AMD
Smarter Choice

DirectX® 10.1

Enabling breakthrough graphics on the ATI Radeon™ HD 3800 series

ATI RADEON GRAPHICS

Table of Contents

Introduction

Microsoft® DirectX® 10.1 is the latest application programming interface from Microsoft that unlocks the state of the art in GPU technology, represented by the ATI Radeon™ HD 3800 series. Key features include an updated shader model, improved anti-aliasing support, more flexible data access, and tighter specifications for better application compatibility. These features will enable exciting new techniques, such as real-time global illumination, that will define the future direction of interactive 3D graphics.

DirectX 10 was one of the most significant updates to the API since its inception. DirectX 10.1 represents an evolutionary update that addresses some of the limitations identified after the specification was finalized. DirectX 10.1 support will be coming to the Windows Vista™ operating system with the release of a service pack in early 2008.

For many years, ATI was at the forefront of DirectX technology development, working actively with Microsoft to identify and implement new graphics features. The merger of ATI with AMD in 2006 not only continues this tradition, but also enables new possibilities for platform-level synergies between the GPU, CPU, and system chipset.

The new ATI Radeon HD 3800 series of GPUs are the first to be designed for DirectX 10.1, as well as other cutting edge technologies, including PCI Express 2.0, Unified Video Decoder (UVD), hardware accelerated tessellation, and power efficient 55nm transistor design. The products are perfectly positioned to deliver the best experience in not only today's games, but also in next-generation titles releasing in 2008 and beyond.

This paper describes the new features of DirectX 10.1, and provides a number of examples showing how they can be put to use. To help illustrate these techniques, AMD has created an accompanying interactive game called PingPong. This game makes extensive use of the DirectX 10.1 features on ATI Radeon HD 3800 series products to highlight the benefits in a fun and informative way.

The Evolution of DirectX

DirectX 10.1 maintains the overall structure and programming model of DirectX 10, while providing numerous enhancements. The vertex, geometry, and pixel shader instruction sets have been updated to Shader Model 4.1.

The new features of DirectX 10.1 can be divided into three general categories: new shading and texturing capabilities, anti-aliasing improvements, and tighter specifications. The following table highlights some of the key features in each of these categories, as well as some of the benefits they provide.

	FEATURE	FUNCTION	BENEFITS
Shader and Texture Improvements	*Cube Map Arrays*	Allow reading and writing of multiple cube maps in a single rendering pass	Efficient Global Illumination in real time for complex, dynamic, interactive scenes Enable many ray trace quality effects including indirect lighting, color bleeding, soft shadows, refraction, and high quality glossy reflections
	Separate Blend Modes per-MRT	Allows pixel shaders to output to multiple buffers (MRTs), each with their own blend mode	Efficient Deferred Rendering for improved performance in complex 3D scenes
	Increased Vertex Shader Inputs & Outputs	Doubled from 16 to 32 128-bit values per shader	Improved performance for complex shaders
	Gather4	Allows a 2x2 block of unfiltered texture values to be fetched in place of a single bilinear filtered texture lookup	Improved performance for Stream Computing applications
	LOD instruction	New shader instruction that returns the level of detail for a filtered texture lookup	Custom texture filtering techniques for optimized performance and quality
Improved Anti-Aliasing	*Multi-sample buffer reads and writes*	Allow individual color and depth samples in a multi-sample buffer to be accessed directly by a shader	Custom edge detect filters for high quality anti-aliasing with optimized performance Improved adaptive anti-aliasing performance Improved compatibility of anti-aliasing quality with HDR rendering Improved anti-aliasing compatibility and performance with deferred rendering techniques
	Pixel Coverage Masks	Enables programmable anti-aliasing in a pixel shader	
	Programmable AA Sample Patterns	Gives programmers control over individual sample locations for each pixel	Temporal anti-aliasing Improved image quality for multi-GPU anti-aliasing techniques

