AMD Accelerated Parallel Processing TECHNOLOGY

# SAMPLE

# Box Blur-Filter

## 1 Overview

### 1.1 Location

`$(AMDAPPSDKSAMPLESROOT)\samples\opencl\cl\app`

### 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 `$(AMDAPPSDKSAMPLESROOT)\samples\opencl\bin\x86\` for 32-bit builds, and `$(AMDAPPSDKSAMPLESROOT)\samples\opencl\bin\x86_64\` for 64-bit builds.

Type the following command(s).

1. `BoxFilter`
   This applies a box blur filter on the input image.
2. `BoxFilter -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 the 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. |
| `-x` | `--width` | Filter width. |

# 2 Introduction

Box filtering, also known as average or mean filtering, is a method of reducing the intensity variation between pixels in an image, and is a commonly used technique to reduce noise.

# 3 Implementation Details

Two versions of Box filter have been implemented –

1. BoxFilter separable
2. BoxFilter with precomputed summed area tables

## 3.1 BoxFilter Separable

Filtering an M-by-N image with a P-by-Q filter kernel requires roughly MNPQ multiplies and adds.

If the kernel is separable, you can filter in two steps. The first step requires about MNP multiplies and adds. The second requires about MNQ multiplies and adds, for a total of MN(P + Q).

Implementation consists of applying the filter horizontally, then vertically.

Figure 1 compares the performance between a naïve and a separable BoxFilter.

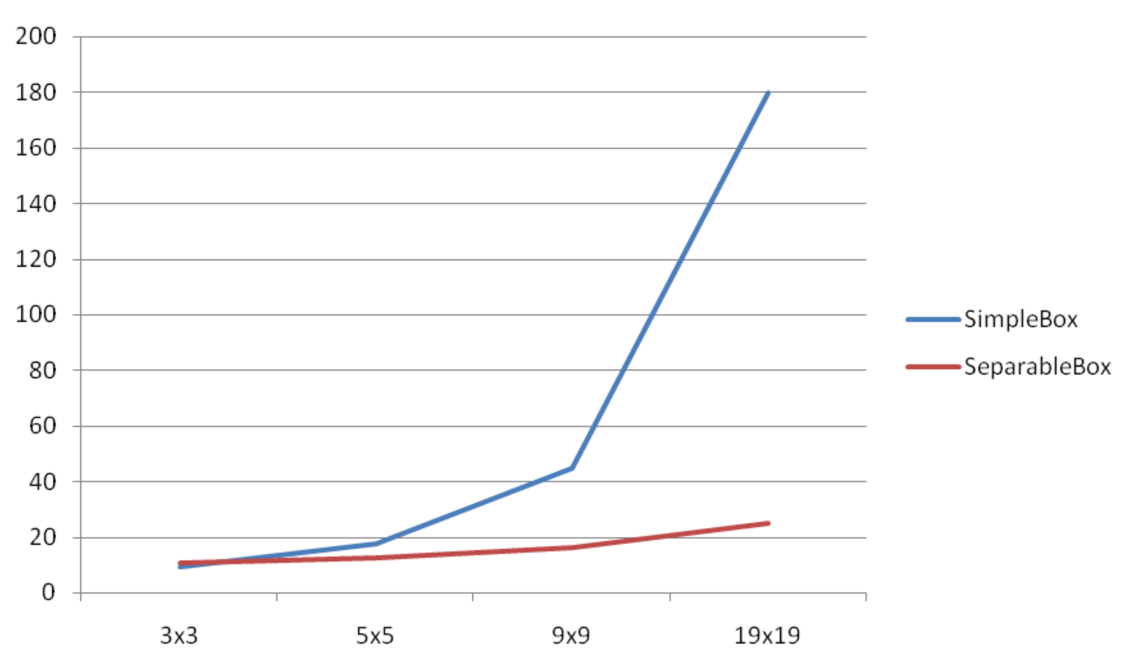


**Figure 1 Time Taken (in msec) vs Filter Size on an ATI Radeon™ HD 4870 GPU**

Loop unrolling optimization results in approximately 10% to 20% improvement, depending on the specified filter width. This means approximatelhy 180 fps for a 1024 x 1024 image (filter size 9x9) on an ATI Radeon™ HD 5770 with GL interoperability.

The flag for using hardware local memory in the horizontal pass of the separable filter has been commented out due to performance decrease.

### 3.2 BoxFilter with SAT

Summed-area tables (SATs) were introduced by Crow (see reference [1]) to accelerate texture filtering. Each element in a SAT is the sum of all texture elements in the rectangle above and to the left of the element (see Figure 2).

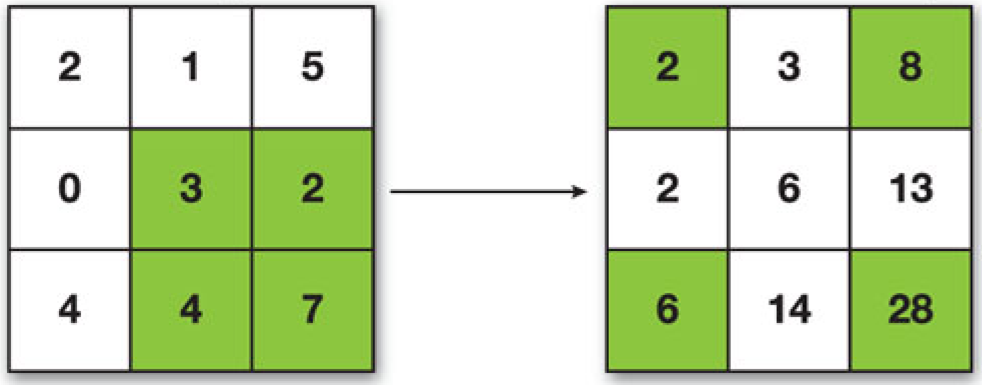


**Figure 2 Sample Data and Corresponding SAT**

The sum of any rectangular region then can be determined in constant time using:

$$s = t\,[x_{max}, y_{max}] - t\,[x_{max}, y_{min}] - t\,[x_{min}, y_{max}] + t\,[x_{min}, y_{min}]$$

It is easy to compute the average over this region by dividing by the number of pixels in the region.

SATs let us sample arbitrary rectangular regions, which is sufficient for applying a box filter of any size on an image.

### 3.3 Computing SAT

Computing a SAT is done in two passes.

1. Horizontal pass – the prefix sum is applied on each row separately.
2. Vertical pass – the prefix sum is applied on column separately.

After computing a SAT, a final BoxFilter kernel requires fetching only four values from a global buffer to compute the final filtered image.

This technique is very fast for interactive applications because, after applying the precomputation, it is possible to change the filter size immediately without degrading performance. For example, a 1024 x 1024 image gives us about 250 fps using GL interoperability on the ATI Radeon™ HD 5770, irrespective of box filter size.

Note that the value of the sums (and, thus, the dynamic range) can become very large; the table entries require extended precision. The number of bits of precision needed per component is calculated using:

$$P_s = \log_2 (w) + \log_2 (h) + P_i$$

where: w and h are the width and height, respectively, of the input image.
$P_s$ is the precision required to hold values in the SAT.
$P_i$ is the number of bits of precision of the input.

Given this, a 256 x 256 texture with eight-bit components requires a SAT with 24 bits of storage per component. Thus, we use 32-bit per pixel image data (12 + 12 + 8 = 32) for the calculation of the SAT. This can maximally process a 4096 x 4096 image.

# 4 References


1. Crow, Franklin (1984). "Summed-area tables for texture mapping". *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pp. 207–212.



**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
**Forum:** developer.amd.com/openclforum