AMD FirePro Technology SDK Code Samples

AMD FirePro SDK is a repository of samples with complete source code intended for educational purpose. The base code is up to date with the latest specification and can be seen as a guideline for good coding practice, to achieve the best performance from your graphic card.

In each release of our SDK, you will find numerous code samples, complete with straightforward source code, whitepapers, and/or videos to help you take advantage of the latest technology from AMD FirePro professional graphic cards.

This code is released free of charge for use in derivative works, whether academic, commercial, or personal.

We appreciate and value your feedback, feel free to email us. The source of the SDK is also available in a GitHub repository.

The FirePro SDK Beta 0.9.0 has been successfully tested with *Catalyst 8.982.2*. Please, ensure that the drivers are updated before running the samples. For the moment, it can only be used under Windows and requires CMake to create a solution. Visual C++ 2005 to 2012 and both 32 bit and 64 bit builds are supported.

The samples below using OpenGL follow the specification as of August 2012. They favor using OpenGL® 4.2 core profile contexts and avoid using deprecated features.

Some sample below use OpenCL™ (Open Computing Language) for general-purpose computations on heterogeneous systems. OpenCL™ allows programmers to preserve their expensive source code investment and easily target multi-core CPUs, GPUs, and the new APUs.

Textured Cube

OpenGL

This sample is the rendering of a simple cube. The cube is created through an index buffer and a vertex buffer. The rendering is achieved by a Shader program.

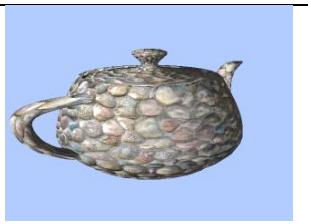
AMD FirePro V series+
AMD Radeon HD 5000 series+

Lighting

OpenGL

This sample shows how to load a 3D model and render it. For the loading, we use the Assimp library.

The rendering is a classical tangent space lighting method that uses normal and specular maps.



AMD FirePro V series+
AMD Radeon HD 5000 series+

Advanced Rendering Techniques: OIT, SSAO, Über-shader

OpenGL

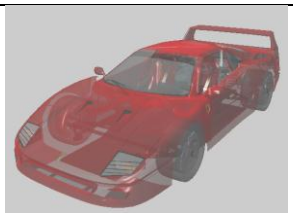
This sample exhibits several technics introduced by games which are now very popular in Computer Aided Design (CAD) and Digital Content Creation (DCC) applications and in particular:

- Order Independent transparency (*OIT*) is a rendering technique that allows rendering overlapping semi-transparent objects without having to sort them.
  OIT is entirely processed on the GPU and doesn't tax the existing pipeline.

- Screen-Space Ambient Occlusion (*SSAO*) is a shading method that increases realism by taking into account attenuation of light due to local occlusion.

- Self-Shadowing, blurring and glowing effects highlight the advantage of off-screen rendering for multiple special effects.

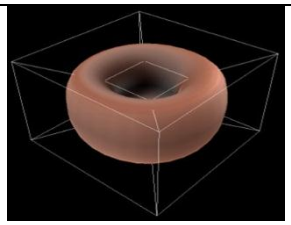
- Usage of a single unique shader (*über-shader*) containing all the effects allowing smooth transition between them.



AMD FirePro V series+
AMD Radeon HD 5000 series+

Tessellation

OpenGL



AMD FirePro V series+
AMD Radeon HD 5000 series+

In Computer Aided Design (CAD) and Digital Content Creation (DCC) applications, Non-Uniform Rational B-Spline (NURBS) surfaces are the most common representative of parametric surfaces. Tensor product Bicubic Bézier patches are a special case of NURBS surfaces. In fact, any regular face in an input mesh can be converted into a degree 3 by 3 patches in tensor-product Bézier form by the standard B-spline to Bézier conversion rules.

For its simplicity and popularity, this sample demonstrates how to create and adaptively evaluate Bézier patches from input polygonal mesh by leveraging tessellation feature introduced in OpenGL 4.0.