	FEATURE	FUNCTION	BENEFITS
Tighter Specification	*FP32 filtering required*	Filtering of 128-bit floating point texture formats now a requirement instead of an optional feature	Encourages use of these high precision data formats by ensuring hardware compatibility
	Int16 blending required	Blending of 64-bit integer pixel formats now a requirement instead of an optional feature	
	Minimum 4x MSAA support required	Multi-sample anti-aliasing with at least 4 samples per pixel must be supported for all 32-bit and 64-bit pixel formats	Ensures anti-aliasing can behave identically on all DirectX 10.1 GPUs Encourages support for anti-aliasing by improving consistency
	Standardized AA sample patterns	Pre-defined sample locations for 2x, 4x, 8x, and 16x AA modes that hardware must support	
	Increased precision for floating point operations	0.5 ULP precision required for all floating point math (add/subtract/multiply/divide) and blending operations	Eliminates rounding errors Matches IEEE standard requirements for these operations

Next-Generation Image Quality

Lighting and Shadows

One discontinuity between real time and non-real time rendering today is the latter's use of global illumination for physically based, realistic lighting. The process of determining the amount and color of lighting reflecting off any given point is critical to giving rendered scenes a sense of depth, and helping viewers gauge the location and movement of objects in 3D space.

An example of this is how indirect or bounced lighting helps you visually place an object in the scene. When one object gets close to another brightly colored object, bounced/reflected light causes color bleeding, and this provides an important visual cue for the closeness of the two objects. So when a white ball is approaching a red wall (for example), the part of the ball facing the wall will start to take on a reddish hue from the light that is bouncing off the wall. The human perceptual system processes this information and intuitively understands that the ball is near the wall. This is important for game play, particularly in our PingPong game, because it enables the player to gain a natural grasp of where the balls are in 3D space. This gives the player a much richer experience, and makes it easier to make strategic gameplay decisions.

This scene from the ATI Radeon PingPong game shows global illumination in action, providing realistic lighting and shadows with thousands of physically simulated objects that the player can interact with. The right hand side of the image shows how the scene looks with global illumination disabled. Using ray tracing techniques to render this type of highly dynamic scene would be practically impossible to achieve in real-time on today's consumer hardware.

There are a number of different lighting methods used in 3D rendering today, including light/shadow mapping, radiosity, and ray tracing. Ray tracing tends to perform better on CPU-like architectures, while light/shadow mapping and certain radiosity techniques tend to perform better on GPU-like architectures.

Light/shadow mapping works by rendering a version of the image from the point of view of each light source to one or more textures. For each pixel, lookups into these textures are performed to determine the amount and color of light contributed by each, and also whether or not each light source is visible from that pixel (if not, it is considered shadowed). Surface shader programs then evaluate and blend all the contributions, taking into account material properties, to determine the final pixel color.

This lighting method is straightforward for capturing standard diffuse lighting, but handling reflections generally requires separate cube maps to be rendered. It is relatively fast and handles dynamic scenes well, but reflection and shadow quality can suffer from limited resolution of light/shadow maps. It also doesn't handle large numbers of point or area light sources well, and it doesn't capture indirect lighting.

Ray tracing works by casting rays out from the viewpoint toward the location of every visible pixel on the screen. It then determines the first point at which each ray intersects an object, and casts rays into each of those points from all directions. A shader is executed at each point to determine the amount and color of light reflecting off of it. This process can be repeated as necessary for multiple bounces to get indirect lighting. It's also good at handling reflection, refraction, and shadows. However, it requires billions of rays per second to be cast to achieve real time frame rates at reasonable resolutions. It also needs complex data structures to store the locations of all objects in the scene, which makes it inefficient for dynamic scenes (since they require these data structures to be updated each frame). Finally, ray tracing is bad at handling shading, since complex surface shaders must be executed repeatedly for each pixel to handle many incoming rays.

Since ray tracing is prohibitively expensive on today's hardware, most games today handle indirect lighting with a "constant ambient term" (i.e. uniform illumination with no particular source that contributes to the entire scene). This is a very rough approximation at best, and it doesn't handle color bleeding or shadows.

