**Morphological Laplacian Operator Edge Detection Filter using NVIDIA’s CUDA Parallel Processing (SDK version 8)**

—Ian Calegory, 12/20/2016

Running the LaplacianOperator.exe program in the Visual Studio 2015 solution LaplacianOperator will bring up a console like the following. Note that the default size of the kernel printf buffer is included in the device statistics—I upped that by a factor of 50 to allow printf statements in my CUDA kernels to output properly (and then reset it to default after I finished debugging).

Use the “-“ (or “\_”) and “=” (or “+”) keys to select among the following structuring elements (visual representations of these can be found in the project file laplacianFilter\_kernel.cu):  
 disk3x3,

disk5x5,

disk7x7,

square3x3,

square5x5,

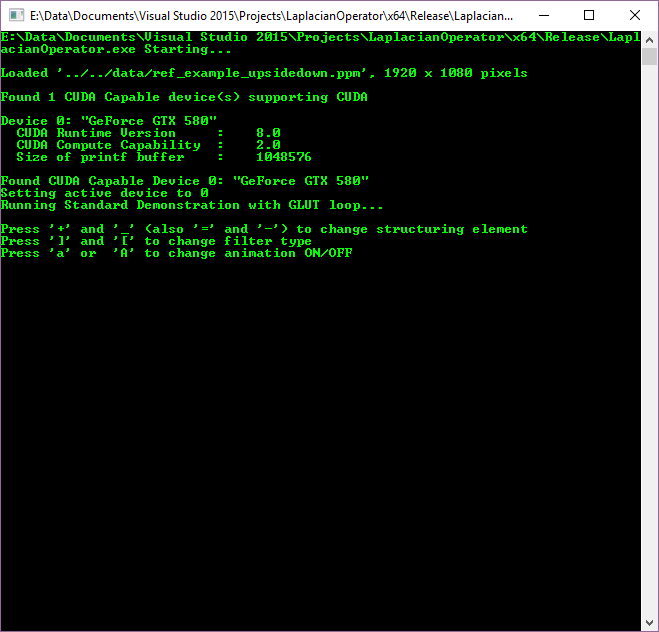
square7x7,

ring3x3,

ring5x5,

ring7x7

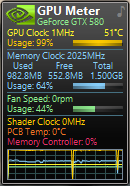
And, use the keys “[“ and “]” to cycle among the implemented filters, which are (of my creation):  
 AlmostAReference, AlmostFlattened, AntiAliasingSmoothFuzz, FuzzInWideOutline, GhostEdges, InvisoWithWideOutlines, MosaicInGray, PsychedelicLines, PsychedelicMellowed, ReliefInGray

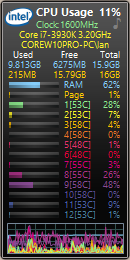


I was amazed at how much faster the 64-bit Release version is compared to the Debug version! See the following, which shows the “Mosaic In Gray” filter using the 5x5 ring structuring element being processed at 362.6 FPS! For comparison, the Debug version is only around 23.4 FPS.



It’s a neat experience filling the CUDA cores to capacity and having my liquid cooled system’s fans rev up to top RPM’s based on for a change the GPU temperature, as opposed to the CPU temperature (note how the GPU meter shows the GPU maxed out, while the CPU is basically idling by):









From laplacianFilter.cpp:

The image loaded for processing is the file ref\_example\_upsidedown.ppm

in the LaplacianOperator/data folder, so replace the image with another

one of the same name to load a custom image. The image should be

a .ppm file, 1920x1080.

Here are the comments for the main processing of the filters, in the file  
laplacianFilter\_kernel.cu file in the function:

\_\_global\_\_ void

d\_laplacianFilter\_rgba(unsigned char \*id, unsigned char \*od, int w, int h, FilterTypeEnum filter, int\* d\_structuringElement, unsigned int n)

/\*

Perform 2D morphological Laplacian operator (approximately? along with a number

of variations) on image using CUDA

This works by calculating the dilation and erosion of the image using the structuring

element centered on the current pixel being processed. It's passed in as the array

d\_structuringElement, which is a 2d array flattened into a 1d array for passing into

CUDA with device cudaMemcpyHostToDevice calls. Dilation is computed by finding the

maximum r, g, and b values for the pixels around the current pixel determined by the

mask of the structuring element. (If the and of the masking structuring element pixel

and the source image pixel in the corresponding position with the mask overlaid onto

the source image is 1, include that pixel in the source of pixels for choosing maximum

values.)

