

SIMD Image Processor

Eric Liskay

Andrew Northy

Neraj Kumar

Features

- **SIMD architecture**

- Large array of cell processors operating in parallel
- Each processor performs the same function on a kernel
 - Kernel size is a 3x3 array of pixels
- Number of cell processors equal to (Tile Size in kernels)²
 - For example, if the tile size is 10 kernels, each tile will be 30x30 pixels

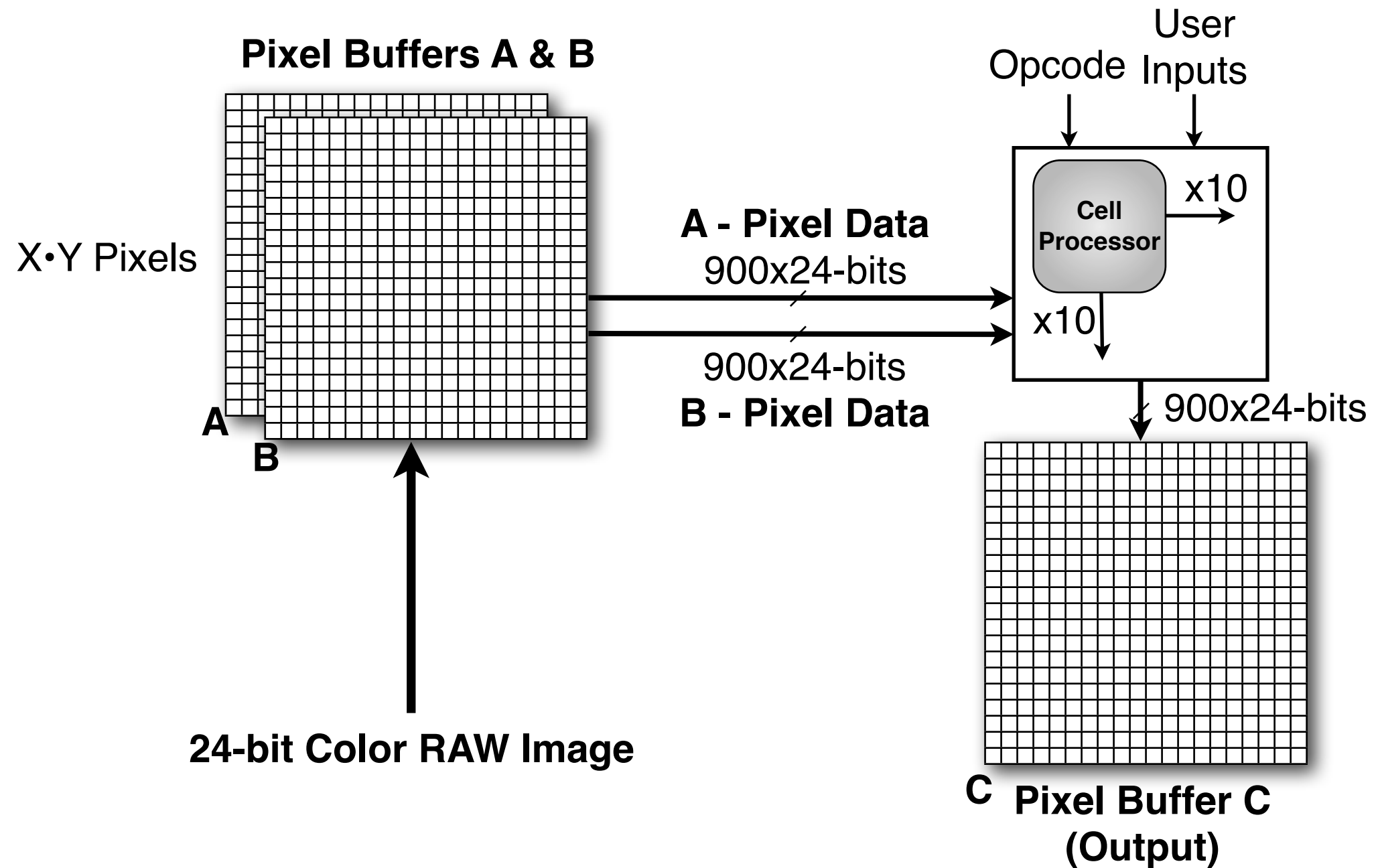
- **Convolution Mode**

- 3x3 pixel matrix read into the cell, only center pixel is written out
- Used for minimum, maximum, average, and edge detection operations

- **User Inputs**

- User can specify parameters for different functions
- User can specify a custom kernel for convolution

