Application Report
SPRAB78–November 2009

Canny Edge Detection Implementation on TMS320C64x/64x+ Using VLIB

Y Senthil Kumar

ABSTRACT

This application note contains instructions for implementing Canny Edge Detection algorithm using VLIB. The general ideas given here can also be applied for an application which calls a sequence of VLIB functions in a block-based manner.

Contents

List of Figures

List of Tables

1 Introduction

Canny edge detector is the optimal and most widely used algorithm for edge detection. Compared to other edge detection methods like Sobel, etc canny edge detector provides robust edge detection, localization and linking. It is a multi-stage algorithm and the stages involved are illustrated in Figure 1. Thus, instead of providing the whole algorithm as a single API, kernels are provided for each stage. This way, the user can have more flexibility and better buffer management. For more details on the theory and formulation please refer to the Canny edge detection paper [1].

Optimized kernels and wrappers required for the implementation of canny algorithm are provided to reduce the efforts of the integrator. This document is to aid the integrator to use the kernels, understand the algorithm implemented and use the appropriate parameters. Basic image processing and C64x programming knowledge are assumed. For a formal description of the APIs for these kernels, refer to the VLIB documentation (Section 16 through Section 19) [3].

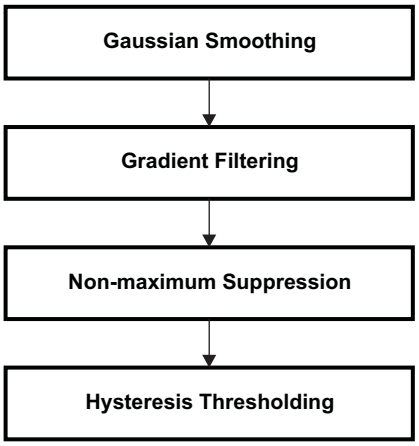

Figure 1. Flow Diagram of Canny Edge Detector

2 Canny Edge Detector Algorithm

The kernels involved in the canny edge detector algorithm are discussed in detail in this section. At each stage, for computing the output pixel at a particular row, we need input pixels at rows below and above. Thus, output at the first and the last row are undefined. The same happens in the case of columns too. Thus, the size of the valid pixels in the output reduces after each step. To incorporate this, the output width and height and the output buffer's position changes after each step. This is illustrated in each stage as API argument adjustments. This API adjustment technique is the same for other applications where a sequence of VLIB functions are called.

2.1 Noise Reduction

Noise brings about high gradient magnitudes which in turn produces unintended edges. To reduce this effect, the image is convolved with a 2D Gaussian filter, which brings each pixel in closer harmony with its neighbors. This is basically a smoothing process. Practically, a discrete Gaussian function requires a window size of *5*sigma* approximately (where sigma is the standard deviation of Gaussian function). Sigma is the parameter which controls the smoothing, which in turn boils down to window size. Larger window sizes are generally not recommended because it is computationally very expensive and also causes over smoothing, removing weak edges. A window size of 5x5 or 7x7 is recommended. The Gaussian filters can be generated using the matlab command *fspecial*.

For the implementation of this step on the C64x, optimized 2D convolution functions *IMG_conv_3x3_i8_c8s*, *IMG_conv_5x5_i8_c8s* and *IMG_conv_7x7_i8_c8s* can be used depending on the window size (*conv_window_size*) required. Due to the edge effects of convolution, the output image's *height* and *width* decreases by (*conv_window_size-1*). Care is to be taken in using the modified height and width for the next step according to the *conv_window_size* used. Please refer to IMGLIB documentation [2] for more details on the usage of the above convolution functions.