Erosion is computed similarly, though replacing the source pixel with the components

having the minimum instead of maximum values.

Dilation results in what's called an internal gradient, while erosion results in an

external gradient. For further reference on computing the internal and external

gradients, see for example http://www.inf.u-szeged.hu/ssip/1996/morpho/morphology.html

The grayscale filter uses the luminosity algorithm for converting to grayscale:

0.21 R + 0.72 G + 0.07 B

--Ian Calegory, 12/20/2016

// Comment from original box filter left here for reference--so as a reminder to check

// for coalescence

Note that the x (row) pass suffers from uncoalesced global memory reads,

since each thread is reading from a different row. For this reason it is

better to use texture lookups for the x pass.

The y (column) pass is perfectly coalesced.

Parameters:

id - pointer to input image in device memory (not used here--texture is used instead)

od - pointer to destination image in device memory

w - image width

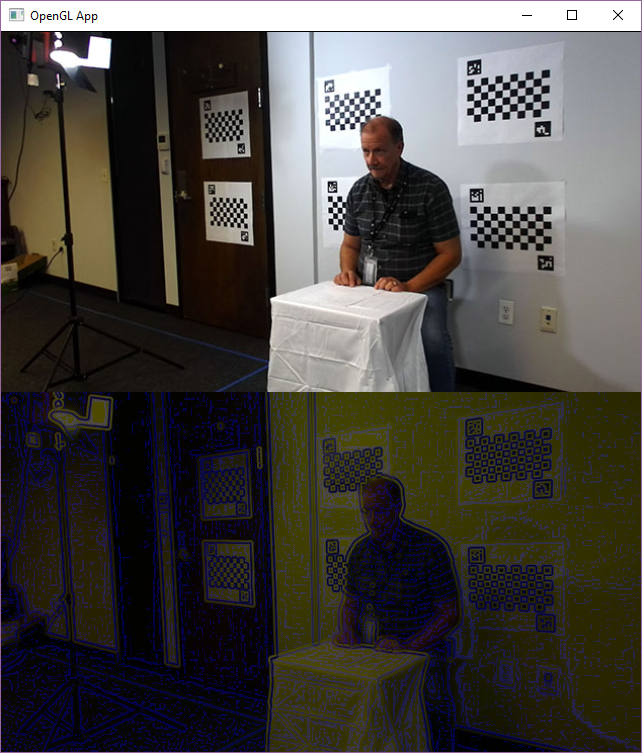
h - image height

d\_structuringElement - element 0 of the structuring element array

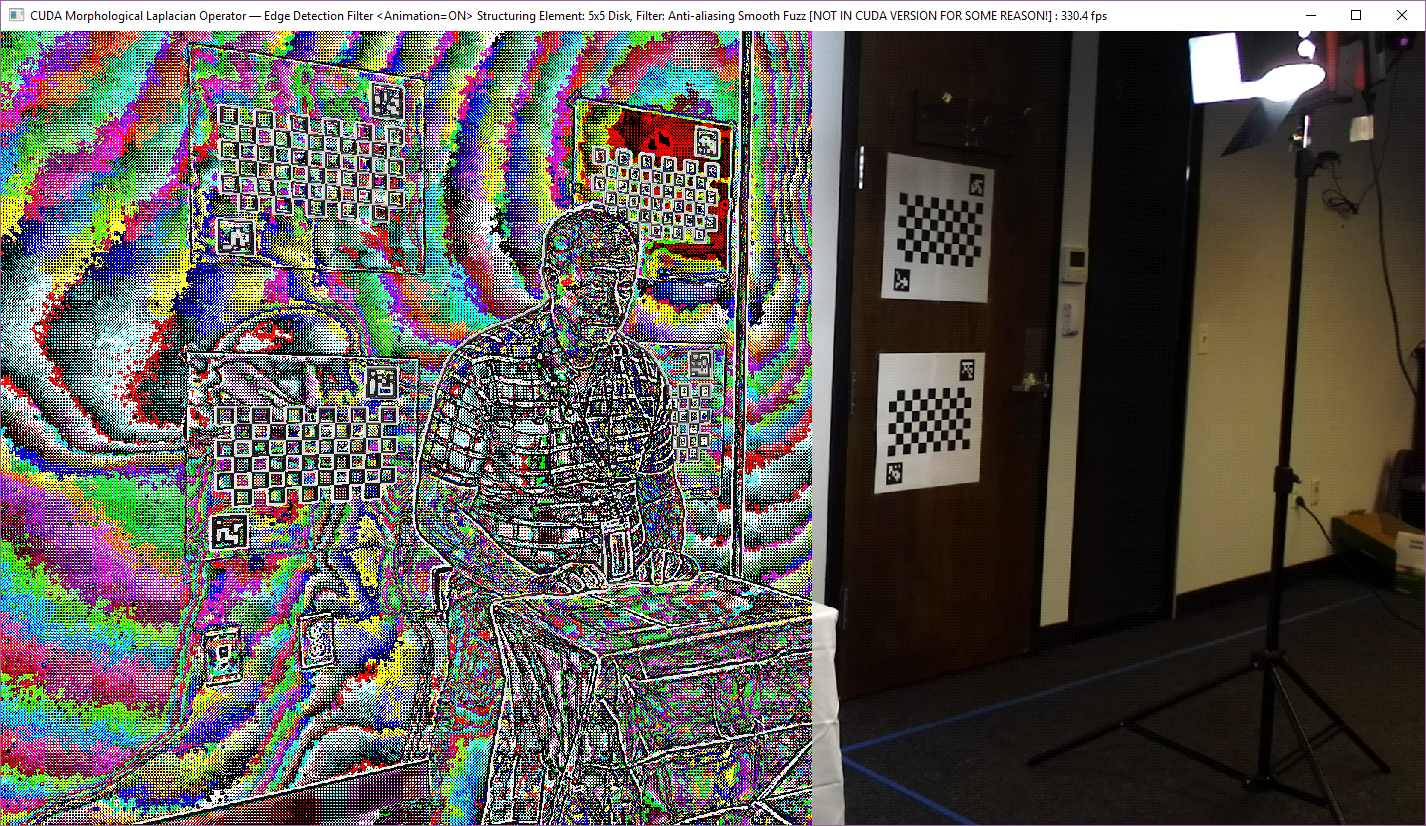
n - structuring element is nxn matrix

\*/

See the Visual Studio 2015 solution OpenGLDisplayImage for a CPU version of these filters. Interestingly though a couple of the filters look remarkably different in the CPU version compared to the CUDA version. I suspect it may have something to do with the differences caused by the CPU version using an OpenGL image buffer, while the CUDA version is using an OpenGL texture, instead. Here’s the image produced by the AntiAliasingSmoothFuzz filter in the CPU version (the original image appears atop the filtered image):