Top Level Diagram



Data Structures

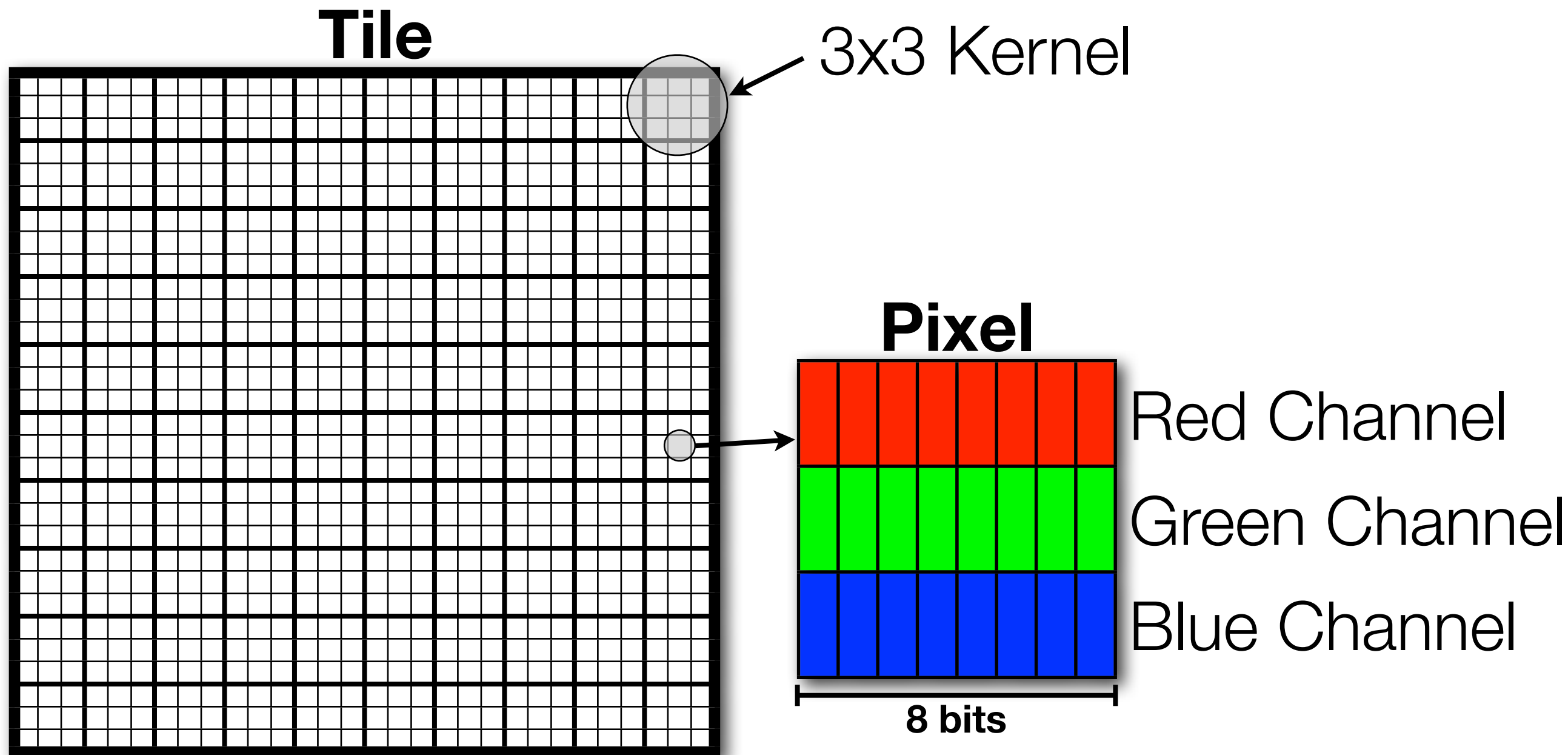


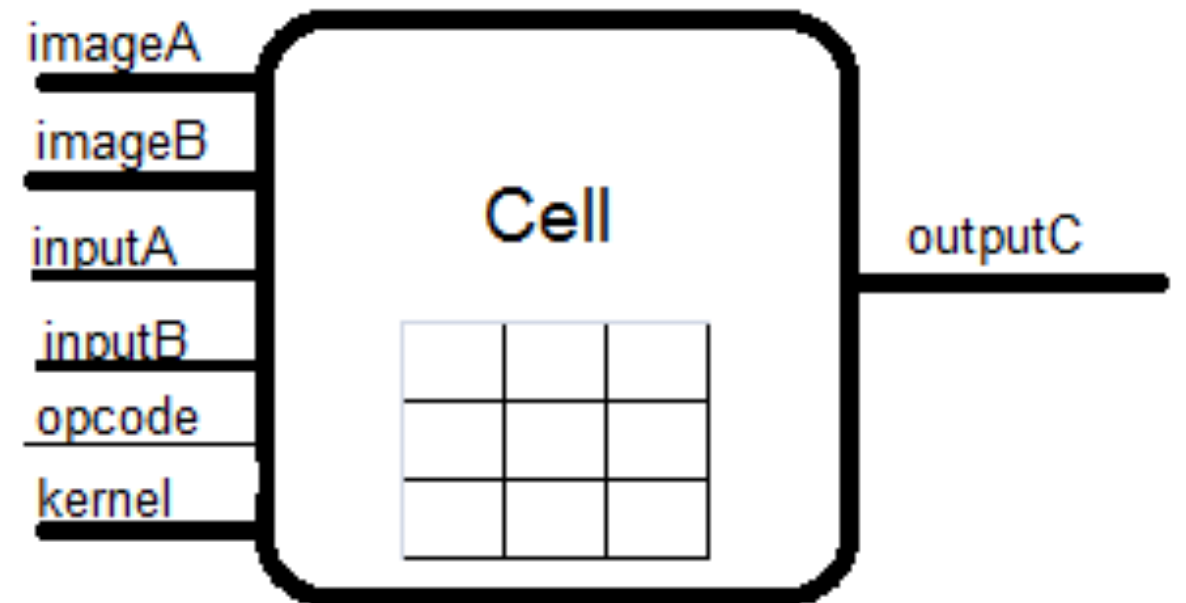
Image Processor Controller

- **4 Processes**

- Load images from files into pixel buffers
- Positive Edge of Clock: Send data to cell processors
- Negative Edge of Clock: Read image from cell processors into Pixel Buffer C
- Write Output to file when finished processing the image

Cell Processor

- Each Cell processor operates on 3x3 pixel array, each pixel has 3 color channels
 - Two image inputs
 - Two user inputs
 - Opcode input
 - Kernel input (user's filter)
 - One image output



Cell Functions

- **Arithmetic and logical operations**
 - Add, subtract, multiply divide
 - AND, OR, NOR, XOR
 - Operands may be two images, or one image and user input
 - Operation can be limited to one color channel or all channels

Cell Functions

- **Darken Highlights**

- Compare sum of color channels with threshold value
- Decrease value from all channels by set amount if sum is larger than threshold

- **Brighten Shadows**

- Compare sum of color channels with threshold value
- Increases value from all channels by set amount if sum is smaller than threshold

Cell Functions

- **Convolution**

- Sobel, Prewitt, and Robinson Edge Detection
- User specified Kernel
- 3x3 Minimum, Maximum, and Averaging Filters

Edge Detection Using the Sobel Operator

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * A \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * A$$

$$G = \sqrt{G_x^2 + G_y^2}$$

- The Sobel operator uses two 3x3 kernels which are convolved with a 3x3 array of pixels from the image
- A is the 3x3 array of pixels from the image
- G_x and G_y are the derivatives of the convolutions
- $*$ is the convolution operator
- G is the gradient magnitude

The Opcodes

Regular Processing Mode
(9 pixels in, 9 pixels out)

Mnemonic	Opcode	Description
add	0x00	add pixel channel values in array A to array B
addi	0x01	add userInputA to values in all channels from array A
addir	0x02	add userInputA to value in red channel from array A
addig	0x03	add userInputA to value in green channel from array A
addib	0x04	add userInputA to value in blue channel from array A
sub	0x05	subtract pixel channel values in array B from array A
subi	0x06	subtract userInputA from values in all channels from array A
subir	0x07	subtract userInputA from value in red channel from array A
subig	0x08	subtract userInputA from value in green channel from array A
subib	0x09	subtract userInputA from value in blue channel from array A
mult	0x0A	multiply pixel channel values in array A by array B
mult2	0x0B	multiply pixel channel values in array A by 2
multi	0x0C	multiply pixel channel values in array A by shifting left by userInputA
div2	0x0D	divide pixel channel values in array A by 2
divi	0x0E	divide pixel channel values in array A by shifting right by userInputA
inv	0x0F	Invert pixel channel values in array A
and	0x10	AND pixel channel values in Array A with pixels in array B
or	0x11	OR pixel channel values in Array A with pixels in array B
nor	0x12	NOR pixel channel values in Array A with pixels in array B
xor	0x13	XOR pixel channel values in Array A with pixels in array B
drkn	0x14	Darken Highlights: If sum of pixel channel values in array A are above userInputA, subtract userInputB
brtn	0x15	Brighten Shadows: If sum of pixel channel values in array A are below userInputA, add userInputB
grayr	0x16	Grayscale based on Red Channel
grayg	0x17	Grayscale based on Green Channel
grayb	0x18	Grayscale based on Blue Channel
pixflip	0x19	Pixels mirrored over middle pixel

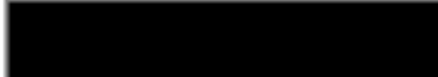















The Opcodes

Convolution Mode
(9 pixels in, 1 pixels out)

Mnemonic	Opcode	Description
sobel	0x1A	Sobel
sobel2	0x1B	Sobel Method 2
prewitt	0x1C	Prewitt
robn	0x1D	Robinson North
robnw	0x1E	Robinson Northwest
robne	0x1F	Robinson Northeast
robe	0x20	Robinson East
robw	0x21	Robinson West
robs	0x22	Robinson South
robse	0x23	Robinson Southeast
robsw	0x24	Robinson Southwest
avggry	0x25	3x3 pixel average in grayscale
avgclr	0x26	3x3 pixel average in color
ukernel	0x27	User programmable filter
min	0x28	3x3 pixel minimum
max	0x29	3x3 pixel maximum

Color Codes

Basic colors:

Color	HTML/CSS Name	Hex Code #RRGGBB	Decimal Code (R,G,B)
	Black	#000000	(0,0,0)
	White	#FFFFFF	(255,255,255)
	Red	#FF0000	(255,0,0)
	Lime	#00FF00	(0,255,0)
	Blue	#0000FF	(0,0,255)
	Yellow	#FFFF00	(255,255,0)
	Cyan / Aqua	#00FFFF	(0,255,255)
	Magenta / Fuchsia	#FF00FF	(255,0,255)
	Silver	#C0C0C0	(192,192,192)
	Gray	#808080	(128,128,128)
	Maroon	#800000	(128,0,0)
	Olive	#808000	(128,128,0)
	Green	#008000	(0,128,0)
	Purple	#800080	(128,0,128)
	Teal	#008080	(0,128,128)
	Navy	#000080	(0,0,128)

