users may notice significant stuttering or slow performance on Flash-based Web browsing, video playback, or gaming.

[2] Data collected on NVIDIA Tegra 2 Dev Platform and Samsung GalaxyS phone at 800x480 resolution, Android 2.2

Even if there are no background tasks, some Flash games such as RaidenX and Floodrunner are so intensive that running these on the CPU results in poor performance. In addition, the CPU is a general-purpose processor and is highly inefficient when processing content such as video and games delivered via Flash. The CPU not only struggles to process Flash-based content, but also consumes large amounts of power, which decreases battery life.

The NVIDIA Tegra 2 mobile processor addresses this problem by processing nearly all Flash-based rendering tasks on the GeForce GPU core.

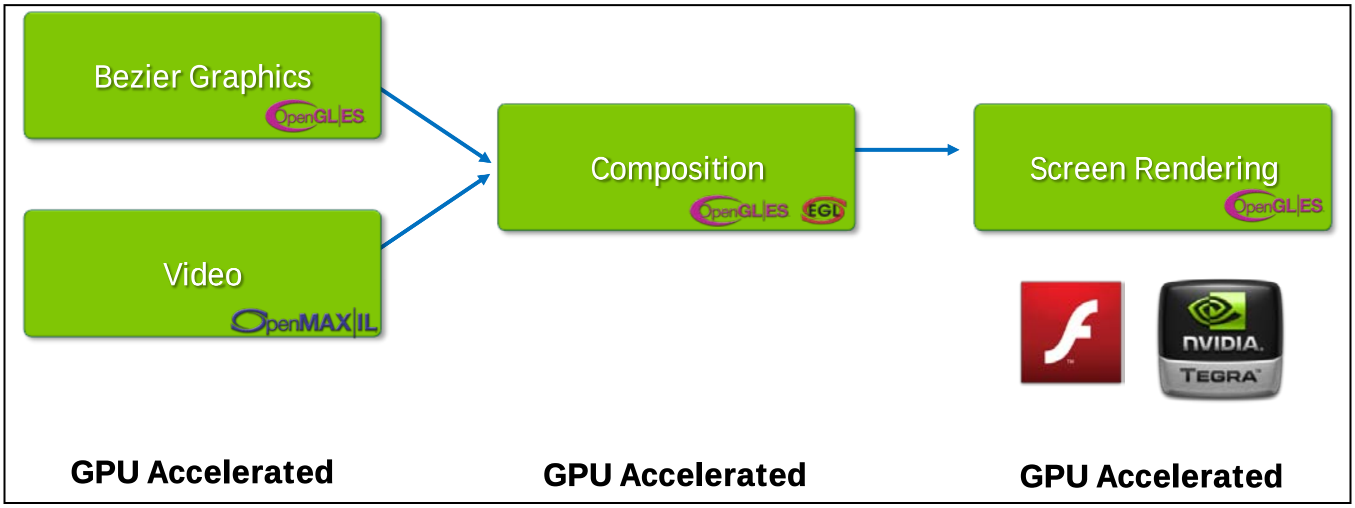


**Figure 8 GPU Flash Accelerated Pipeline**

Since Flash video playback and gaming involves graphics and pixel processing, the GPU core is better equipped to process these tasks efficiently and at high performance. NVIDIA has worked closely with Adobe to implement a mobile Adobe Flash Player that completely offloads all Flash-based video and game content processing to the GeForce GPU.

Offloading the Flash processing to dedicated hardware that is designed specifically for graphics tasks not only delivers much higher performance, but also improved power utilization. Another significant benefit is that the CPU is freed up to handle other applications.

The following two diagrams illustrate the processing pipeline for Flash video playback and Flash-based graphics. The blocks highlighted in green show the processing steps that are hardware-accelerated and processed in the GeForce GPU.