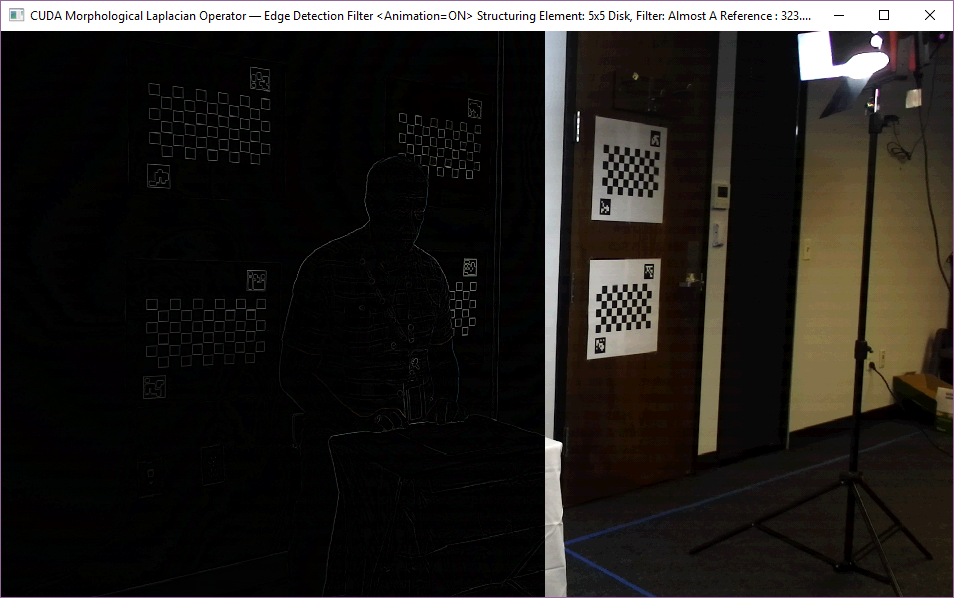


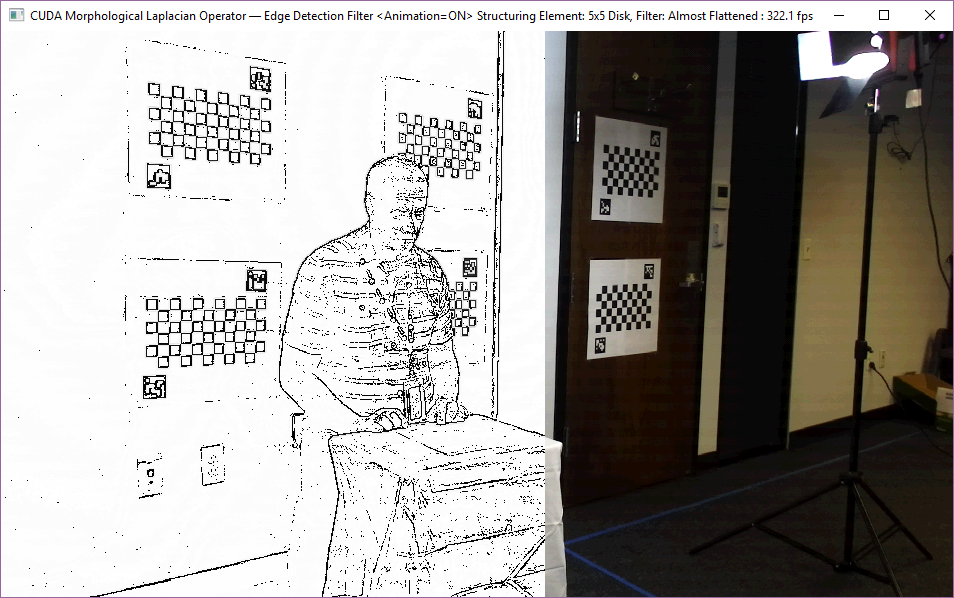
Now, check out the image produced by the CUDA version (wow!):