Regular Processing Mode

Original



Processed



Red Image

```
outputC(x,y)(0) <= B"11111111";  
outputC(x,y)(1) <= inputA(x,y)(1);  
outputC(x,y)(2) <= inputA(x,y)(2);
```


Original



Processed



Red Image

```
outputC(x,y)(0) <= inputA(x,y)(0);  
outputC(x,y)(1) <= B"11111111";  
outputC(x,y)(2) <= inputA(x,y)(2);
```


Original



Processed



Red Image

```
outputC(x,y)(0) <= inputA(x,y)(0);  
outputC(x,y)(1) <= inputA(x,y)(1);  
outputC(x,y)(2) <= B"11111111";
```


Original



Processed

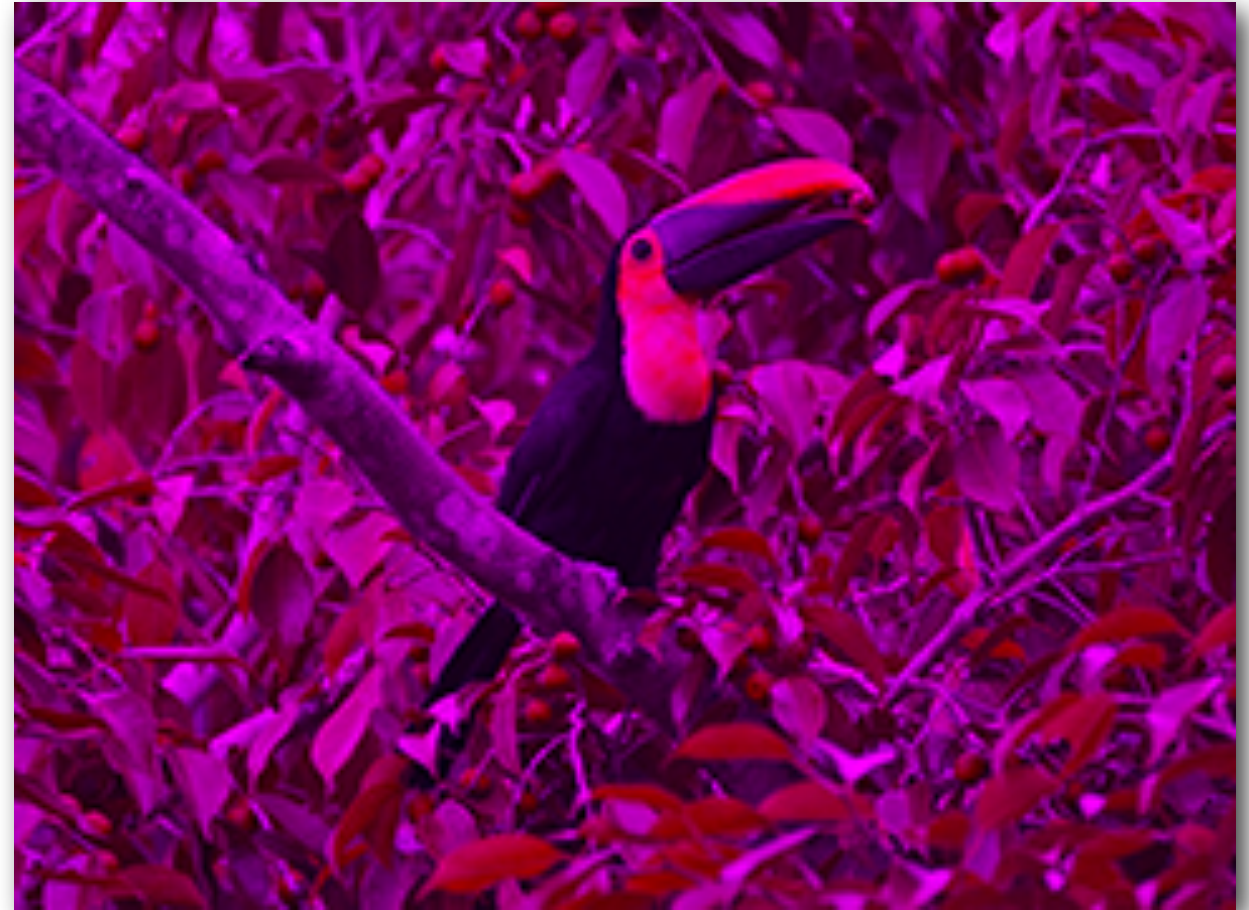


Red = 0

Original



Processed



Green = 0

Original



Processed



Blue = 0

Original



Processed



Add: Image A + Image B



Original



Processed



Add Immediate: Image A + User Input A

Original



Processed



**Add Immediate Red: Image A(red channel) +
User Input A**

Original



Processed



Add Immediate Green: Image A(green channel) + User Input A

Original



Processed



**Add Immediate Blue: Image A(blue channel)
+ User Input A**

Original



Processed



Subtract: Image A - Image B

Original



Processed



Subtract Immediate: Image A - User Input A

Original



Processed