Radiosity is another lighting and shadowing technique used in some games today that combines some of the benefits of both light/shadow mapping and ray tracing. It works by using a method similar to ray tracing to pre-compute and store lighting values for a scene, then uses those stored values at render time to allow fast indirect lighting. However, the extensive pre-computation required means that radiosity doesn't work well for dynamic scenes with moving or changing light sources.

Global Illumination

Global illumination is a rendering technique that combines the benefits of light/shadow mapping with indirect lighting and support for practically unlimited dynamic light sources, realistic reflections, and soft shadows. With DirectX 10.1, developers can use indexed cube map arrays and geometry shaders to implement global illumination efficiently in real time, even with thousands of physically modelled objects in a complex, interactive scene.

This technique works by dividing the scene into a 3D array of cubes. For each cube face, a simple version of the scene is rendered from the point of view of someone at the center of the cube looking outward (referred to as a "light probe"). The resolution and detail of these images generally doesn't need to be as high as that of the final image, but is easily scalable according to the level of accuracy and performance required. When the six faces of each cube are complete, they are stored in a cube map texture.

The next step is to convert each cube map into a compressed spherical representation using spherical harmonics. With this new representation, it becomes possible to quickly determine the amount and color of light falling on any point in the cube from any and all directions with a few simple math operations. For points between the light probes, lighting values can be interpolated between values taken from adjacent cube maps.

Adding to the effectiveness of the global illumination is a dynamic ambient occlusion technique. This is an important factor because it captures local visibility changes beyond what is provided by the global illumination alone. In the PingPong game, a dynamic ambient occlusion technique gives the balls soft-edged shadows as they approach each other and as they approach the walls of the scene. These contact shadows are an important visual cue for resolving objects' positions in a scene.

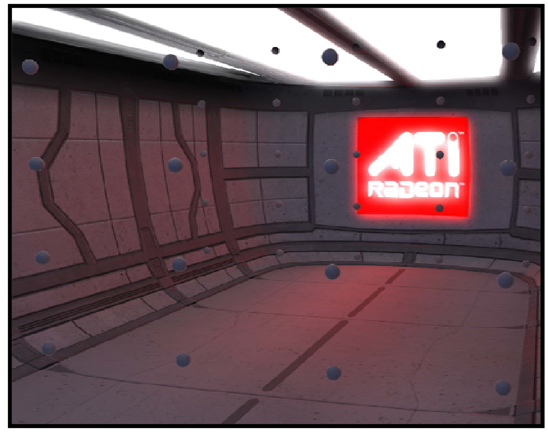


Illustration of light probes used to capture global illumination.

The cube map textures can also be used to create high quality reflections on shiny or glossy objects in the scene. In the PingPong game, looking closely you can see that each of the thousands of balls has a reflection of the environment on its surface.

This method of calculating lighting is highly scalable. The level of quality can be controlled by changing the size and number of cubes used, as well as the level of detail in the cube face images. With DirectX 10.1 cube map arrays, large numbers of cube maps can be rendered to and sampled from simultaneously in parallel, making this technique particularly effective on single GPU or multi-GPU systems.