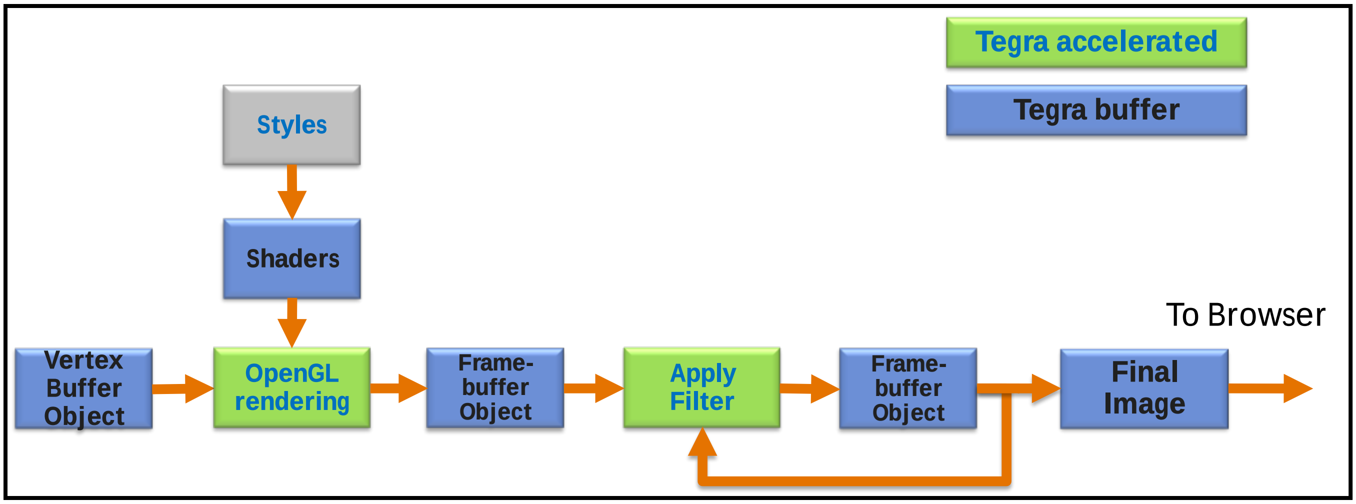


**Figure 9 Acceleration of Flash graphics on Tegra GeForce GPU**

Figure 9 shows how the OpenGL ES pipeline is fully leveraged to accelerate Flash-based graphics effect. A “style” shown in the diagram describes how to render the inside of a Flash object on the screen. It could be a solid color, gradient fill, an image or video applied to the object, etc. The style also describes textures to apply, or which OpenGL ES vertex and pixel shaders to be used to achieve the desired rendering effect. Vertices are the basic building block of the 3D graphics pipeline, and are used to describe the outlines of the Flash objects.

The Figure 9 diagram flow assumes a complex Flash scene, where a variety of filter affects are applied in a multi-pass process. Examples of filters include blurring an image, edge detection, applying highlights to an image, etc. Filter effects are implemented as fragment shaders, and also rendered with the GPU using OpenGL ES. Some complex scenes can be made up of more than 10 filter effects. A GPU can chew through that pretty quickly. Farmville, for example, applies a lot of filters.

The result of the rendering step is the final image, still stored in GPU memory. If capable, the browser can pull the image straight from GPU memory for further compositing into the web page. That compositing can also be done by the GPU, which will be an additional performance benefit.