Subtract Immediate Red: Image A(red channel) - User Input A

Original



Processed



Subtract Immediate Green: Image A(green channel) - User Input A

Original



Processed



Subtract Immediate Blue: Image A(blue channel) - User Input A



Original

Processed



Multiply: Image A * Image B

Original



Processed

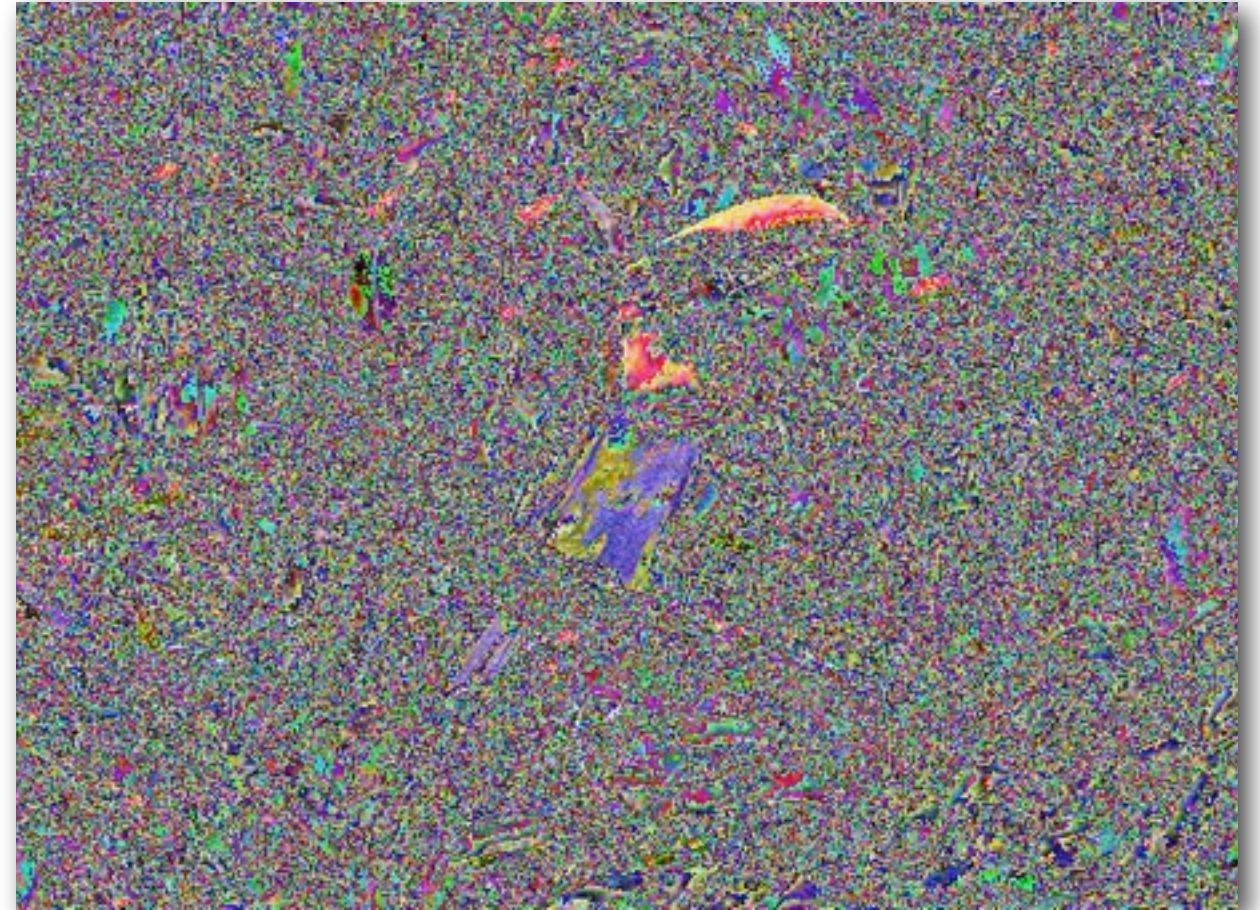


Multiply by 2: Image A * 2

Original



Processed



Multiply Immediate: Image A * User Input A

Original



Processed

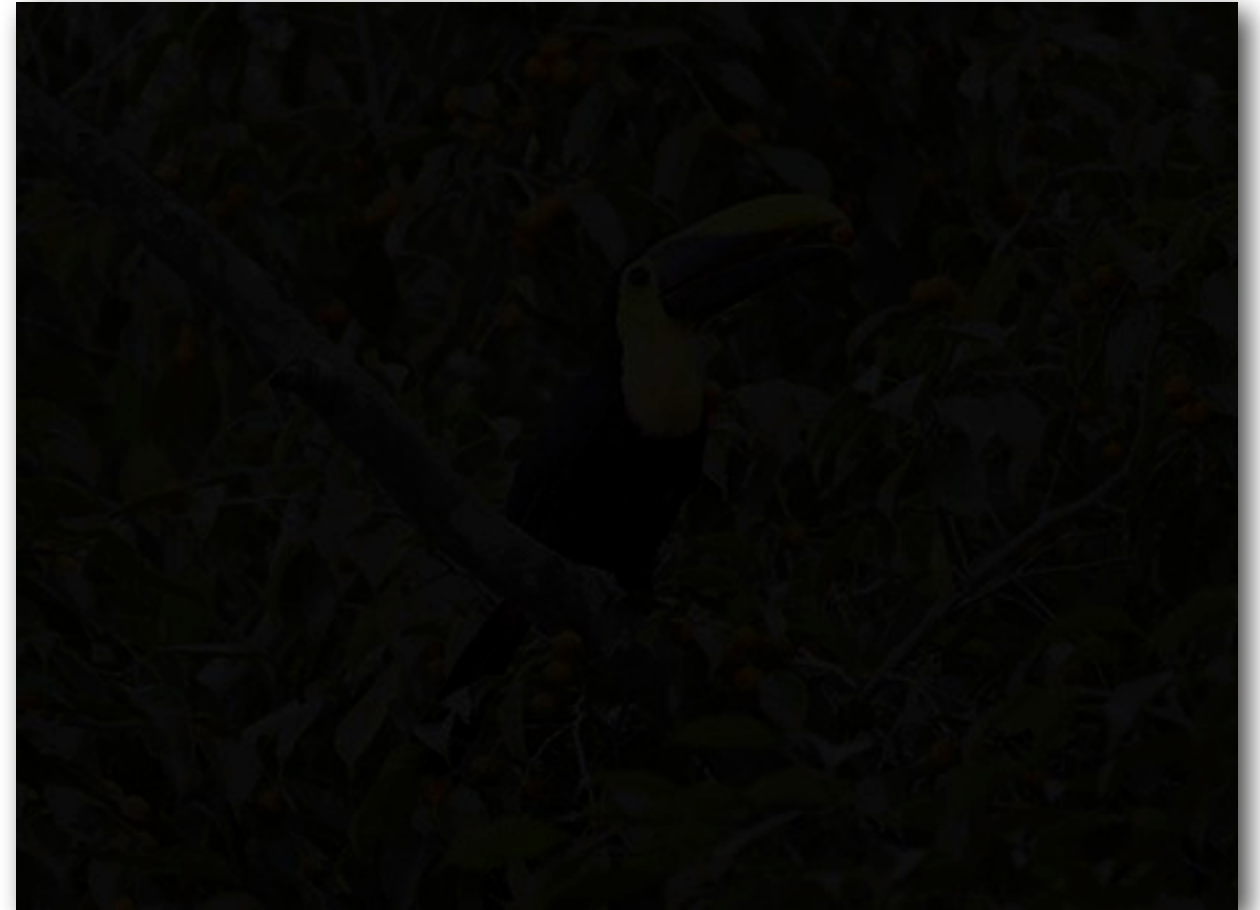


Divide by 2: Image A / 2

Original



Processed



Divide Immediate: Image A >> User Input A

Original



Processed



Invert: Image A XOR "11111111"

Original



Processed



AND: Image A AND Image B

Original



Processed



OR: Image A OR Image B

Original



Processed



NOR: Image A NOR Image B

Original



Processed



XOR: Image A XOR Image B

Original



Processed



Darken Highlights: Decreases brightness by (User Input B) if average value of the pixels is above User Input A

Original



Processed



Brighten Shadows: Increases brightness by (User Input B) if average value of the pixels is below User Input A