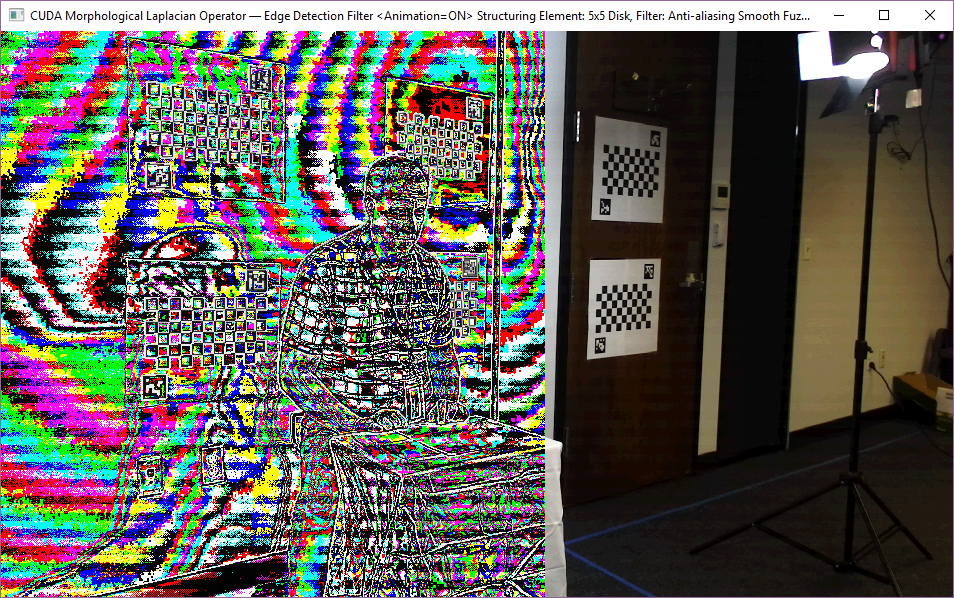


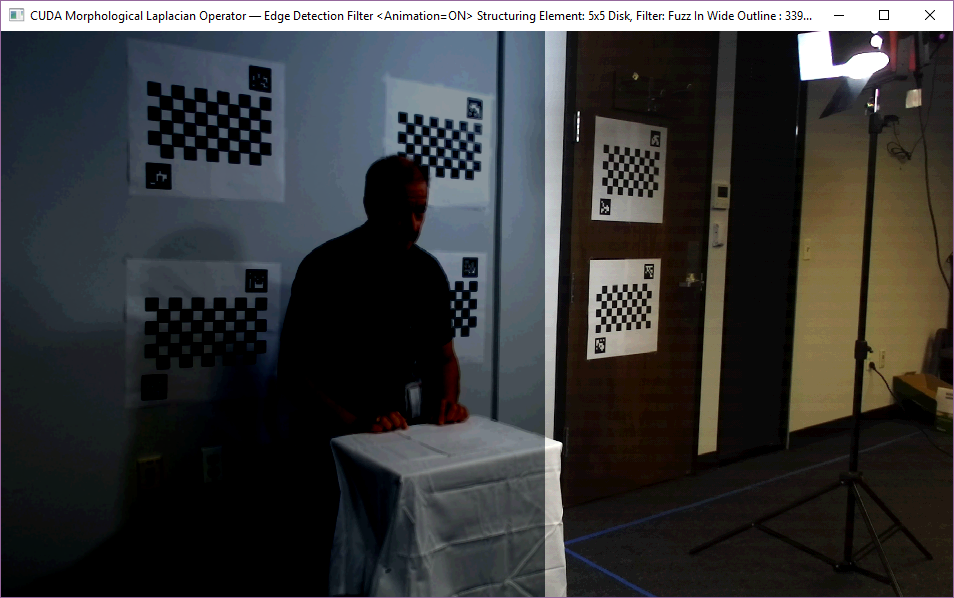
It should be noted that the effect of the filter in the CPU version was actually a neat side effect of an error in the computation of the red component, caused by an extra set of parentheses which changed the order of operations. Even changing the CUDA version to (supposedly) compute it the same as the CPU version still is vastly different than the first image, above.