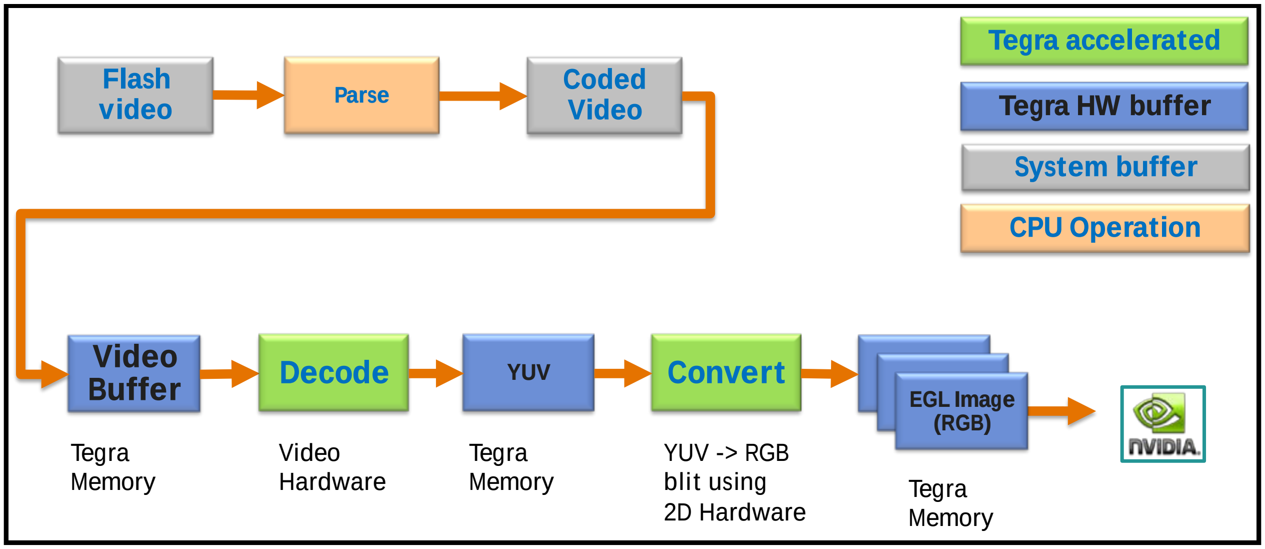


**Figure 10 Flash Video Acceleration on NVIDIA GeForce GPU core**

Figure 10 shows the video acceleration path in Flash. A video file first gets parsed and stored as a coded video stream inside a Flash player buffer. The coded video stream is then transferred to a Tegra hardware buffer where dedicated video hardware will process it, and produce a YUV image for each video frame. As the final rendering is always in RGB, the YUV image is converted to RGB color space by another Tegra hardware block. The resulting series of RGB images can simply be used as another rendering style, and rendered using OpenGL as textured quads.

In Figure 11 below, results from GUIMark2®, a popular Flash benchmark, show that a hardware-accelerated Flash player running on a NVIDIA Tegra 2 based mobile device delivers 2x to 3x the performance of a device that utilizes the CPU for processing Flash content. Performance measurements on popular online Flash-based games such as RaidenX, Murloc, Crush the Castle 2, and Farmville show that hardware-accelerated Flash on NVIDIA Tegra 2 delivers almost 2x the performance of competing devices that use the CPU to process Flash content.