Original



Processed



Convert to Grayscale using Red Channel

Original



Processed



Convert to Grayscale using Green Channel

Original



Processed

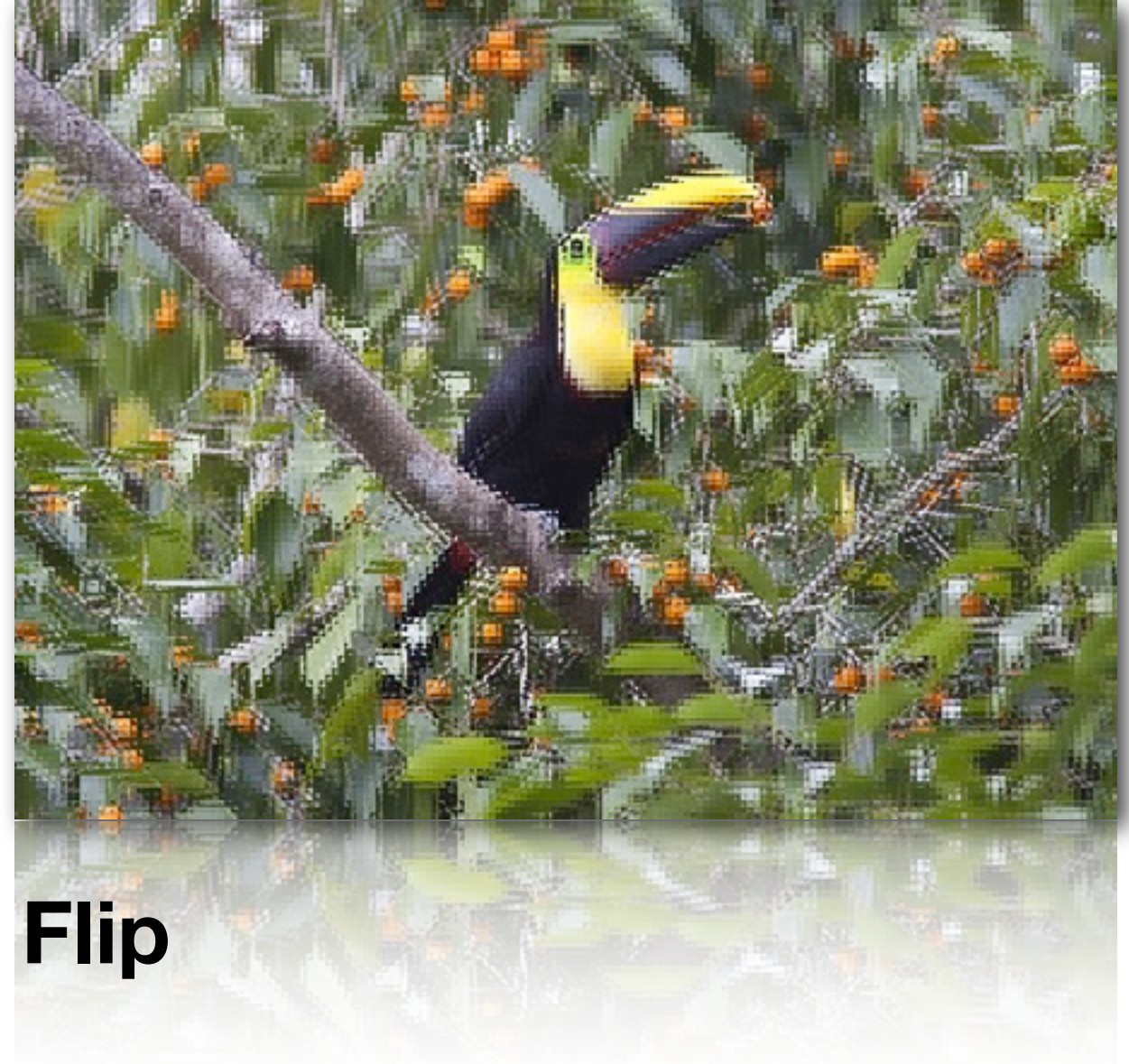


Convert to Grayscale using Blue Channel

Original

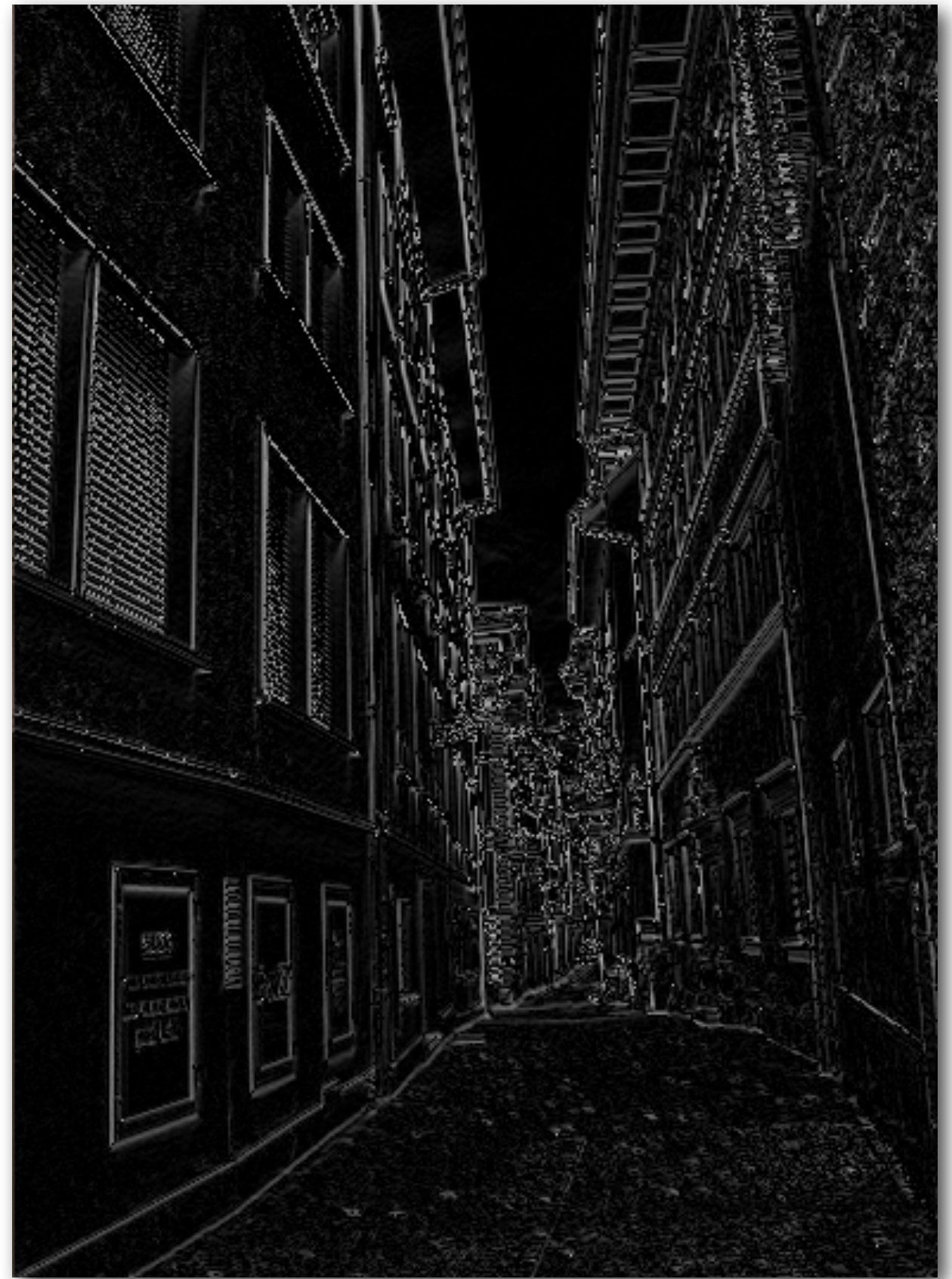


Processed



Pixel Flip

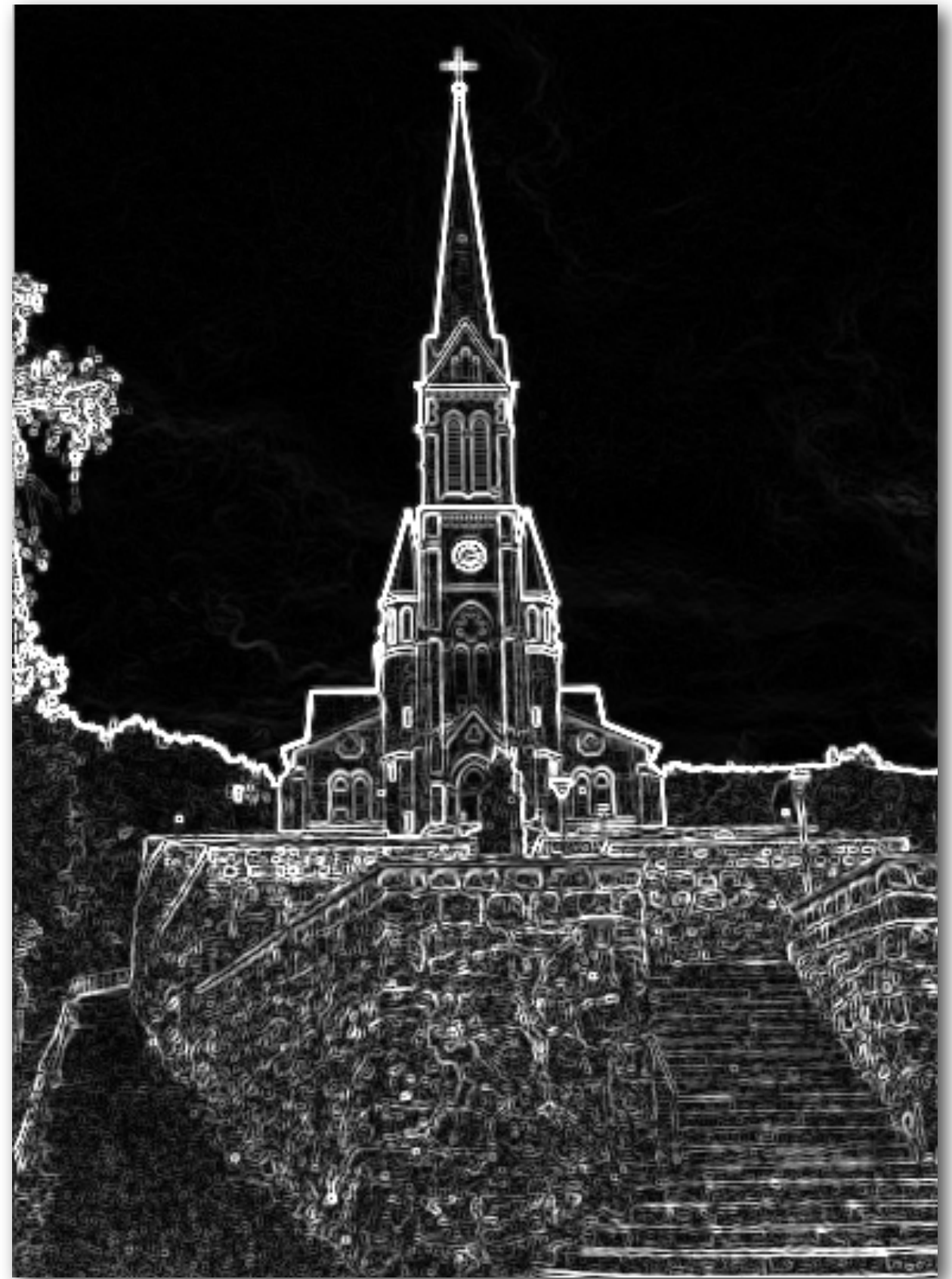
Convolution Mode



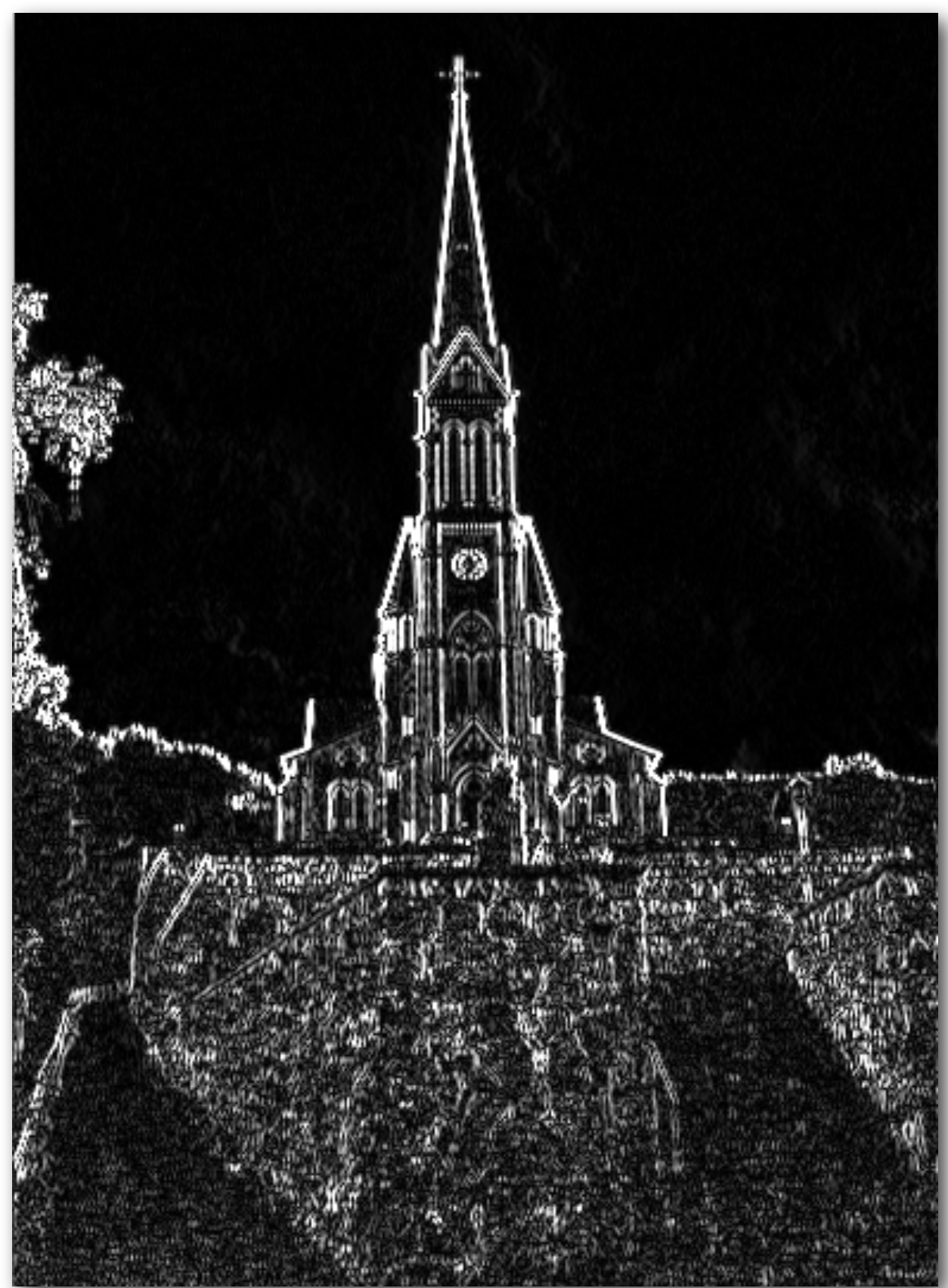
Sobel



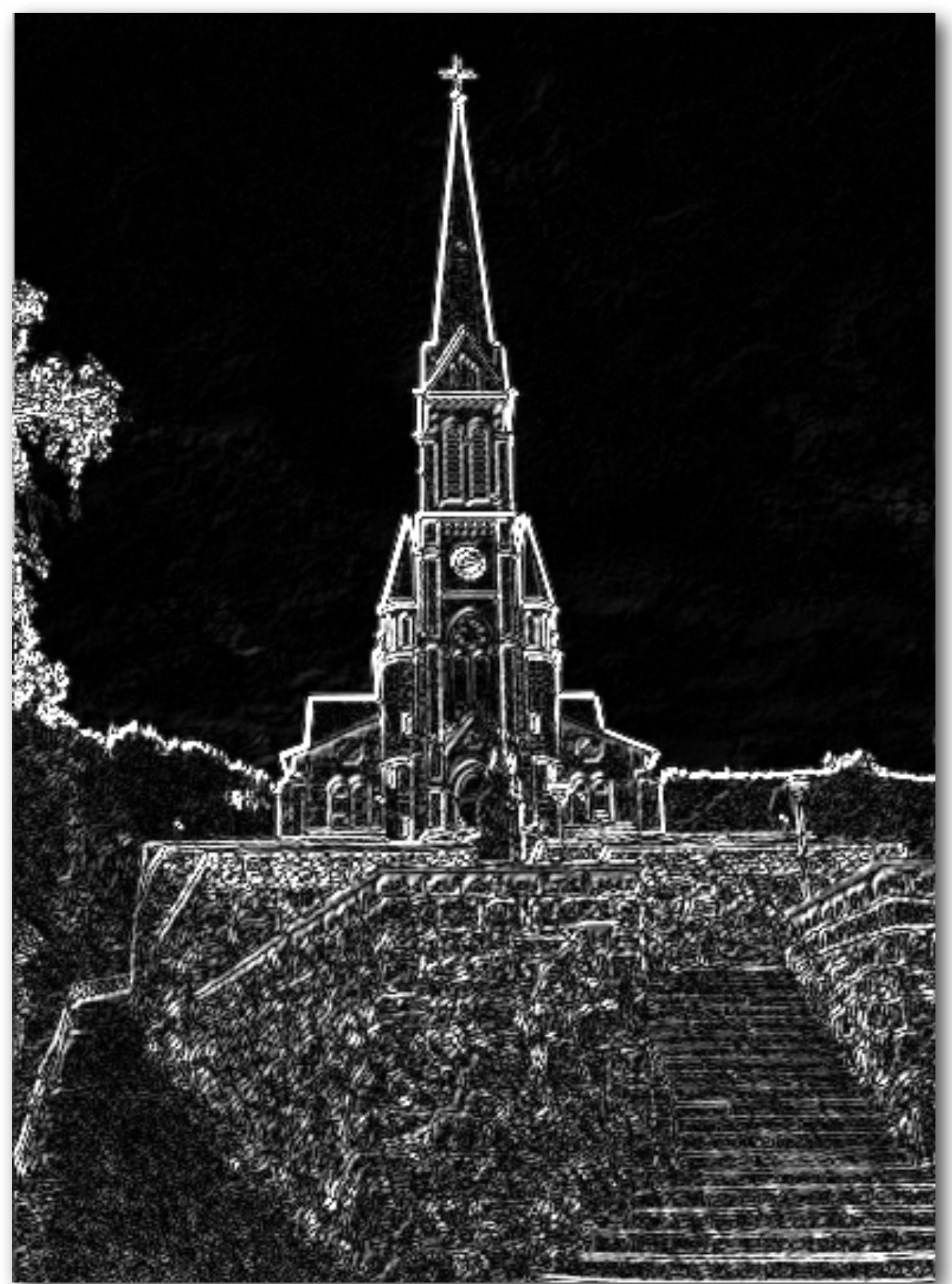