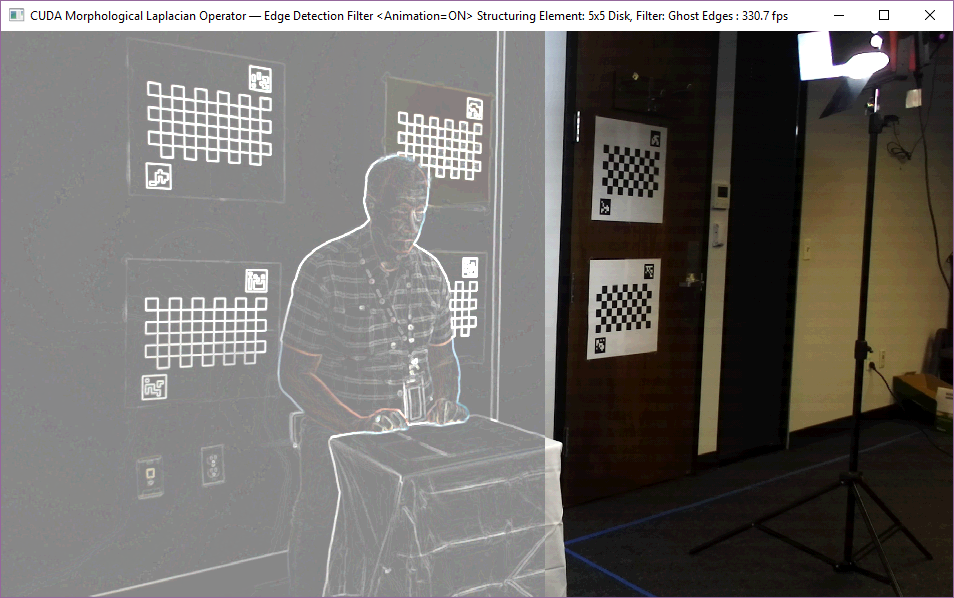
And, it should be noted that how I discovered the neat variations of these filters was by trying a *large* number of different formulas, in early versions of my code when I was trying to get the bits to compute and computations to align properly. Once I got the pixel and transfer processes operating correctly, I discovered that most of the different formulas I had constructed actually produced some pretty cool results! Here are screen shots, taken from the CUDA version. Interestingly, some of the filters (notably, here the ones with wild colors) look a lot different in the CPU version.