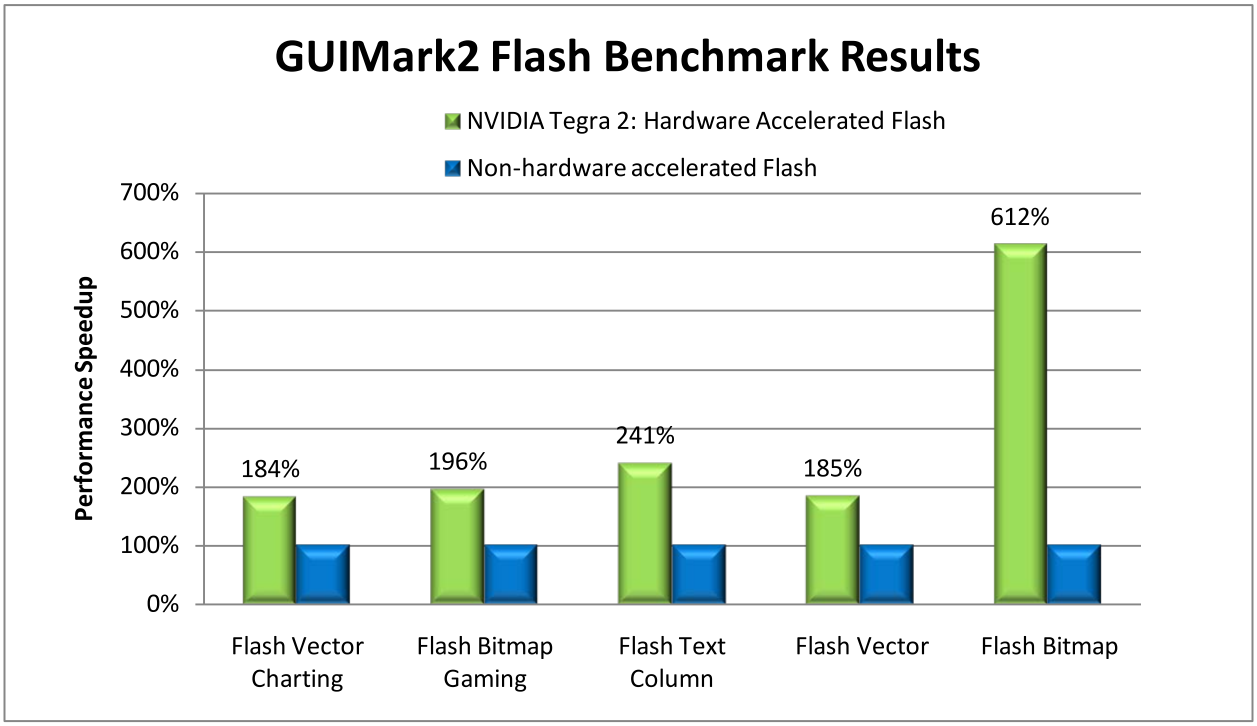


**Figure 11 GUIMARK2 Flash Benchmark Results**[3]

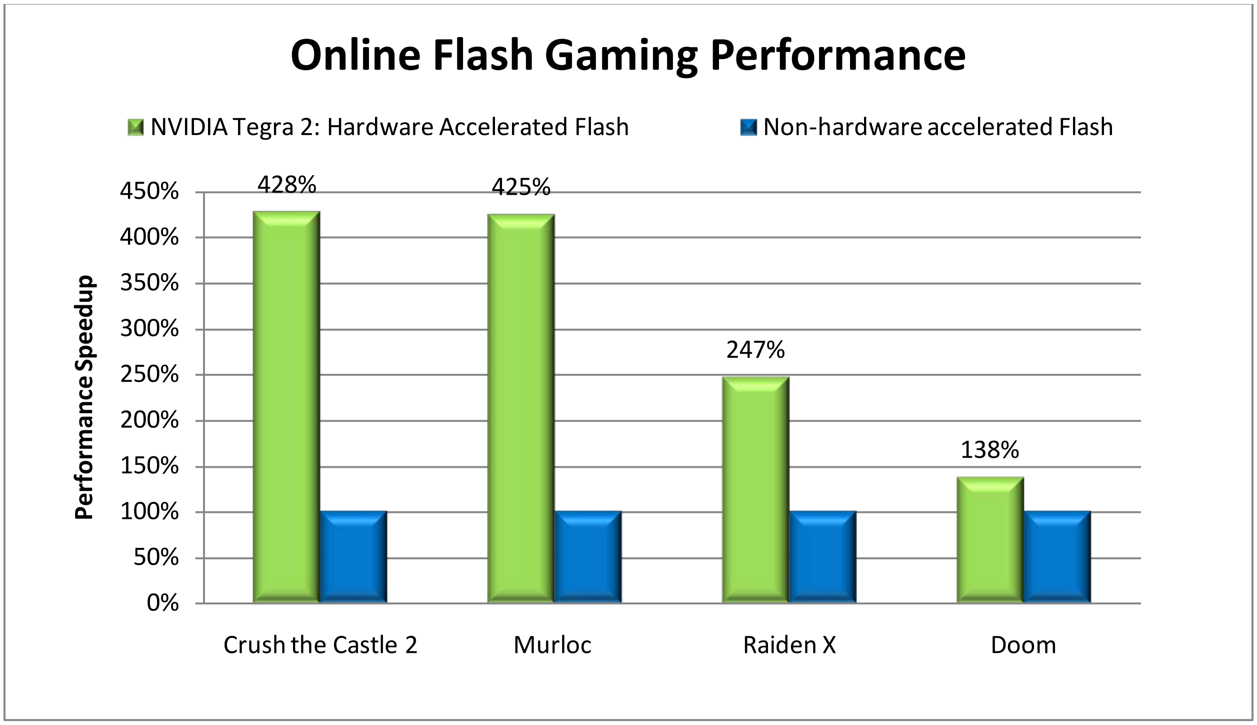


**Figure 12 Flash Gaming Performance on GeForce GPU**

[3] Data collected on NVIDIA Development Platform and Samsung GalaxyS phone at 800x480 resolution on Android 2.2

### Highly responsive User Interfaces

Touch is one of the most natural user interface mechanisms, and most smartphones today support touch-based user interfaces. The availability of multi-touch enabled touch screen displays has further increased the variety of touch-based interactions that are possible with mobile devices. Touch-based displays offer the benefit of natural interactivity that permits users to directly touch and interact with elements that they see on a display. However, touch-based displays also make the user highly sensitive to responsiveness and visual fluidity. Touch-based interactions with mobile devices could range from basic operations such as selection, scrolling, zooming, and panning, to advanced multi-touch operations within games and applications.