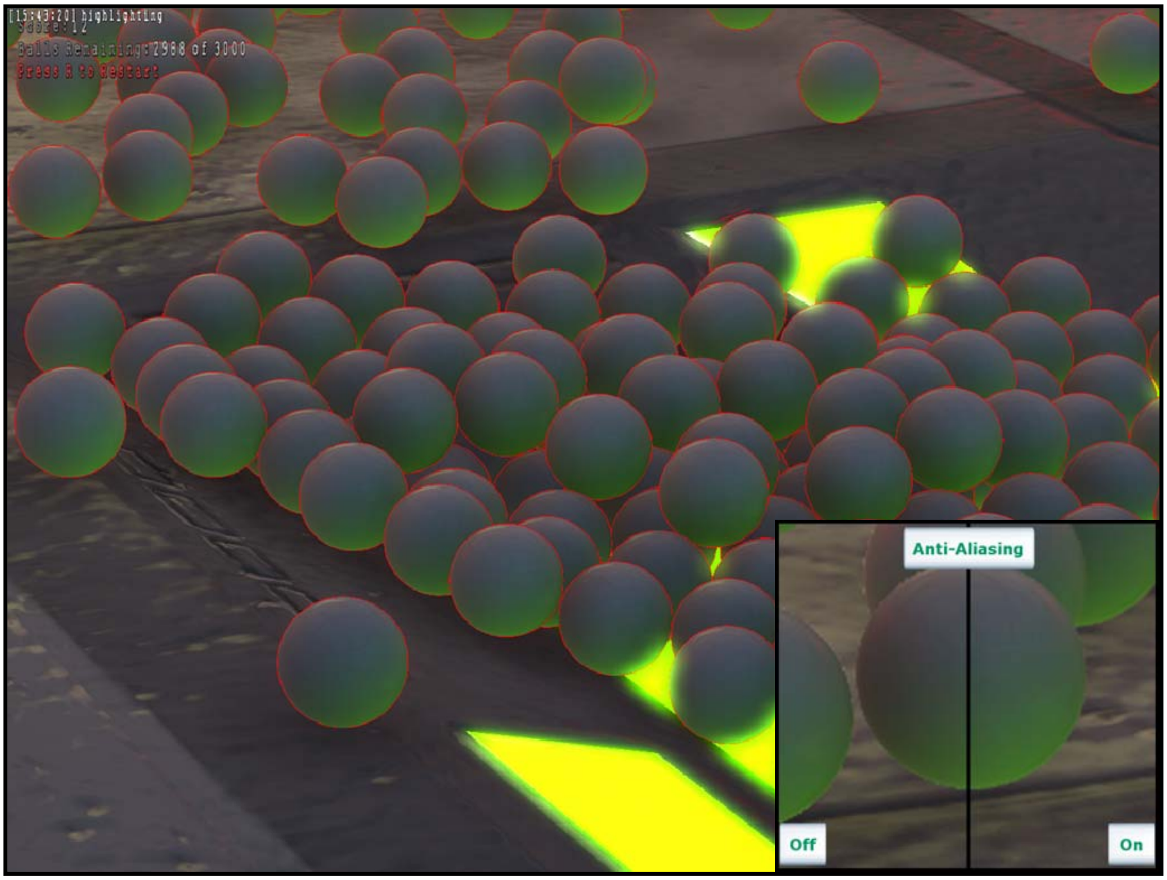
If required for extremely complex scenes, multiple light bounces can easily be simulated by repeating this process and accumulating the results before rendering the final image. Furthermore, the quality of the lighting can be decoupled from the number of objects or polygons in the final scene; in the case of the PingPong game, the number of balls that can be rendered is limited only by the speed of the physics simulation.

In summary, DirectX 10.1 enables a breakthrough in lighting quality with real-time global illumination effects, including indirect lighting, color bleeding, soft shadows, reflection and refraction.

Anti-aliasing Improvements

Jagged, shimmering edges caused by aliasing can be very distracting in rendered images. These artifacts are a consequence of insufficient rendering and/or display resolution. They can be reduced or eliminated using filtering techniques, which work by taking multiple samples per pixel and blending them together. The trick is to choose sample points and weights so as to maximize edge smoothing without blurring out detail, while maintaining good performance.

The most commonly used anti-aliasing technique today is multi-sample anti-aliasing (MSAA), but this only works on polygon edges; it doesn't address texture aliasing or shader aliasing. The ATI Radeon HD 2000 series introduced Custom Filter AA (CFAA), which enabled sophisticated programmable filters. The latest Catalyst drivers support 4 different filter types that can be applied to almost any game using DirectX 9 or earlier. The most advanced of these filters is the edge detect filter, which detects and anti-aliases all edges (including those in textures and shaders).



Edge pixels, detected by the edge detect AA filter and highlighted in red in this image from the ATI Radeon PingPong game, are assigned more samples than other pixels in the image. The inset at the lower right shows how this anti-aliasing removes jagged edges and shimmering artifacts from the edges of the balls; the effects are more pronounced when the image is in motion.

