AMD Accelerated Parallel Processing TECHNOLOGY

SAMPLE

Gaussian Noise with GL Interoperability

1 Overview

1.1 Location

`$<AMDAPPSDKSamplesInstallPath>\samples\opencl\cl\1.x`

1.2 How to Run

See the *Getting Started* guide for how to build samples. You first must compile the sample.

Use the command line to change to the directory where the executable is located. The pre-compiled sample executable is at
`$<AMDAPPSDKSamplesInstallPath>\samples\opencl\bin\x86\` for 32-bit builds, and
`$<AMDAPPSDKSamplesInstallPath>\samples\opencl\bin\x86_64\` for 64-bit builds.

Type the following command(s).

1. `GaussianNoiseGL`
   This generates Gaussian noise in the input image.
2. `GaussianNoiseGL -h`
   This prints the help file.

1.3 Command Line Options

Table 1 lists, and briefly describes, the command line options.

Table 1 Command Line Options

Short Form	Long Form	Description
-h	--help	Shows all command options and their respective meaning.
	--device	Devices on which the program is to be run. Acceptable values are cpu or gpu.
-q	--quiet	Quiet mode. Suppresses all text output.
-e	--verify	Verify results against reference implementation.
-t	--timing	Print timing.
	--dump	Dump binary image for all devices.
	--load	Load binary image and execute on device.
	--flags	Specify compiler flags to build kernel.
-p	--platformId	Select platformId to be used (0 to N-1, where N is the number of available platforms).
-d	--deviceId	Select deviceId to be used (0 to N-1, where N is the number of available devices).
-v	--version	AMD APP SDK version string.
-i	--iterations	Number of iterations for kernel execution.
-f	--factor	Noise factor.

2 Introduction

Gaussian noise is statistical noise that has a probability density function of the normal distribution (also known as Gaussian distribution). The values that the noise can take on are Gaussian-distributed.

This sample takes an input image and generates a Gaussian deviation by using the pixel value as a seed. This deviation is then added to all the components of the pixel.

When running the GaussianNoiseGL sample, the output image appears in a window. To change the clarity of the output image, use the ‘w’ key on the keyboard to increase the factor by +2; use the 's' key to change the factor by -2.

3 Implementation Details

Each thread generates two uniform random numbers in the range (0, 1), using a linear congruential generator function.

A minimal standard linear congruential generator proposed by Park and Miller (see reference [1]) is:

$I_j + 1 = a\, I_j \bmod m$

where a = 16807 (7^5), and m = $2^{31} - 1$.

To implement this, we use Schrage’s method (see reference [2]), which is based on an approximate factorization of m.

m = aq + r, that is: q = [m/a], r = m mod a

We then apply a shuffling algorithm by Bays and Durham, as described in Knuth (see reference [3]), to remove low-order serial correlations.

A Box-Muller transform is then applied to obtain the numbers in the Gaussian distribution. This takes two uniform samples, u_0 and u_1, and transforms them into two Gaussian distributed samples, r_0 and r_1, using the following relations.

$$
\begin{aligned}
r_0 &= \sin(2\pi u_0)\sqrt{-2\log(u_1)},\\
r_1 &= \cos(2\pi u_0)\sqrt{-2\log(u_1)}.
\end{aligned}
$$

This method, which is the simple version of this transform, is suitable for GPUs because it is mathematically intensive and free of loops and branches.

Another version of this transform, called the Polar form, relies on looping, which is less efficient on GPUs. The Polar form uses rejection to discard numbers, as shown in the following code sample.

```
float x1, x2, w, y1, y2;

do {
        x1 = 2.0 * ranf() - 1.0;
        x2 = 2.0 * ranf() - 1.0;
        w = x1 * x1 + x2 * x2;
} while ( w >= 1.0 );

w = sqrt( (-2.0 * ln( w ) ) / w );
y1 = x1 * w;
y2 = x2 * w;
```

Using this form results in reduced performance compared to the simple (Box-Muller) version.

When running the GaussianNoiseGL sample, the output image appears in a window. To change the clarity of the output image, use the 'w' key on the keyboard to increase the factor by +2; use the 's' key to change the factor by -2.

4 Environment

This Sample must run in the OpenCL 1.2 environment. The following APIs are part of OpenCL 1.2:

- `clLinkProgram()`

  Links a set of compiled program objects and libraries for all the devices, or a specific device(s) in the OpenCL context, and creates an executable.

- `clGetExtensionFunctionAddressForPlatform()`

  Returns the address of the extension function by a given name.

- `clCreateFromGLTexture()`

  Creates an OpenCL image object, image array object, or image buffer object from an OpenGL texture object, texture array object, texture buffer object, or a single face of an OpenGL cubemap texture object.

- `clCompileProgram()`

  Compiles a program's source for all the devices, or one or more specific devices.

5 References

1. Park, S.K., and Miller, K.W 1988, *Communications of the ACM*, vol. 31, pp., 1192-1201.
2. Schrage, L. 1979, *ACM transactions on Mathematical Software*, vol. 5, pp. 132-138.
3. Knuth, D.E, 1981, Seminumerical Algorithms, 2nd ed., vol. 2 of *The art of computer programming*, 3.2-3.3.

Contact

Advanced Micro Devices, Inc.
One AMD Place
P.O. Box 3453
Sunnyvale, CA, 94088-3453
Phone: +1.408.749.4000

For AMD Accelerated Parallel Processing:
URL: developer.amd.com/appsdk
Developing: developer.amd.com/
Support: developer.amd.com/appsdksupport