Due to the heightened user sensitivity to touch-based interactions, it is critical that mobile devices are able to deliver a highly responsive and fluid user interaction experience, even under heavy multitasking conditions. User interaction tasks on most mobile processors are processed on the CPU and offer fairly acceptable performance but under heavy multi-tasking conditions, the CPU is usually heavily loaded, and may not have the processing bandwidth to handle all requests, which could easily stall the processing of some tasks. Users may not notice these stalls on most background tasks, but will immediately take notice if their phone does not quickly respond to their touch inputs. The CPU has to detect user input, process the task requested by the user, and quickly display on screen the response to the user input. This can be as simple as user scrolling down a menu tree, or as advanced as using multi-touch inputs to rotate an image on a map.

Similar to Adobe Flash content processing, touch interactions trigger a significant amount of pixel processing, and most of the pixel processing for the user interface (UI) in current operating systems is performed by the CPU. Therefore to deliver fluid and snappy UI responsiveness even under heavy multi-tasking conditions, mobile processors must either have sufficient headroom in CPU processing power, or offload some of the UI-related pixel processing to a GPU core that is optimized to handle such tasks. The NVIDIA Tegra 2 mobile processor has a dual core ARM A9 processor that provides almost twice the processing power of competing single core CPU based mobile processors, and delivers outstanding user responsiveness and fluidity even in the absence of GPU-accelerated UI processing.

As mentioned, most mobile operating systems currently use the CPU to handle the pixel processing for user interfaces, and therefore cannot offload this task to the GPU. It is highly likely that future versions of mobile operating systems will enable hardware acceleration of user interfaces, and the Tegra 2 mobile processor will be capable of completely offloading all UI pixel processing to its GeForce GPU.

The transition to GPU-based UI rendering happened on PCs with the introduction of Windows Vista® and its Aero interface, and was further enhanced with Windows 7®. NVIDIA GeForce GPUs flawlessly accelerated the visually rich Windows Vista and Windows 7 UIs, and delivered a highly responsive experience. The GeForce GPU core of the NVIDIA Tegra 2 processor is similar in architecture to its desktop counterpart and is equipped today to deliver outstanding user responsiveness and visual interaction for upcoming releases of hardware-accelerated mobile operating systems.

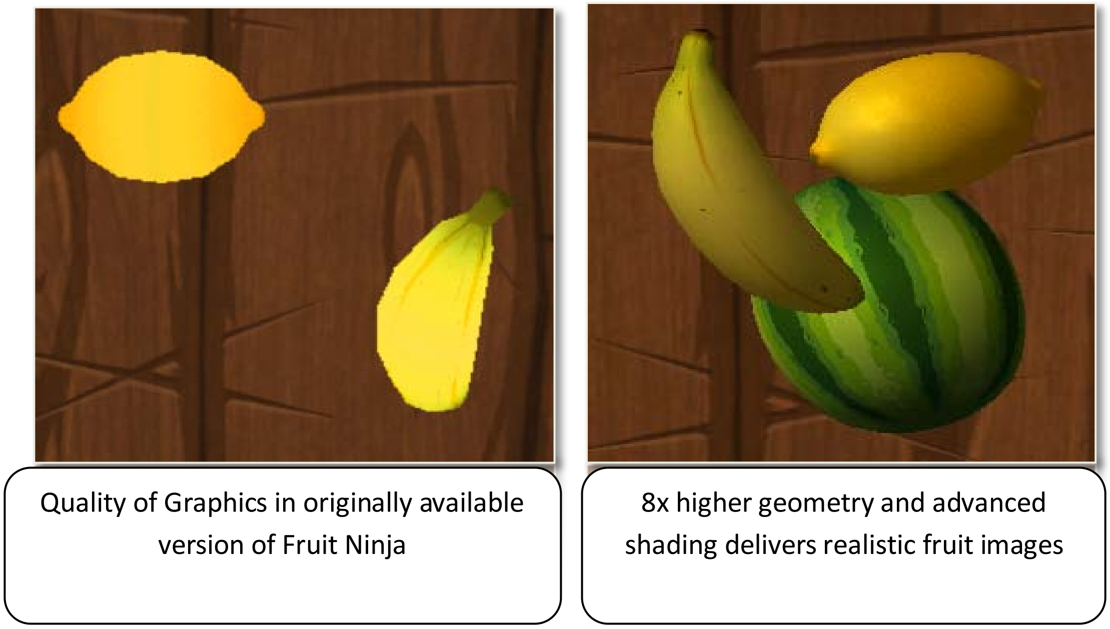
### Premium Content for Tegra Based Mobile Devices

NVIDIA has the processor industry's largest dedicated Content Development team that works closely with game developers to optimize games so they deliver the best performance and quality on NVIDIA GPUs. NVIDIA is leveraging its Content Development team to support the development of high performance, high quality games and apps for mobile Tegra 2 devices.