The required API argument adjustments are illustrated in Equation 1. The parameter *conv_window_size* is the window size of the convolution kernel.

$$
\begin{aligned}
&pBufIn = input;\\
&pBufOut = putput + width * (conv\_window\_size - 1)/2 + (conv\_window\_size - 1)/2;\\
&pGx = pBufGradX + width * (conv\_window\_size - 1)/2 + (conv\_window\_size - 1)/2;\\
&pGy = pBufGradY + width * (conv\_window\_size - 1)/2 + (conv\_window\_size - 1)/2;\\
&pMag = pBufMag + width * (conv\_window\_size - 1)/2 + (conv\_window\_size - 1)/2;\\
&output\_height = height - (conv\_window\_size - 1);\\
&output\_width = width - (conv\_window\_size - 1);
\end{aligned}
\tag{1}
$$

API Usage:

```
IMG_conv_7x7_i8_c8s( pBufIn,
                     pBufOut,
                     output_width* output_height,
                     width,
                     gaussian_7x7,
                     8 );
```

It is to be noted that Gaussian filter is a spatially separable kernel. Thus the 2D convolution can be performed as 1D row and column convolutions which saves *conv_window_size*/2 computations for each output calculation. But for smaller window sizes (<7) this does not give significant improvement in performance on the C64x.

2.2 Gradient Filtering

This kernel computes the first order horizontal and vertical gradients, and computes the magnitude of the gradient using L1 norm. This step creates an additional 1-pixel border of invalid data around the input image. As before, care has to be taken to use the modified height and width for the next step. The calculation steps used in the algorithm are listed in Equation 2.

$$
\begin{aligned}
&Gx = I(x+1,y) - I(x-1,y)\\
&Gy = I(x,y+1) - I(x,y-1)\\
&Gmag = (|Gx| + |Gy|)
\end{aligned}
\tag{2}
$$

The required API argument adjustments are illustrated in Equation 3.

$$
\begin{aligned}
&pGx+ = (width + 1);\\
&pGy+ = (width + 1);\\
&pMag+ = (width + 1);\\
&output\_height - +2;
\end{aligned}
\tag{3}
$$

API Usage:

```
VLIB_xyGradientsAndMagnitude( pBufOut,
                              pGx,
                              pGy,
                              pMag,
                              width,
                              output_height );
```

2.3 Non-Maximum Suppression

The edge magnitude image may contain wide ridges around the local maxima which are visualized as thick edges. Non-maximum suppression produces thin edges removing the non-maxima pixels along the normal direction preserving the connectivity of contours. At each pixel location, two virtual points lying along the normal direction on either side of the current location are interpolated using the gradient magnitudes from surrounding neighbors. If the gradient magnitude of the current position is greater than those at the virtual points, it is declared a possible edge by giving a value of 127. Else it is made 0. The direction estimation and interpolation steps were combined to eliminate division, and hence, removing the reciprocal tables required for division. Finally, the output is binary valued containing either 0 or 127. This step creates another 1-pixel border of invalid data around the image.

The required API argument adjustments are illustrated in Equation 4.

$$
\begin{aligned}
&pGx+=(width+1);\\
&pGy+=(width+1);\\
&pMag+=(width+1);\\
&pBufOut+=(2*width+2);\\
&output\_width-=4;\\
&output\_height-=2;
\end{aligned}
\tag{4}
$$

API usage:

```
VLIB_nonMaximumSuppressionCanny( pMag,
                                 pGx,
                                 pGy,
                                 pBufOut,
                                 output_width,
                                 width,
                                 output_height );
```

2.4 Hysteresis Thresholding

This stage of the algorithm is split into a block based (*VLIB_doublethresholding*) and a non-block based (*VLIB_edgeRelaxation*) kernel. This is done to give the flexibility of using a part of this stage in a block based manner. Double thresholding uses two parameters *highthreshold* and *lowthreshold*. If the gradient magnitude is above high threshold and it is a possible edge, then it is declared a strong edge and a value of 255 is stored. Its location is stored in a dynamic list where the size of the list grows based on the edge structure.