Sobel Method 2



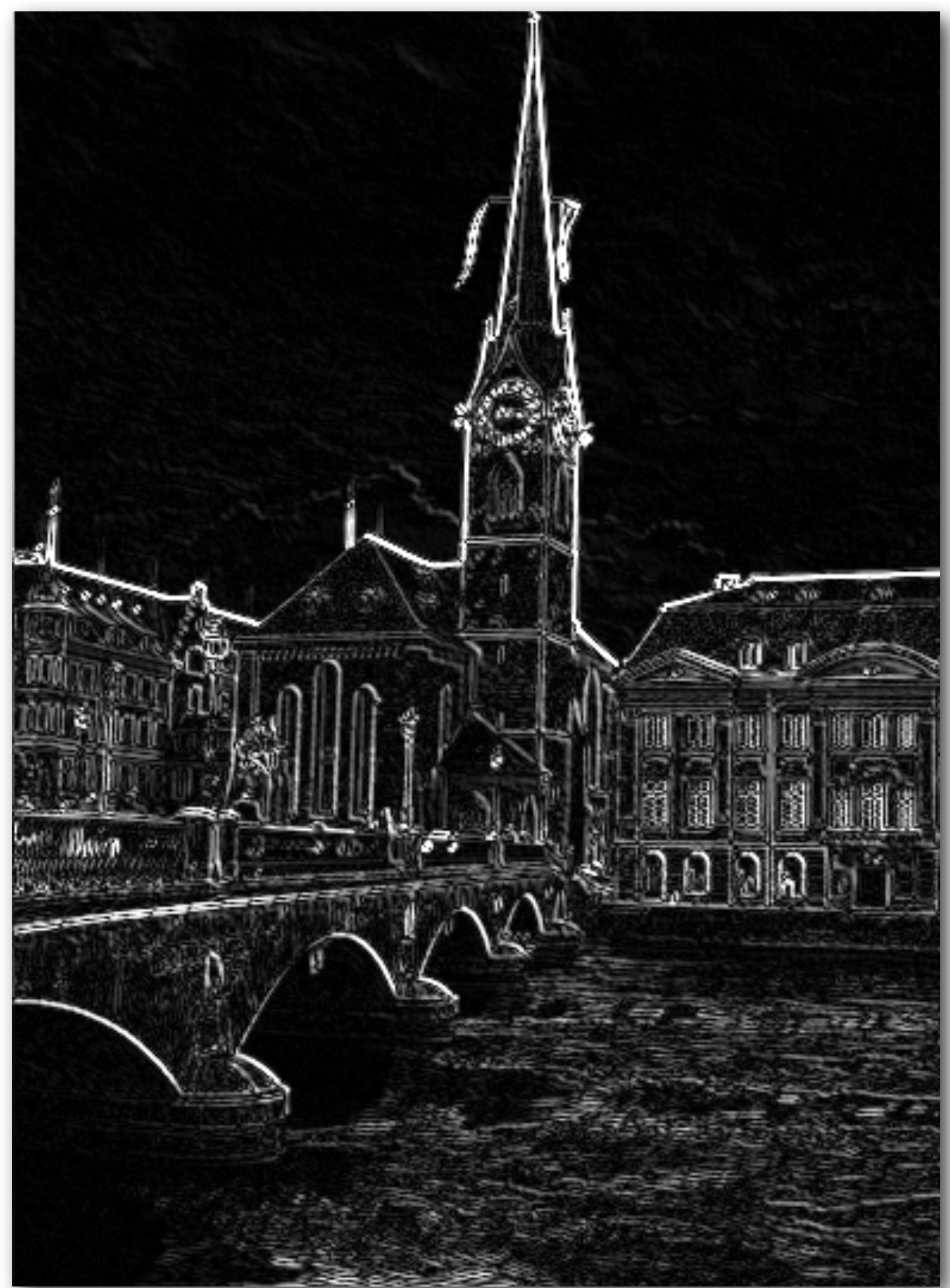
Prewitt



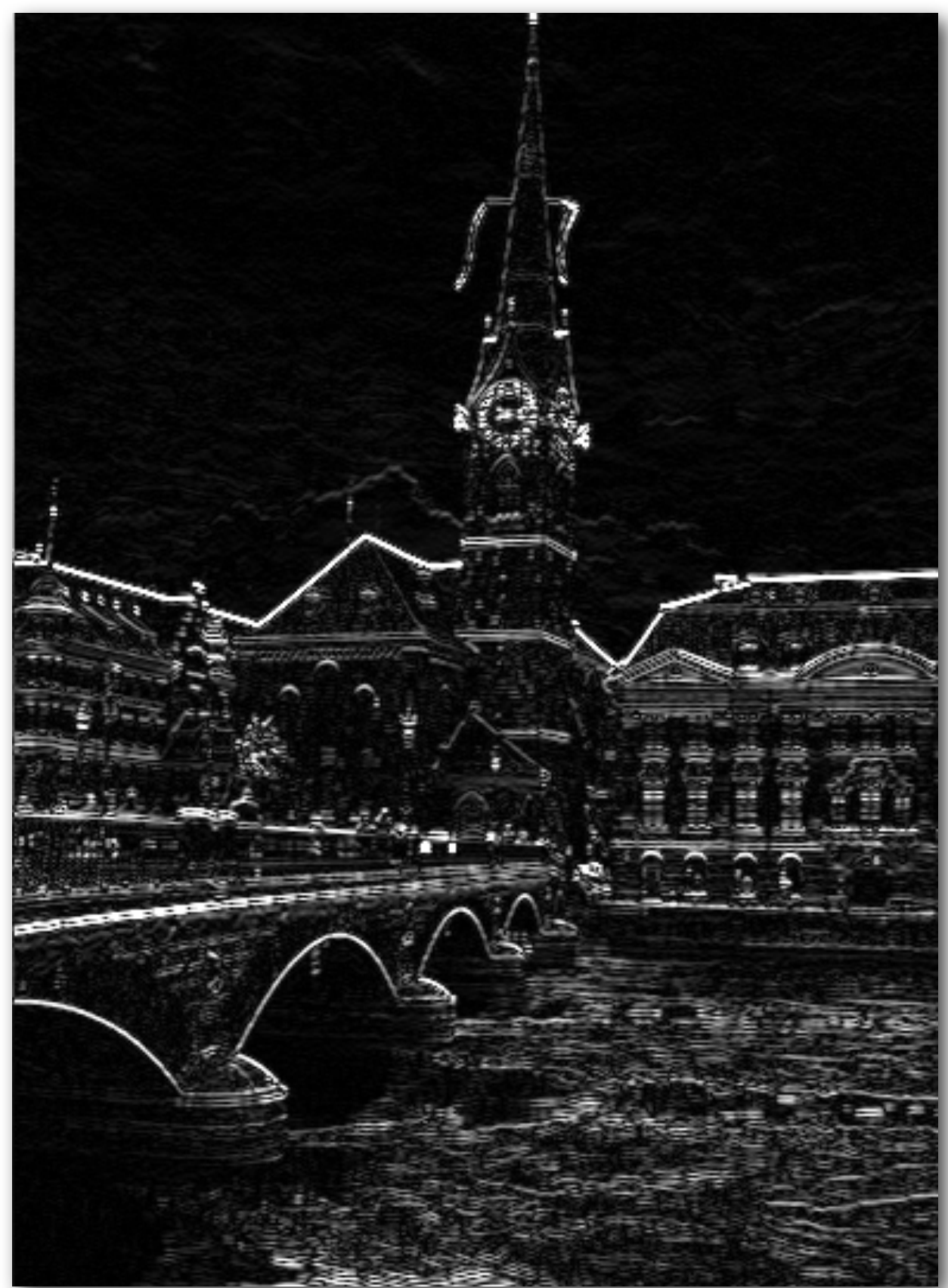
Robinson North



Robinson Northwest



Robinson Northeast



Robinson East



Robinson West



Robinson South



Robinson Southeast



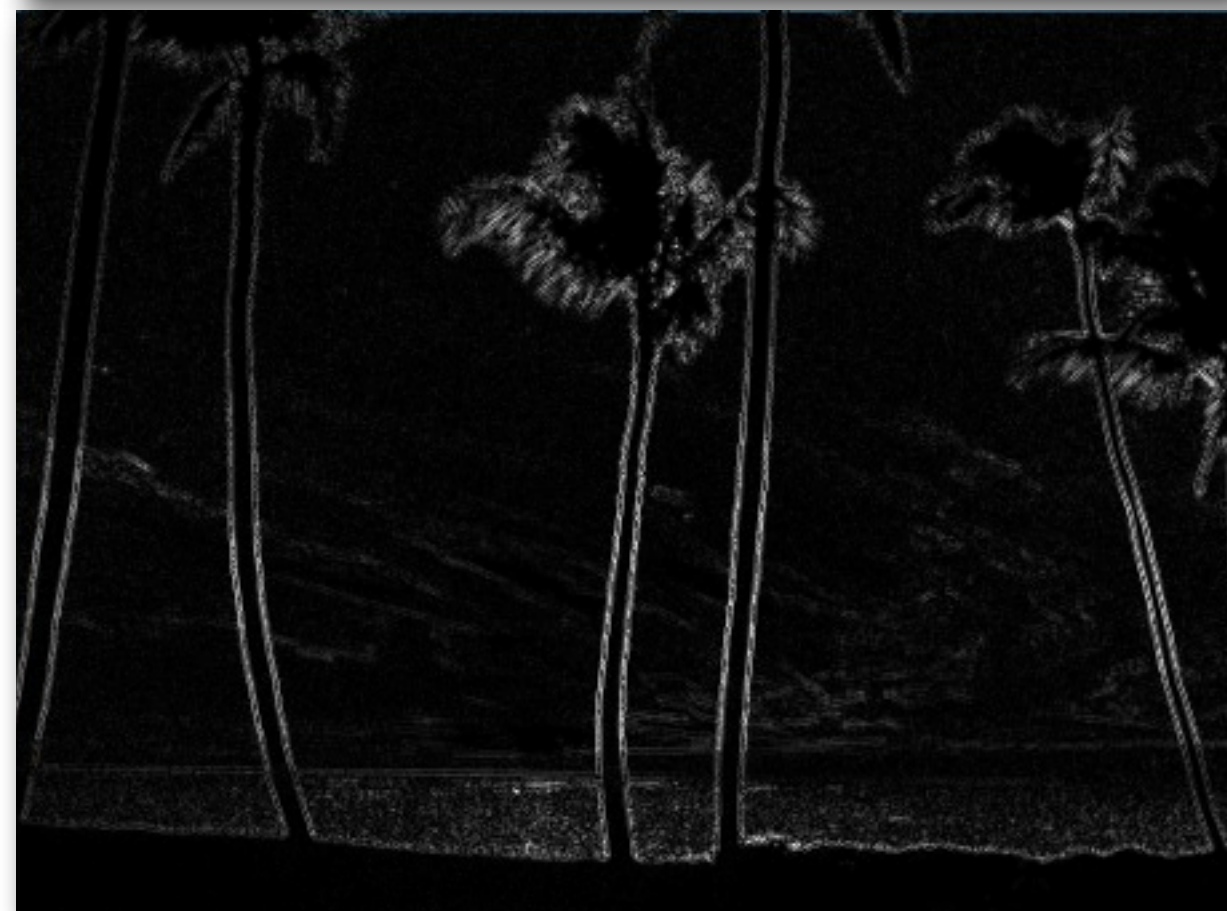
Robinson Southwest



3x3 Pixel Average Grayscale



3x3 Pixel Average Color



User Programmable Filter

Original



Processed



3x3 Pixel Minimum

Original



Processed



3x3 Pixel Maximum

Future Work

- Make Image Processor synthesizable
 - Move loading files into testbench(HVL) and separate from image processor and Cells(HDL)
 - Veloce?
- Enable the processing of larger images by loading and writing only part of the image at a time
- Other convolution ideas(e.g. Noise Reduction)
- Parameterized Kernel Size to enable larger kernels
- Gradient Threshold on “Darken Highlights” and “Brighten Shadows” for a less abrupt transition
- Multiple Operations done on image before writing it

Original



Processed



Horizontal Flip

Original



Processed



Vertical Flip

Original



Processed



Rotate 180° Clockwise

References

- **All Photographs © Eric Liskay**
- **Lecture 10. Edge detection. Sobel and similar filters. Convolution**
 - http://web.cecs.pdx.edu/~mperkows/CLASS_VHDL_99/2012/2012-lecture010_edge-detection-SobelG.ppt
- **Wikipedia - Sobel Operator**
 - http://en.wikipedia.org/wiki/Sobel_operator