# SWIR Comparison Documentation

## What This Code Does

This code utilizes nVidia’s CUDA GPU API to compute a similarity metric between the 8 SWIR band reflectance recordings that come with each pixel in a World View 3 image and an open library of reflectance data available from USGS. The code then uses the similarity metric to reject the hypothesis of library components being present at individual pixels.

### Limitations

This code is written to be executed under the CUDA framework. At the bare minimum you need the nVidia CUDA SDK installed and a CUDA compatible graphics card to compile and run the code. You can find all the CUDA relevant downloads at nVidia’s website. As of November 2015 that we site was

<https://developer.nvidia.com/cuda-downloads>

## Algorithm Walkthrough

The algorithm that the code uses is straightforward. For each pixel we have a vector in R8.

Which we normalize:

And then remove the bias and variance:

Where the mu and sigma values come from characterizing the splib06a library.

We then normalize once more to get the final vector that we use to calculate our similarity metric.

Our similarity metric is calculated for each library element and mathematically is the dot product between this transformed pixel value and the library reflectance vector.

Once we calculate the metric values, we select the top three scoring projections for further consideration. To decide on whether or not we reject a location as possibly containing the material associated with the high scoring spectral library element, we threshold the values, and then reject all the materials whose similarity metrics fall below the threshold.

## Program Structure

This program is designed to execute the spectral rejection filter described earlier and to then display that results of the algorithmic processing via the OpenGL API.

## Program Global Members

double g\_RegX = 0.0;

double g\_RegY = 0.0;

double g\_RegScale = 1.0;

double g\_ScaleTranslate = 0;

int g\_ZoomLevel = 0;

GLuint g\_texture[2];

int g\_Width = 512 \* 2;

int g\_Height = 512 \* 2;

int g\_imagewidth = 512;

int g\_imageheight = 512;

int g\_mouseX = 0;

int g\_mouseY = 0;

bool g\_leftdown = 0;

This variables all control the display and interaction with the display window this program uses to showcase algorithmic results.

g\_RegX:

Controls the left / right location of the algorithmic showcase image within the display window.

g\_RegY:

Controls the up / down location of the algorithmic showcase image within the display window.

g\_RegScale:

Controls the magnification level of the algorithmic showcase image within the display window.

g\_ScaleTranslate:

Compensates the left, right, up, and down location of the algorithmic showcase image within the display window for Zoom levels to ensure that zooming in keeps the image centered within the display window.

g\_ZoomLevel:

Controls the g\_ScaleTranslate variable at each new magnification level.

g\_texture:

holds the device texture ids for the OpenGL textures which display in the display window

g\_Width:

Controls the display window width and the OpenGL Ortho Matrix width related values.

g\_Height:

Controls the display window height and the OpenGL Ortho Matrix height related values.

g\_imagewidth:

Sets the width of the textures used to display the algorithmic output

g\_imageheight:

Sets the height of the textures used to display the algorithmic output

g\_mouseX:

Holds the X location of the mouse in the display window.

g\_mouseY:

Holds the Y location of the mouse in the display window.

g\_leftdown:

True when the left mouse button is being held down. This is used to left the user grab the image in the display window and move it around with the mouse.

## Program Functions

### Display Related Functions

void display(void)

void keyboard(unsigned char key, int, int)

void SpecialInput(int key, int x, int y)

void MouseCallback(int button, int state, int x, int y)

void MotionCallback(int x, int y)

void reshape(int x, int y)

void startGLUT(int argc, char\* argv[])

display:

Updates the display window. Called on a window resize, keyboard interaction, or mouse interaction.

Keyboard:

Called when the display window has focus and a “normal” key is pressed on the keyboard. This function is where most of the global display variables are updated.

SpecialInput:

Called when an “abnormal” key is press when the display window has focus. For this program this function is only called when arrow keys are pressed.

MouseCallback:

Called when a mouse event occurs. In this program is it just checking to determine if the left mouse button is being held down.

MotionCallback:

Updates the g\_RegX and g\_RegY variables with the deltas of mouse movements when the left mouse button is being held down. The effect is for users to be able to “grab” the displayed image and move it around.

startGLUT:

instantiates all the objects required for the program to open a window and display an image.

### Algorithmic Functions

int main(int argc, char \* argv[]);

\_\_global\_\_ void ProcessesSWIR(unsigned short \* SWIRData0, unsigned short \* SWIRData1, unsigned short \* SWIRData2, unsigned short \* SWIRData3,

unsigned short \* SWIRData4, unsigned short \* SWIRData5,

unsigned short \* SWIRData6, unsigned short \* SWIRData7,

int imagewidth, int imageheight,

float \* ReflectanceLibrary, int ReflectanceStride,

int NumReflectanceItems,

int ReflectanceIndexStart, int ReflectanceIndexEnd,

int \* CurrentMatch, float \* CurrentScore,

int \* PMatch, float \* PScore,

int \* P2Match, float \* P2Score,

int\* LibIndices,int numLibIndices,

float \* dLibraryMoments,

bool reprocess);

main:

The programs main function that sets up the CUDA and OpenGL environments, reads data and displays the user interactive window.

ProcessSWIR:

The GPU device kernel that is executed in parallel on the device. In this program the algorithm lives here.

## How To Use This Class

### Prerequisites:

THic code require compilation via CUDA using the nvcc compiler nVidia provides.

This code requires the RPC Mapper library and the Spectral Data library available from NGA’s github repos:

RPCMapper:

<https://github.com/ngageoint/Rational-Polynomial-Coefficients-Mapper>

Spectral Library:

<https://github.com/ngageoint/Spectral-Library-Reader>

The code also requires libTIFF, tinyxml2, and freeGLUT to work:

libTIFF:

<http://bigtiff.org/libtiff.lib-4.1.zip>

tinyxml2:

<https://codeload.github.com/leethomason/tinyxml2/zip/master>

freeGLUT: <http://files.transmissionzero.co.uk/software/development/GLUT/freeglut-MSVC-3.0.0-2.mp.zip>

### Code Execution:

#### Interacting with the Display Window

Zoom Image: “+” increase magnification / “-” reduces image

Translate: a,s,w,z move in steps of +/-10 , up, down, left, and right arrows move in steps of 1

Mouse: holding down the left mouse button and dragging will move the image with the mouse.