The API usage is illustrated below. Buffer listptr holds the positions of strong edges. A buffer used for pGx or pGy can be re-used for this buffer.

```
VLIB_doublethresholding( pMag,
                         pBufOut,
                         listptr,
                         numItems,
                         output_width,
                         width,
                         output_height,
                         loThres,
                         hiThres
                         0);
```

The second kernel traverses each strong edge and updates the list with a weak edge if it is a neighbor. Thus, it is not block based as it traverses through the image arbitrarily. The performance of this kernel depends on the number of strong edges and weak edges, as well as their organization. Usually, the percentage of edges in an image is small, thus this kernel runs for less time.

```
VLIB_edgeRelaxation( pBufOut,
                     listptr,
                     numItems,
                     width );
```

3 Wrapper Function

Two implementations of wrapper function for Canny are provided; namely, Frame based and block based processing implementations.

3.1 Frame Processing

The first implementation contains a straight forward usage of the kernels. As it is not complicated, it is easier to understand and get started. This implementation is suitable for small images that can fit in the L1 memory. Code segments shown before as API adjustments were taken from *VLIB_testCannyEdgeDetector.c*, which is present in INSTALL_DIR\VLIB_V_2_1\examples\src.

3.2 Block Processing

For larger images, all the buffers will not fit L1 memory. Thus, we have to load blocks of the image into L1 memory from DDR and do the processing of each stage. We provide a *BlockBasedCanny.c* file to illustrate this block based processing of Canny Edge Detector Algorithm. The first four kernels are used in a block based manner. The last kernel *VLIB_edgeRelaxation* cannot be implemented in a block based manner. Thus, it is run on the whole image.

The first three kernels reduce the output size by 2 each (for *conv_window_size*=3). Thus, seven lines of input are required for the computation of one line of output. From these seven lines of input, we obtain five lines of Gaussian smoothed output, three lines of Gradient filtered output and one line of non-maximum suppressed output. This method is required only for the first output line computation. For the computation of the next output line, we need the next seven lines where the first six lines were used for previous computation. By trying to use the previous computations, we can reduce the computation for the next output line and figure out a steady state scheme.

In steady state, we need to allocate only four lines of memory for input, Gaussian smoothed output, Gradient filtered output and Non-maximum suppressed output buffers. The scheme is illustrated in Figure 2. In each iteration, four lines of input are used to compute two lines of Gaussian smoothed output and then the last two lines shaded in red are moved to the first two lines of the input buffer for the next iteration. Now for the next iteration, only two new input lines are loaded to the last two lines. Similarly in the Gaussian smoothed output buffer, two lines are reused from previous iteration and two lines are computed. The same logic is extended for the subsequent stages.

This block based version of the code can be easily plugged into a ping-pong buffering scheme using EDMA drivers.

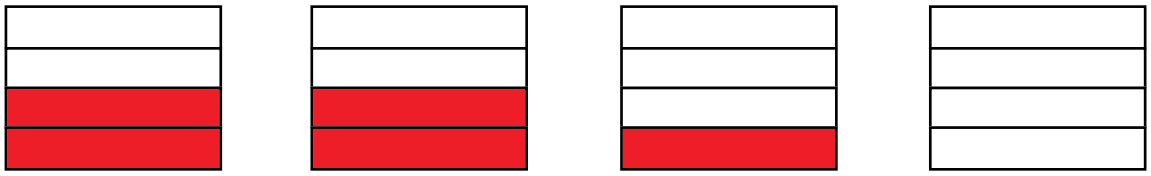

Figure 2. The Sequence of Block Outputs After Each Stage

In Figure 2, the sequence of block outputs after each Stage: four rows of Input block, Gaussian Smoothed block, Gradient Filtered block and Non-maximum suppressed block. In steady state, all these steps produce four rows of output. Red filled rows are re-used for the next iteration.

4 Experiments and Results

Canny edge detector algorithm explained as above was run on the DM6437 EVM and the results after each step is shown in Figure 3. The performance numbers for each kernel and its parameters involved are tabulated below.