The sixteen control points of a single bicubic Bézier patch are computed in tessellation control shader stage. Each Bézier patch is then adaptively evaluated in tessellation evaluation shader using those control points. The implementation is applicable for interactive picking, animation or deformation of the parametric surfaces because each parametric patch is dynamically generated per frame.

Transfer Compute Overlap

OpenCL

AMD FirePro W series+
AMD Radeon HD 7000 series+

This OpenCL sample demonstrates how transfer of data between the host and the device can be overlapped with computations made on the device. We make use of up to 3 command queues dedicated to the specific tasks of moving data from the host to the device and vice-versa or executing an OpenCL kernel on the device.

- With 2 command queues (MAX_Q=2), the 1st one manages the computation, the 2nd deals with all sort of data transfers.
- With 3 command queues (MAX_Q=3), the data transfers are managed by 2 different queues based on the direction of the data movement.
- The OpenCL kernel performs a computation of sin(sin(sin(…))) on the corresponding data and the result is sent back to the host. The number of nested sin() calls depends on the MAXCOMP parameter.
- The amount of data moved forth and back and used in the computation varies from 1Kbyte to 16Mbytes.
- The overall performance is measured in term of aggregate bandwidth to and from the device.

QuadBuffer Stereo Rendering

OpenGL

AMD FirePro V series+

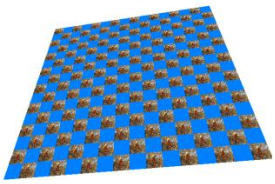
Many Computer Aided Design (CAD) and Digital Content Creation (DCC) applications have a stereo feature. The classic method to get stereo is to draw twice the geometry, once for the left eye and once for the right eye. Therefore the application is emitting twice the number of draw calls than usual, which can affect the performance since the driver has then twice the number of command to process.

This sample exhibits an advanced technics to get a stereoscopic content without having to draw twice from the CPU. It uses a geometry shader to generate the right and left view so it has only one draw per geometry mesh.

The two different views are drawn in different layer of the texture-array bound to the FBO we are drawing to.
Then the sample is drawing the result of that texture in the back left and back right buffer.

Sparse Texture

OpenGL



AMD FirePro W series+
AMD Radeon HD 7000 series+

This sample is presenting the *GL_AMD_sparse_texture* extension introduced by AMD FirePro W and AMD Radeon HD 7000 series

Recent advances in application complexity and a desire for higher resolutions have pushed texture sizes up considerably. Often, the amount of physical memory available to a graphics processor is a limiting factor in the performance of texture-heavy applications. Once the available physical memory is exhausted, paging may occur bringing performance down considerably - or worse, the application may fail. Nevertheless, the amount of address space available to the graphics processor has increased to the point where many gigabytes of address space may be usable even though that amount of physical memory is not present.

This extension allows the separation of the graphics processor's address space (reservation) from the requirement that all textures must be physically backed (commitment). This exposes a limited form of virtualization for textures. Use cases include sparse textures, texture paging, on-demand and delayed loading of texture assets and application controlled level of detail.

Framelock

OpenGL

This sample shows how to enable Framelock in an application using the *WGL_NV_swap_group* extension.

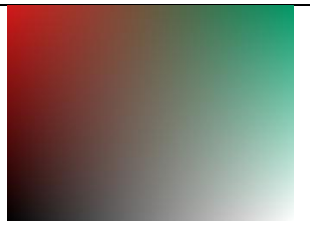
Framelock is available on systems that have their AMD FirePro GPUs connected to an ATI FirePro S400 board. Framelock can be used to synchronize the SwapBuffers of several application instances running on different GPUs. Those can be either multiple GPUs in one system or multiple systems each with one or more GPUs. Having Framelock enabled will ensure that all participating GPUs will execute the SwapBuffer of the application at exactly the same time.

AMD FirePro V series+

10 bit

OpenGL

Sample OpenGL source code that illustrates how to create and display 10-bit per component surfaces.



AMD FirePro V series+