AMD **Accelerated** Parallel Processing TECHNOLOGY

SAMPLE

BOLT Box Blur-Filter

1 Overview

1.1 Location

`$<APPSDKSamplesInstallPath>\samples\bolt\`

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 `$<APPSDKSamplesInstallPath>\samples\bolt\bin\x86\` for 32-bit builds, and `$<APPSDKSamplesInstallPath>\samples\bolt\bin\x86_64\` for 64-bit builds.

Type the following command(s).

1. `BoxFilterSAT`
   This command applies a box blur filter on the input image, using the default options.
2. `BoxFilterSAT -h`
   This command prints the help file.
3. `BoxFilterSAT_TBB -h`
   This command generates a build with the multiCoreCpu path (the Thread Building Block library), enabled.

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`	Explicit device selection for Bolt [auto/openCL/multiCoreCpu/SerialCpu]
`-q`	`--quiet`	Quiet mode. Suppresses most text output.
`-e`	`--verify`	Verify results against reference implementation.
`-t`	`--timing`	Print timing.
`-v`	`--version`	AMD BOLT library and run-time version string.
`-i`	`--iterations`	Number of iterations for kernel execution.
`-w`	`--width`	Filter width.

Note: The `--device multiCoreCpu` option becomes available when the sample is compiled with `ENABLE_TBB` defined. Microsoft Visual Studio build configurations `Debug_TBB` and `Release_TBB` are created for this purpose. These configurations have `ENABLE_TBB` defined to enable the TBB path (multiCoreCpu) for all the AMD BOLT functions used in the sample.

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. The Box filter can implemented using different techniques. This sample uses the precomputed Summed Area Tables (SAT) technique.

3 Implementation Details

3.1 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, as shown in Figure 1.

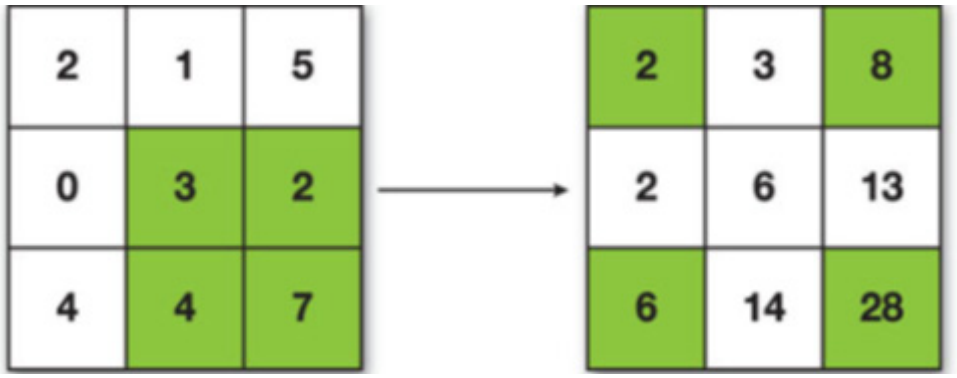


Figure 1 Sample data and the corresponding SAT

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

```
s = t [xmax, ymax] - t [xmax, ymin] - t [xmin, ymax] + t [xmin, ymin]
```

The average over this region can be computed 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.

3.2 Computing the SAT

Computing the 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 each 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 pre-computation, the filter size can be changed immediately without degrading performance.

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 the following formula:

```
Ps = log2 (w) + log2 (h) + Pi
```

In the above formula,

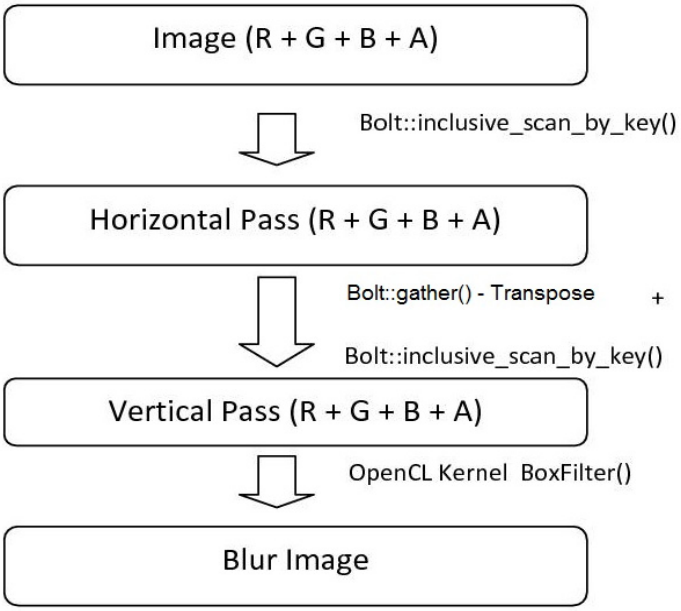
- `w` and `h` are the width and height, respectively, of the input image.
- `Ps` is the precision required to hold values in the SAT.
- `Pi` is the number of bits of precision of the input.

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

3.3 BoxFilter using BOLT APIs

Box filter is applied on each component (R, G, B, and A) of the pixels of the input image. As discussed in the preceding section, generating the SAT requires two passes, one for the horizontal scan and the other for the vertical scan.

The i`nclusive_scan_by_key()` BOLT API is used for performing the horizontal scan and it uses a key buffer for each pixel (the key is the same as the row number). For the vertical scan, the result matrix from the horizontal scan is first transposed and is then followed by the prefix scan. The BOLT `gather()` API has been used for transpose operation and the transposed matrix is passed to the next scan operation. The following flowchart illustrates the workflow.



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