High quality game content can be defined by the following qualities:

- Visually realistic looking scene objects and character that use much more geometry, higher resolution textures, and complex shaders.
- Highly interactive gaming that employs larger number of characters on screen and more instances of independent animation.
- More challenging games that use advanced Artificial Intelligence (AI) processing with complex worlds and player management.
- Premium levels that are only accessible via Tegra-based devices and additional game character equipment and special moves that are unlocked on Tegra based devices.

The following two figures illustrate some of the enhancements that are being built into games to deliver a premium experience. Figure 13 illustrates the changes implemented in the popular Fruit Ninja game. The original versions of the game use lower quality geometry and textures, resulting in the fruit looking more like lumps of color than real fruit. Through optimizations the game developer is able to use higher geometry and more detailed textures for the fruit to deliver a more realistic visual experience.

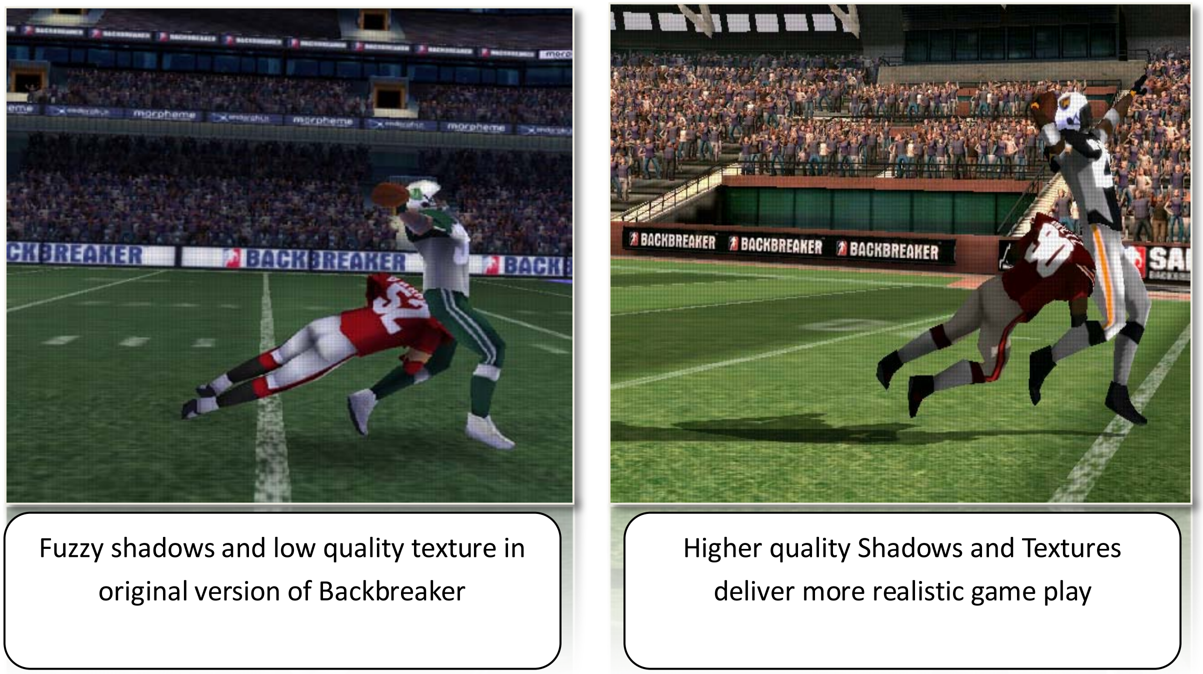


Quality of Graphics in originally available version of Fruit Ninja

8x higher geometry and advanced shading delivers realistic fruit images

**Figure 13 Fruit Ninja Enhancements**

The figure below illustrates the enhancements done in the premium version of Backbreaker. Using the higher processing power, developers are able to implement real and dynamic shadows, and higher quality textures.