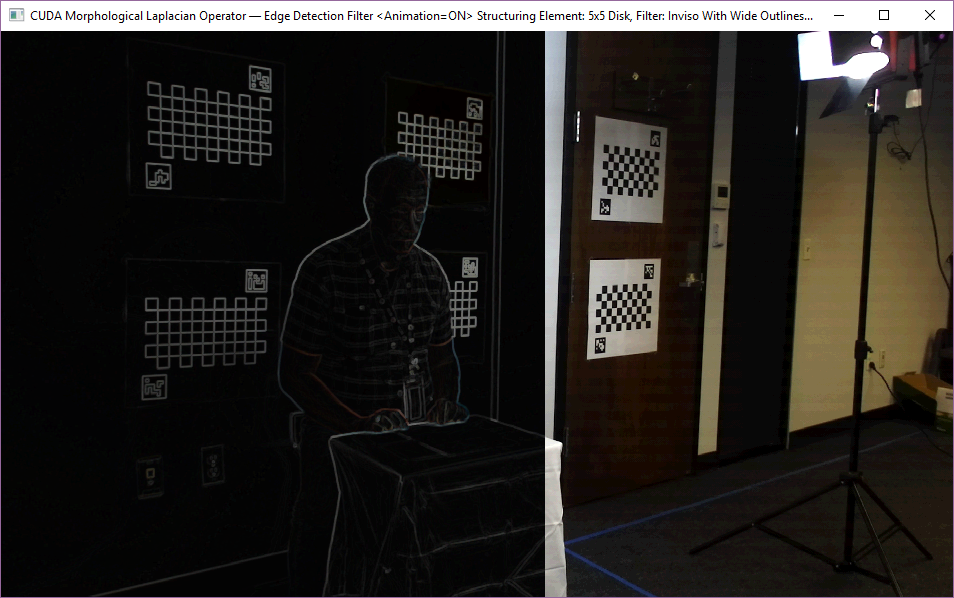


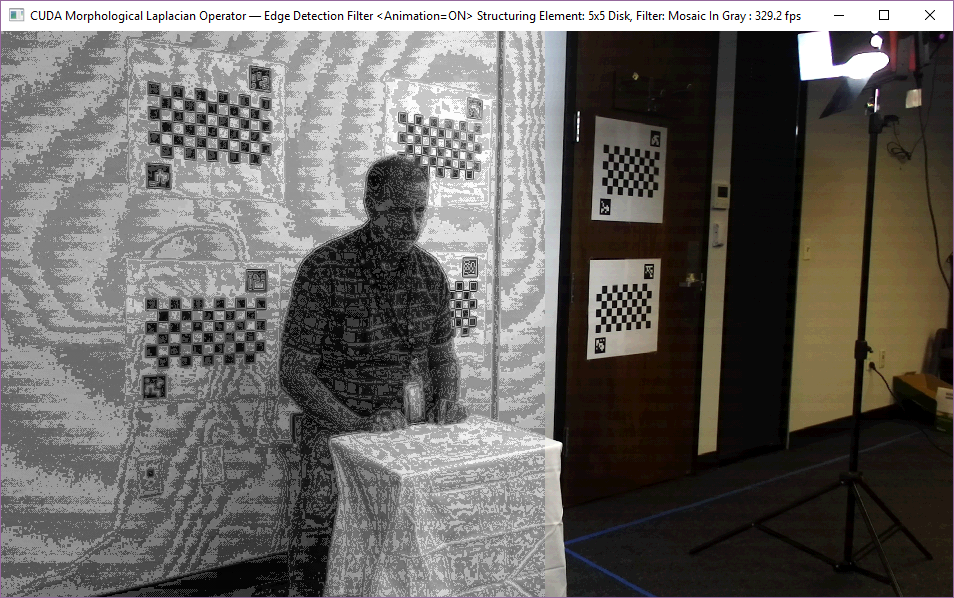


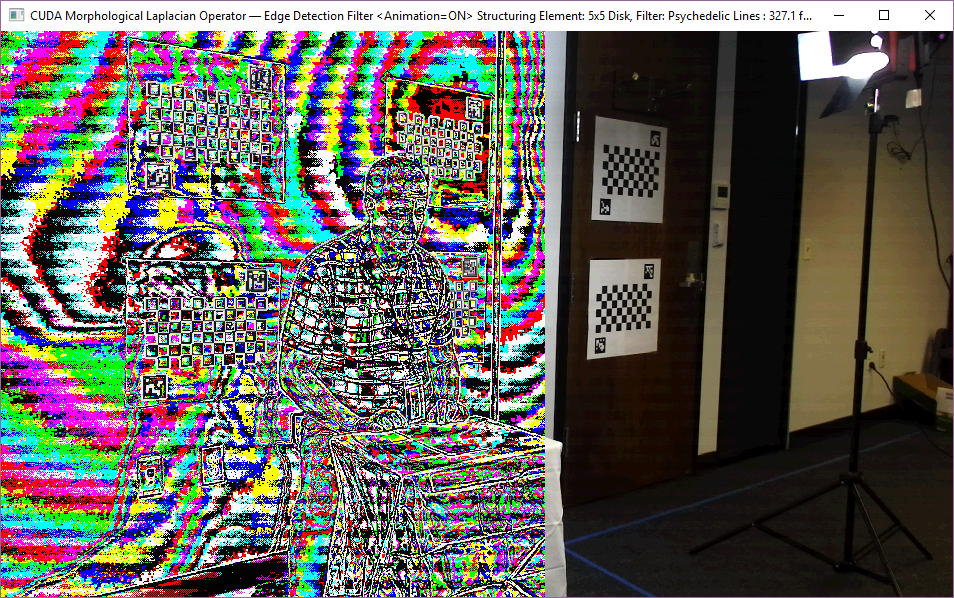


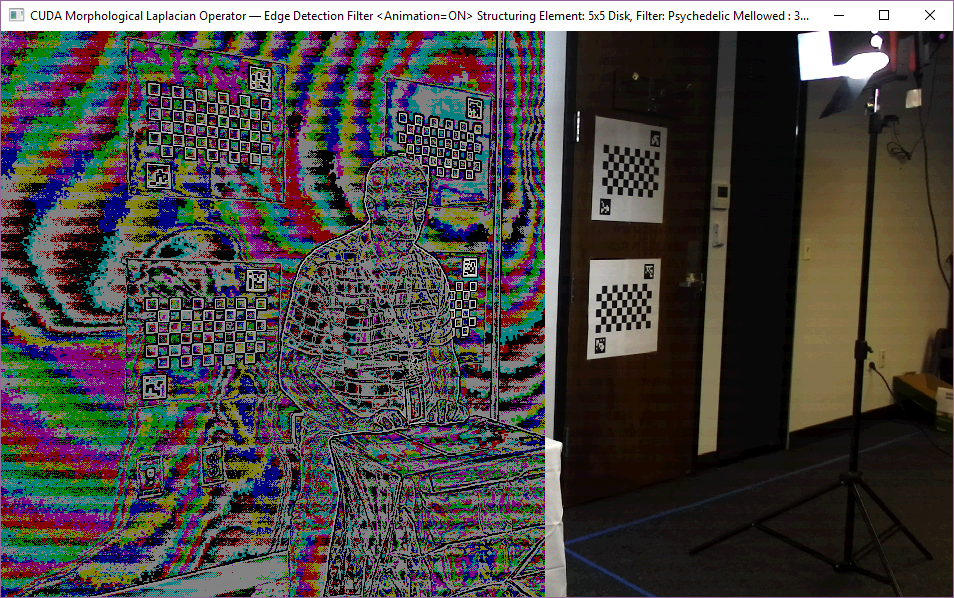


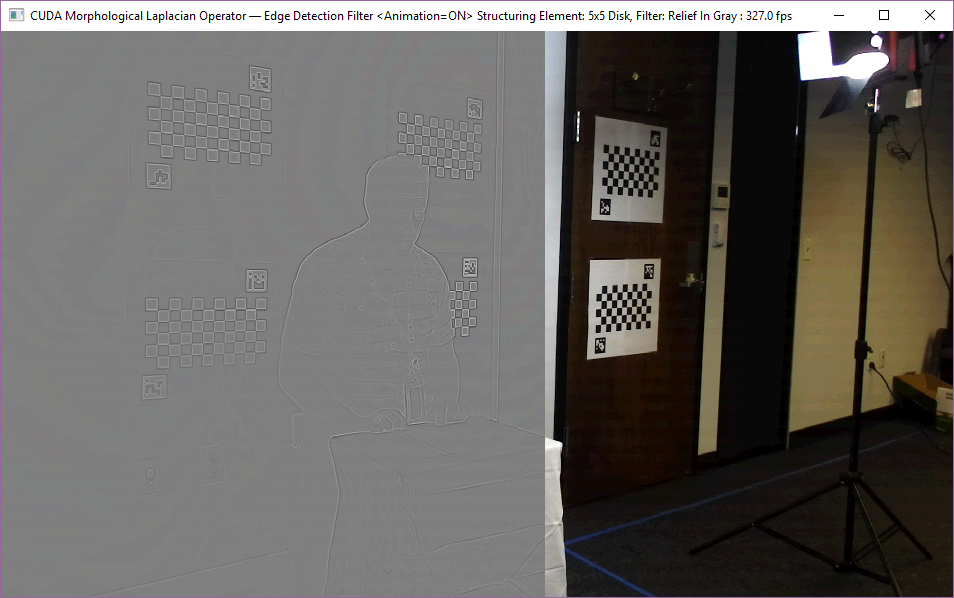














Ian Calegory

Software Engineer

[ianc102002@yahoo.com](mailto:ianc102002@yahoo.com)

[http://www.mandelmania.net](http://www.mandelmania.net/)

CONFIDENTIALITY NOTICE: This is a transmission from Ian Calegory and may contain information which is privileged, confidential and protected by the Electronic Communications Privacy Act, 18 U.S.C Section 2510 et. seq. and may also be protected by the attorney-client or attorney work product privileges. If you are not the addressee, note that any disclosure, copying, distribution, or use of the contents of this message is prohibited. If you receive this transmission in error, please destroy it and notify me immediately at 303-351-4392.