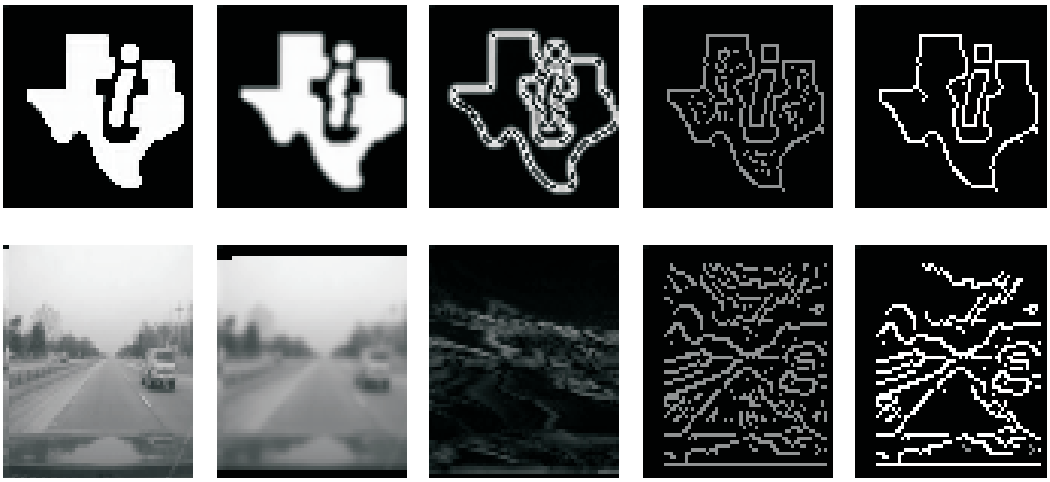

Figure 3. The Sequence of Image Outputs After Each Stage

In Figure 3, the sequence of image outputs after each Stage: Input Image, Gaussian Smoothed Image, Gradient Filtered Image, Non-maximum suppressed Image, Final Edge Output

Table 1. Performance Figures on DM6437 EVM

Function	Cycles Per Pixel	Parameters
VLIB_gaussianFilter7X7	1.5(3x3), 4.25 (5x5), 7.5(7x7)	conv_window_size
VLIB_xyGradientsAndMagnitude	0.8	
VLIB_nonMaximumSuppressionCanny	8.7	
VLIB_doublethresholding	2.4	High and low threshold
VLIB_edgeRelaxation	0.6-3.6(1)	

(1) Input dependent

5 Conclusion

The implementation of canny edge detector using VLIB was discussed. Simulation and benchmarking was performed on the 64x+ simulator and the functionality was tested on DM6437, DM642 and DM6446.

6 References

1. *A Computational Approach to Edge Detection* by Canny, J., IEEE Trans. Pattern Analysis and Machine Intelligence, 8:679-714, 1986.
2. TMS320C64x+ DSP Image/Video Processing Library (v2.0.1)
3. *Vision Library (VLIB) Application Programming Interface Reference Guide* (SPRUG00). Describes the VLIB Application Programming Interface (v2.1)

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Audio	www.ti.com/audio	Communications and Telecom	www.ti.com/communications
Amplifiers	amplifier.ti.com	Computers and Peripherals	www.ti.com/computers
Data Converters	dataconverter.ti.com	Consumer Electronics	www.ti.com/consumer-apps
DLP® Products	www.dlp.com	Energy and Lighting	www.ti.com/energy
DSP	dsp.ti.com	Industrial	www.ti.com/industrial
Clocks and Timers	www.ti.com/clocks	Medical	www.ti.com/medical
Interface	interface.ti.com	Security	www.ti.com/security
Logic	logic.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Power Mgmt	power.ti.com	Transportation and Automotive	www.ti.com/automotive
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com	Wireless	www.ti.com/wireless-apps
RF/IF and ZigBee® Solutions	www.ti.com/lprf		
	TI E2E Community Home Page		e2e.ti.com

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265