Fuzzy shadows and low quality texture in original version of Backbreaker

Higher quality Shadows and Textures deliver more realistic game play

**Figure 14 Enhancements to textures and higher quality shadows in Backbreaker**

## Conclusion

The smartphone is quickly becoming an essential computer. Consumers are hyper-aware of responsiveness, performance and visual quality of their mobile devices and are demanding highly responsive user interfaces, an uncompromised full-featured web browsing experience, visually compelling online and offline gaming, and access to all their content in a form factor that provides cell phone-like mobility and battery life.

Mobile gaming is a rapidly growing use case and it is estimated that end-user spending on mobile gaming will grow from $ 5.6 billion in 2010 to $11.4 billion worldwide in 2014[4]. Adobe Flash based online gaming is resurging due to the tremendous growth of Farmville and other social games that run on the Flash platform. The vast majority of streaming videos available on the Web are delivered via the Flash platform, and users are spending more time on the Web watching videos than browsing text-based websites.

As the resolution of mobile devices continues to increase, they will need GPUs that not only handle the increased pixel processing loads, but also remain within mobile power budgets. Higher resolutions and display sizes will drive consumer demand for applications that employ higher graphics quality features such as complex lighting, shadows, anti-aliasing, higher detail textures, and more.

The GeForce GPU in NVIDIA Tegra 2 is tailored to meet the growing demands of current and future mobile use cases. The high performance architecture has several unique optimizations that not only deliver near console quality graphics, but also ultra low power consumption. Features such as Early-Z, pixel caches, texture caches, integrated pixel shader & blender, CSAA, 16x AF, and an optimized memory controller deliver outstanding visual quality and scalable performance, while keeping power consumption within mobile budgets. Advanced power optimizations such as multi-level clock gating, DVFS, and the use of low leakage fabrication processes delivers industry-leading battery life for NVIDIA Tegra 2 based mobile devices.

The GeForce GPU delivers outstanding graphics quality for mobile gaming, silky smooth online Flash video streaming experience, excellent Flash gaming experiences, and highly responsive User Interfaces. NVIDIA Tegra 2 is the first mobile processor that is capable of playing multi-platform console-class games without compromising visual quality or game play experience. In addition, the TegraZone Marketplace application created specifically for NVIDIA Tegra 2 based mobile devices will enable users with access to premium high quality mobile games and applications specifically optimized to take advantage of the superior graphics capabilities of the GeForce GPU.

Almost all of Adobe Flash software pipeline is now being accelerated by the GeForce GPU core on the NVIDIA Tegra 2 processor. This not only delivers smooth Flash based video playback, but also significantly higher performance on Flash based online games such as Farmville,

[4] http://www.gartner.com/it/page.jsp?id=1370213

Crush The Castle 2, and Floodrunner 2. The hardware acceleration of Flash is available today only on NVIDIA Tegra 2-based mobile devices, and these devices deliver a far superior user experience than competing mobile processors that use CPU-based software processing of Flash workloads.

The size and resolution of mobile device displays have rapidly increased over the last couple of years and the growing popularity of tablets will further push the resolution and display sizes to be close to that of laptop PCs. These current and future mobile devices will need GPUs that can not only handle the increased pixel processing loads, but also remain within mobile power budgets. Higher resolutions and display sizes will drive consumer demand for applications that employ richer PC-like graphics quality employing complex lighting, shadows, anti-aliasing, higher quality textures, and more.

The GPU core in the NVIDIA Tegra 2 mobile processor is based on the proven and successful GeForce architecture and is built for the future with both the performance and power efficiency to deliver an outstanding visual experience that can easily scale with increases in resolution and graphics application complexities.

# Document Revision History

Rev 1.0

**Notice**

ALL INFORMATION PROVIDED IN THIS WHITE PAPER, INCLUDING COMMENTARY, OPINION, NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, “MATERIALS”) ARE BEING PROVIDED “AS IS.” NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE.

Information furnished is believed to be accurate and reliable. However, NVIDIA Corporation assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication or otherwise under any patent or patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. NVIDIA Corporation products are not authorized for use as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

**Trademarks**

**Copyright**