Custom Anti-Aliasing

DirectX 10.1 allows custom anti-aliasing filters to be implemented with pixel shaders. Custom filters can offer improved quality in certain cases where standard MSAA can have issues, such as with HDR lighting and deferred shading techniques. All DirectX 10.1 compatible hardware must support a minimum of 4x MSAA. The specification now includes some pre-defined AA sample patterns, in contrast to earlier versions of DirectX where they were left entirely up to each particular GPU to define.

A new feature of DirectX 10.1 allows all AA buffers to be accessed directly by shaders. Previously, it was only possible to access multi-sampled color buffers; it was impossible to access information from a depth buffer for each sample individually This allows developers to implement more advanced custom AA techniques using a combination of shaders and dedicated hardware, much like ATI Radeon HD GPUs do today with CFAA.

ATI Radeon HD also introduced support for Adaptive Anti-Aliasing, which provides texture anti-aliasing for partially transparent textures (such as foliage and chain-link fences). DirectX 10.1 expands on this capability by introducing sample coverage masking, which provides control over the specific sample locations where pixel shaders are executed. This allows developers to extend the Adaptive Anti-Aliasing technique to address more types of aliasing artifacts. Custom sample patterns can also be specified to complement the basic set that must be supported by all compatible hardware. Many of these capabilities were already present in previous generations of ATI Radeon GPUs, but the DirectX 10.1 allows them to be exposed directly to developers for the first time.

In summary, DirectX 10.1 finally gives developers the tools they need to eliminate all types of aliasing artifacts from interactive real-time games, delivering a major increase in image quality.

Tighter Specification

GPU compatibility issues have historically been a significant roadblock that has slowed the adoption of new 3D features by developers. These arise when certain elements of the programming interface they are using are interpreted slightly differently by different GPUs, causing unexpected performance drops, image quality problems, error messages, or even crashes. Rather than spend time working around these problems, many developers preferred to avoid them entirely by targeting lowest common denominator GPU feature sets.

DirectX 10 made major strides toward eliminating these issues by more tightly defining the required GPU behavior for each function and instruction, and greatly reducing the number of optional features that might be present on one DirectX 10 GPU but not present on another. Many of the improvements in DirectX 10.1 were introduced to take the API even further down this path.

New texture format requirements

One obstacle that has prevented developers from using the higher precision texture and output formats in recent versions of DirectX has been the limitations in what operations each GPU can perform on them. DirectX 10.1 improves this by requiring all compatible GPUs to support texture filtering of 32-bit floating point formats, and blending operations on 16-bit integer formats.

New multi-sample anti-aliasing requirements

Multi-sample anti-aliasing is a well established technique for improving image quality. However, over the years, many enhancements have been made to the basic technique. Since these modifications can yield significantly different output on different GPU models, developers often tended to avoid supporting them directly in their games. DirectX 10.1 mandates a minimum anti-aliasing quality requirement as well as a set of pre-defined sample patterns. This ensures that a consistent, high quality level for anti-aliasing is supported on all compatible GPUs, while still supporting new techniques on individual GPUs.

Higher precision requirements

All data formats have a limited amount of precision they can support, which depends on the number of bits available. However, operations done on these formats do not necessarily produce output that takes full advantage of all the available precision; in some cases, approximations are used that can cause rounding errors in the least significant bits of the output. This practice can cause unpredictable behavior in some cases (for example, when errors build up due to iterating an operation many times). DirectX 10.1 addresses these issues by mandating that basic math operations on floating point values up to 32-bits take advantage of the full precision available, thus ensuring identical results down to the last bit on all compatible GPUs (and even CPUs as well).

Conclusion

DirectX 10.1 offers incremental improvements to the programming interface that address limitations of DirectX 10, and unlock new graphical techniques that will take the quality of 3D graphics to the next level in 2008 and beyond. Advantages include global illumination delivering lighting and shadow quality in real-time that matches the ray tracing techniques used in CG films, improved anti-aliasing techniques to clean up distracting shimmering artifacts, and tighter specifications for improved compatibility. The ATI Radeon HD 3800 series products are the world's first GPUs to bring these features and benefits to the PC.

Disclaimer

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.